# NetLogo 4.0.4


## Manuel de l'utilisateur

# Table des matières

# Chapitre 1

# Qu'est-ce que NetLogo ?

**NetLogo** est un environnement de modélisation programmable permettant de simuler des phénomènes naturels et sociaux. Il a été créé par Uri Wilenski en 1999 et son développement est poursuivi de manière continue par le *Center for Connected Learning and Computer-Based Modeling*.

**NetLogo** convient tout particulièrement à la modélisation de systèmes complexes évoluant au cours du temps. Les « modélisateurs » peuvent donner des instructions à des centaines ou des milliers d'« agents » opérant indépendamment les uns des autres. Ce qui permet d'explorer les liens entre les comportements des individus à leur niveau et les schémas généraux (comportements de groupe ou de masse) qui émergent des interactions entre de nombreux individus.

**NetLogo** permet aux étudiants d'ouvrir des modèles et de « jouer » avec leurs simulations pour explorer leurs comportements en faisant varier diverses conditions (paramètres). C'est aussi un environnement de programmation permettant aux étudiants, aux enseignants et aux développeurs de cours de formation de créer leurs propres modèles. NetLogo est suffisamment simple pour permettre aux étudiants et aux enseignants d'utiliser les simulations sans trop de difficultés, voire d'en concevoir eux-mêmes. Mais il est aussi assez performant pour servir d'outil puissant pour des chercheurs dans de nombreux domaines.

**NetLogo** est accompagné d'une documentation complète et de nombreux tutoriaux. Il est également livré avec une *Bibliothèque de modèles* qui comprend une grande collection de simulations fonctionnelles pouvant être utilisées telles quelles ou modifiées selon les fantaisies de l'utilisateur. Ces simulations couvrent de nombreux domaines des sciences naturelles et sociales, y compris la biologie et la médecine, la physique et la chimie, les mathématiques et l'ingénierie informatique ainsi que l'économie et la psychologie sociale. De nombreuses sessions d'apprentissage à base de modèles utilisant NetLogo sont actuellement en cours de développement.

**NetLogo** offre aussi un outil puissant de simulations participatives au niveau de la classe appelé *HubNet*. Grâce à l'utilisation d'un réseau d'ordinateurs ou de terminaux «manuels» tels que les calculatrices graphiques de Texas Instruments, chaque élève d'une classe peut contrôler en réseau l'un des agents de la simulation. Voir le chapitre HubNet pour en savoir plus.

**NetLogo** représente la nouvelle génération de toute une série de langages de modélisations multi-agents qui a débuté avec StarLogo. Il a été construit à partir des bases fournies par notre logiciel StarLogoT auquel il apporte toute une série de nouvelles fonctionnalités significatives ainsi qu'un langage et une interface utilisateur entièrement remaniés. NetLogo étant écrit en Java, il peut tourner sur tous les systèmes majeurs (Mac, Windows, Linux et autres). Il fonctionne en tant qu'application indépendante. Les modèles peuvent même être sauvegardés sous forme d'applets Java et tourner dans tous les navigateurs internet modernes.

## Fonctionnalités

Consultez la liste ci-dessous pour prendre connaissance des fonctionnalités offertes par NetLogo.

- ✔ Système :
  - ✗ Multi-plateforme : tourne sur Mac, Windows, Linux, et autres
- ✔ Langage :
  - ✗ Entièrement programmable
  - ✗ Structure du langage simple
  - ✗ Dialecte Logo étendu pour supporter les agents
  - ✗ Agents mobiles (les tortues) se déplaçant sur une grille formée d'agents stationnaires (les patches)
  - ✗ Création de liens entre les tortues pour former des agrégats, des réseaux et des graphes
  - ✗ Important vocabulaire de primitives intégrées au langage
  - ✗ Calculs en virgule flottante double précision (IEEE 754)
  - ✗ Fonctionnement absolument identique sur toutes les plateformes
- ✔ Environnement :
  - ✗ Observation des modèles en 2D ou en 3D
  - ✗ Formes vectorielles redimensionnables et pivotantes
  - ✗ Étiquetage des tortues et des patches
  - ✗ Centre de commande pour un pilotage interactif (en direct) de la simulation
  - ✗ Interface pouvant recevoir des boutons, des curseurs, des commutateurs, des traceurs de courbes, des sélecteurs, des moniteurs, des boîtes de texte, des notes, des zones de sortie
  - ✗ Variateur de vitesse pour accélérer ou ralentir la simulation
  - ✗ Système de sortie graphique puissant et souple
  - ✗ Panneau d'informations pour annoter les modèles
  - ✗ HubNet: simulation participative utilisant des machines en réseau
  - ✗ Moniteurs d'agent pour inspecter et contrôler les agents
  - ✗ Fonctions d'exportation et d'importation (exportation des données, sauvegarde et restauration de l'état d'un modèle, création et enregistrement d'une animation)
  - ✗ Outil BehaviourSpace (espace de comportements) utilisé pour collecter des données provenant de plusieurs sessions de simulations
  - ✗ Modélisation de systèmes dynamiques
- ✔ Web:
  - ✗ Modèles enregistrables sous forme d'applets pouvant ensuite être intégrés dans des pages web (note: certaines fonctionnalités ne sont pas disponibles dans les applets, telles que certaines extensions et la visualisation 3D)

# Chapitre 2

# Copyright Information



To reference this software in academic publications, please use: Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

The project gratefully acknowledges the support of the National Science Foundation (REPP and ROLE Programs) -- grant numbers REC #9814682 and REC #0126227.

## Third party licenses

### MersenneTwisterFast

For random number generation, NetLogo uses the MersenneTwisterFast class by Sean Luke. The copyright for that code is as follows:



Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- ✔ Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- ✔ Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- ✔ Neither the name of the copyright owners, their employers, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.



## Colt

Parts of NetLogo (specifically, the random-gamma primitive) are based on code from the Colt library (http://hoschek.home.cern.ch/hoschek/colt/). The copyright for that code is as follows:



## MRJ Adapter

NetLogo uses the MRJ Adapter library, which is Copyright (c) 2003-2005 Steve Roy <sroy@roydesign.net>. The library is covered by the Artistic License, http://homepage.mac.com/sroy/artisticlicense.html. MRJ Adapter is available from http://homepage.mac.com/sroy/mrjadapter/.

## Quaqua

NetLogo uses the Quaqua Look and Feel library, which is Copyright (c) 2003-2005 Werner Randelshofer, http://www.randelshofer.ch/, werner.randelshofer@bluewin.ch, All Rights Reserved. The library is covered by the GNU LGPL (Lesser General Public License). The text of that license is included in the "docs" folder which accompanies the NetLogo download, and is also available from http://www.gnu.org/copyleft/lesser.html .

## JHotDraw

For the system dynamics modeler, NetLogo uses the JHotDraw library, which is Copyright (c) 1996, 1997 by IFA Informatik and Erich Gamma. The library is covered by the GNU LGPL (Lesser General Public License). The text of that license is included in the "docs" folder which accompanies the NetLogo download, and is also available from http://www.gnu.org/copyleft/lesser.html .

## MovieEncoder

For movie-making, NetLogo uses code adapted from sim.util.media.MovieEncoder.java by Sean Luke, distributed under the MASON Open Source License. The copyright for that code is as follows:

## JpegImagesToMovie

This software is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Licensee represents and warrants that it will not use or redistribute the Software for such purposes.

## JOGL

For graphics rendering, NetLogo uses JOGL, a Java API for OpenGL. For more information about JOGL, see http://jogl.dev.java.net/. The library is distributed under the BSD license:
Copyright (c) 2003-2006 Sun Microsystems, Inc. All Rights Reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc. or the names of contributors may be used to endorse or promote products derived from this software without specific prior written permission.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that this software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

## Matrix3D

For 3D matrix operations, NetLogo uses the Matrix3D class. It is distributed under the following license:
Copyright (c) 1994-1996 Sun Microsystems, Inc. All Rights Reserved.

Sun grants you ("Licensee") a non-exclusive, royalty free, license to use, modify and redistribute this software in source and binary code form, provided that i) this copyright notice and license appear on all copies of the software; and ii) Licensee does not utilize the software in a manner which is disparaging to Sun.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This software is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Licensee represents and warrants that it will not use or redistribute the Software for such purposes.

## ASM

For Java bytecode generation, NetLogo uses the ASM library. It is distributed under the following license:



## Log4j

For logging, NetLogo uses the Log4j library. The copyright and license for the library are as follows:



## Traduction de la documentation, des exemples de modèles et de code

# Chapitre 3

# What's New ?

NetLogo 4.0.4 User Manual

Feedback from users is very valuable to us in designing and improving NetLogo. We'd like to hear from you. Please send comments, suggestions, and questions to feedback@ccl.northwestern.edu, and bug reports to bugs@ccl.northwestern.edu.

## Version 4.0.4 (November 2008)

- ✔ fixed Mac-only bug where up and down arrow keys stop working after Java update
- ✔ the speed slider now disables when view updates are disabled
- ✔ fixed bug where applets were sometimes not wide enough.
- ✔ fixed bug where `sort-by` sometimes didn't report the proper error when passed invalid inputs
- ✔ fixed extensions API bug where an extension couldn't access the drawing when running headless
- ✔ fixed extensions API bug where user-defined primitives didn't work on links by default
- ✔ fixed HubNet bug where input boxes didn't work in clients
- ✔ various fixes to HubNet plot mirroring
- ✔ fixed Gini index calculations in Wealth Distribution model
- ✔ GIS extension fixes:
  - ✗ property name input to gis:property-value is now case-insensitive
  - ✗ underscores are now allowed in dBase field names
  - ✗ fixed bug in matrix element order for gis:convolve
  - ✗ fixed bug in hemisphere clipping during inverse projection
  - ✗ fixed parsing of float field values

## Version 4.0.3 (August 2008)

- ✔ models:
  - ✗ new Computer Science models: Particle Swarm Optimization, Simple Genetic Algorithm
  - ✗ new Networks models: Diffusion on a Directed Network, Virus on a Network
  - ✗ model bugs fixed: Wealth Distribution, Turbulence, Bird Breeder, Daisyworld

- ✔ features:
  - ✗ the GIS extension for handling geographic data is now included (formerly was separate download)
  - ✗ applets now have a small "powered by NetLogo" notice on the side, linked to the NetLogo site
- ✔ engine fixes:
  - ✗ fixed bug where the neighbors primitives could report wrong values in worlds with only one row or column of patches
  - ✗ fixed bug where histograms with intervals smaller than 1 did not plot correctly
  - ✗ fixed bug where `run` and `runresult` could sometimes fail when certain pieces of code were reused
  - ✗ fixed bug where links could not use `turtles-on`
  - ✗ fixed bug where `foo-neighbor?` worked even when foo was a directed link breed
- ✔ other fixes:
  - ✗ expanded and improved documentation for GoGo extension (under Robotics/NetLogoLab)
  - ✗ minor additions and corrections to User Manual
  - ✗ lowered the memory allocation limit which caused NetLogo to fail to launch on some Windows machines
  - ✗ fixed bug where the `set-plot-pen-color` command could lock up the application
  - ✗ fixed bug where links and lines could be drawn incorrectly around view boundaries in non-square worlds
  - ✗ fixed bug where loading a model with multiple link breeds could cause an error depending on what other models you had loaded
  - ✗ fixed bug where importing world files containing drawing data was abnormally slow
  - ✗ fixed bug where some dictionary entries were not available through quick help
  - ✗ fixed bug where quick help lookups sometimes failed
  - ✗ fixed bug where the NetLogo window would jump from monitor to monitor on multiple-monitor systems
  - ✗ fixed bug where the BehaviorSpace dialog could be cut off on small monitors
  - ✗ fixed bug where the view updates checkbox would recheck itself when model code changed
  - ✗ fixed bug where selecting commands from the history menu sometimes did not return focus to the command line
  - ✗ fixed bug where the color button in a color input box was gray
  - ✗ fixed bug in HubNet view mirroring where `stamp` produced an incorrect drawing
  - ✗ fixed bug in GoGo extension where the return type of `gogo:ping` was incorrect
  - ✗ fixed 4.0-only bug where the `user-input` primitive didn't work in applets
  - ✗ fixed 4.0-only, Windows-only bug where plotting in point mode could be abnormally slow
  - ✗ fixed Mac-only bug where the font on printouts was wrong
  - ✗ fixed Mac-only bug where the link shape editor didn't work on Mac OS X 10.3

## Version 4.0.2 (December 2007)

- ✔ documentation:
  - ✗ many small fixes and improvements to User Manual
- ✔ models:
  - ✗ new evolution model: Bug Hunt Coevolution

- ✗ improved models: Climate Change (now verified), GasLab Atmosphere (bugfix), Red Queen, Bug Hunt Camouflage

✔ engine fixes:

- ✗ fixed link death bug (only models with multiple link breeds were affected)
- ✗ `sort-by` on agentsets now randomizes the order of equal agents
- ✗ `link-heading` now gives an error when the ends are on top of each other
- ✗ fixed bug where custom plot pen colors weren't compatible with `import-world`
- ✗ fixed bug in profiler extension sometimes causing nonsensical results when turtles died

✔ user interface fixes:

- ✗ 3D view now works again on Linux
- ✗ NetLogo now warns you when you open or save a model from NetLogo 3.1 or earlier, since 4.0 is not fully interoperable with earlier versions
- ✗ fixed bug where sliders didn't keep their values in bounds if the min or max changed
- ✗ fixed several slider-related bugs causing Java exceptions or unexpected behavior
- ✗ fixed bug where models with long code and many sliders compiled very slowly
- ✗ fixed bugs where you couldn't delete interface items or edit switches in the HubNet client editor
- ✗ improved appearance of dashed lines in 3D view
- ✗ improved compatibility of applets with some web browsers and operating systems

## Version 4.0 (September 2007)

✔ models:

- ✗ new earth science models: Continental Divide, Climate Change
- ✗ new chemistry model: Diprotic Acid
- ✗ new materials science model: Solid Diffusion
- ✗ new mathematics models: PANDA BEAR Solo, Surface Walking 2D
- ✗ new network model: Team Assembly
- ✗ new computer science models: Hex Cell Aggregation, Particle System Basic, Particle System Fountain, Particle System Waterfall, Particle System Flame
- ✗ new game: Planarity
- ✗ new social science models: Language Change, El Farol (old El Farol model is now named El Farol Network Congestion)
- ✗ new NIELS electromagnetism models: Ohm's Law, Series Circuit, Parallel Circuit
- ✗ new Urban Suite curricular models
- ✗ new suite of Connected Chemistry curricular models
- ✗ new suite of BEAGLE Evolution curricular models
- ✗ models improved and promoted to verified: Dice Stalagmite, Autumn, Conic Sections 2, Echo, Rebellion, Daisyworld, Sound Machines, Birthdays, Bug Hunt Speeds, Electrostatics
- ✗ other improved models: Small Worlds (bugfix), most CA 1D models (bugfix), Star Fractal (overhauled code), Genetic Drift T Interact (added optional walls), Flocking (smoother animation), Planarity (simpler code), Mimicry (simpler code), GasLab Circular Particles (cleaner code, added plots), acid/base models (simpler code)
- ✗ new Code Examples: Random Grid Walk Example, Link Lattice Example, Lattice-Walking Turtles Example, Link-Walking Turtles Example, Intersecting Links Example, State Machine Example, Breed Procedures Example, Link Breeds Example, Mouse Drag Multiple Example, Hill Climbing

Example, Rolling Plot Example, Ask-Concurrent Example, Ask Ordering Example, Random Network Example, Fully Connected Network Example, Mobile Aggregation Example, Wall Following Example, Circular Path Example, Profiler Example

- ✗ improved Code Examples: Halo Example (simplified using `tie`), Intersecting Lines Example (bugfix), RGB and HSB Example (now demonstrates RGB lists), File Output Example
- ✗ new HubNet activity: PANDA BEAR
- ✗ new HubNet code example: Template
- ✗ improved HubNet activities: Dice Stalagmite HubNet (now verified), Bug Hunters Camouflage (now verified), Root Beer Game (now verified), Disease Doctors (now verified), Minority Game

✔ documentation:

- ✗ new Transition Guide section in User Manual provides guidance on making models from previous versions work in NetLogo 4.0
- ✗ new Syntax section in Programming Guide section of User Manual
- ✗ new Applets section in User Manual

✔ features:

- ✗ most models now run somewhat faster:
  - while results vary from model to model, typical speedup is around 1.5x
  - the speedup comes from an improved compiler that now partially compiles NetLogo code into Java byte code
- ✗ links are now their own type of agent, alongside turtles and patches; this is useful for network models, geometry models, and much else
- ✗ new Link Shapes Editor lets you control the appearance of links
- ✗ logging allows researchers to record students' actions for later analysis
- ✗ built in tick counter for keeping track of model time (see language changes, below)
- ✗ new view update system:
  - there are now two view update modes, tick-based and continuous
  - continuous is the default when you start up NetLogo; tick-based is the default for most models in the Models Library; continuous update mode is useful for non-tick based models like Termites, but may also be useful during debugging
  - tick-based updates are faster for most models and avoid displaying intermediate state
  - buttons no longer have a "force view update" checkbox; most models should use `tick` and/or `display` instead
- ✗ improved speed slider:
  - you can now use the slider to "fast forward" a model (by updating the view less often)
  - in tick-based update mode, going slower pauses between ticks rather than showing intermediate states
  - in continuous update mode, going slower shows intermediate states; you can even go so slow you can watch agents move one at a time
- ✗ new input boxes in Interface tab permit entering text, numbers, colors, or NetLogo code as model parameters
- ✗ arrays and hash tables are now supported via extensions (see Arrays & Tables section of User Manual)
- ✗ new profiler extension lets you measure the running times of your procedures
- ✗ new experimental `__includes` keyword allows splitting model code into multiple source files
- ✗ color variables can now contain either a NetLogo color or an RGB color (as a list of three numbers)

- ✗ programmable slider bounds (you can now use any NetLogo reporter as the min, max, or increment of a slider)
- ✗ exporting the world now includes all plot data, and when you import a world the contents of plots are restored
- ✗ the sound extension can now play audio files in addition to MIDI sounds
- ✗ notes, monitors and output areas now have editable font size
- ✗ the color of text in notes is now editable, and a note's background may now optionally be transparent
- ✗ "Snap to Grid" in Interface tab
- ✗ added menu items (and F1 keyboard shortcut) for quickly accessing NetLogo Dictionary
- ✗ sliders may now be vertical or horizontal
- ✗ model files with unsaved changes are auto-saved to a temporary files directory, in case of freezes/crashes
- ✗ HubNet clients are much easier to edit now (the client is no longer a separate model)
- ✗ Mathematica-NetLogo link provides a real-time link between Mathematica and NetLogo, for controlling model runs and visualizing and analyzing results after the fact or in real time

✔ language changes:

- ✗ the `ask` command now always runs "without interruption"; if you need the old-style concurrent behavior, use `ask-concurrent`
- ✗ the syntax for using extensions is now simpler (no underscores, no double quotes, no .jar suffix)
- ✗ new tick counter primitives: `tick`, `ticks`, `tick-advance`, `reset-ticks`
- ✗ changes to how numbers work in NetLogo:
  - all numbers are now double precision floating point
  - numbers with no fractional part print as integers (without the decimal point)
  - much larger integers can now be represented (up to about 9x10^15)
- ✗ new primitive `of` replaces `VARIABLE-of`, `value-from`, `values-from`
- ✗ new reporter `all?` tests whether all agents in an agentset satisfy a condition
- ✗ turtle who numbers are now never reused until `clear-turtles` or `clear-all`
- ✗ new primitive `other` reports an agentset excluding the calling agent (this is short for `with [self != myself]`)
- ✗ removed `other-turtles-here` and `other-<breeds>-here` from language; use `other` instead
- ✗ new `move-to` command moves a turtle to the location of the specified turtle or patch
- ✗ for commands that create turtles or links, the commands following are now optional
- ✗ `create-custom-turtles` and `create-custom-<breeds>` no longer exist; instead, use `create-turtles` and `create-<breeds>` which now take an optional command block
- ✗ `create-turtles` now makes turtles with random headings and colors; to get evenly spaced turtles with sequential colors and id numbers use `create-ordered-turtles` (`cro` for short)
- ✗ new `turtle-set`, `patch-set`, and `link-set` reporters let you build agentsets in a variety of ways
- ✗ removed `turtles-from` and `patches-from` from language (use `turtle-set` and `patch-set` instead)
- ✗ new `uphill`, `uphill4`, `downhill`, and `downhill4` commands for doing hill-climbing
  - these new commands replace the old reporters with the same names
  - the new commands have somewhat different semantics that are less prone to problems
  - models using the old reporters will require hand changes
- ✗ new reporters `no-turtles`, `no-patches`, and `no-links` report empty agentsets

- ✗ two agentsets can now be tested for equality
- ✗ the `tie` and `untie` commands are no longer experimental; they now take no inputs and can only be used by links; two tie modes are available, "fixed" and "free"
- ✗ the + operator only adds numbers now; it doesn't work on strings or lists anymore; models must be changed by hand to use `word` instead for strings and `sentence` instead for numbers
- ✗ new `histogram` command replaces old `histogram-list`, `histogram-from` commands
- ✗ removed `random-int-or-float` from the language; some models may need to be changed by hand to use `random` or `random-float` instead
- ✗ removed `nsum` and `nsum4` from language; use `sum [reporter] of neighbors/neighbors4` instead
- ✗ new agentset primitives `min-n-of` and `max-n-of`
- ✗ new primitive `with-local-randomness` runs code without altering the state of the random number generator
- ✗ new file I/O primitive `file-flush` forces output to disk
- ✗ new color primitive `base-colors` reports a list of the 14 basic NetLogo hues
- ✗ the `turtle` primitive no longer accepts non-integer inputs
- ✗ the `patch` primitive now accepts non-integers and rounds them, and also wraps if allowed
- ✗ the observer may no longer use `patch-at`, `turtles-at`, and `BREED-at`; use `patch`, `turtles-on patch`, and `BREED-on patch` instead
- ✗ comparison operators now work on turtles, patches, and links
- ✗ new primitive reporter `plot-pen-exists?`
- ✗ old `rgb` and `hsb` primitives renamed to `approximate-rgb` and `approximate-hsb`; they now expect inputs in 0-255 range instead of 0-1.0
- ✗ the `hsb` and `rgb` reporters now report RGB lists instead of NetLogo colors
- ✗ new primitive `import-pcolors-rgb` imports images into the patches as RGB colors
- ✗ new reporter `netlogo-applet?` lets you test whether the model is running as an applet or not
- ✗ code may now set a slider to values which violate the minimum, maximum, or increment
- ✗ `locals` no longer exists; use `let` instead
- ✗ extension primitives must now (by default) be referred to using the extension name, e.g. `sound:drums` instead of just `drums`
- ✗ the `file-read` primitive now skips over comments
- ✗ got rid of many undocumented alternate names for primitives

✔ user interface changes:

- ✗ "note" is the new name for the widget formerly known as "text box"
- ✗ sliders may now be moved using the mouse scroll wheel
- ✗ built-in variables are now syntax-colored the same purple as primitive reporters
- ✗ dialogs associated with the `user-*` primitives have more consistent appearance and functionality
- ✗ contextual menus added to text areas (for cut/copy/paste and dictionary lookup)
- ✗ Pens button removed from plots (you now edit the plot to show or hide the pens legend)

✔ engine fixes:

- ✗ monitors now use an auxiliary random generator, so code in monitors won't affect model run reproducibility
- ✗ the `run` and `runresult` primitives are now drastically faster when called repeatedly on the same string
- ✗ the `display` command now works even when used by a turtle, patch, or link
- ✗ internally, lists are now linked lists, not arrays; this does not affect the behavior of models, but

may affect performance (positively or negatively, depending on what list operations you are doing; see Programming Guide)
- ✗ when turtles move in a way that does not indicate direction (e.g. `setxy`), the shortest path is now always drawn (even if it wraps)
- ✗ fixed bug where in some conditions, exporting and then re-importing the world during a model run could alter the outcome (by affecting how who numbers were reused)
- ✗ `sort-by` on lists is now stable (that is, it does not disturb the existing order of equal items)
- ✗ the `file-read` primitive is now much faster at reading very long lists
- ✗ fixed bugs where `in-cone` and `distance` didn't work properly in some topologies
- ✗ fixed bug where some layout commands were not controlled by the random seed
- ✗ the global variables associated with sliders, switches, choices, and input boxes now behave the same when running headless as in the GUI, that is, they reject values that are of the wrong type or out of range

✔ other fixes:
- ✗ the Halt item on the Tools menu now works in more situations (rather than hanging NetLogo)
- ✗ if endless recursion occurs, NetLogo now reports the error instead of bombing
- ✗ while startup commands run, the user is now prevented from interacting with the model
- ✗ plotting is now supported when running headless (use `export-plot` or `export-world` to save plotted data for later perusal)
- ✗ in the 3D view, turtle shapes now wrap around the edges if the world has wrapping enabled
- ✗ upgraded JOGL, fixing 3D view bugs for some users
- ✗ applets can now find files associated with the model even if the model file is in a different location than the HTML file
- ✗ in the system dynamics modeler, rate connectors can now be repositioned
- ✗ the system dynamics modeler now shows the location of syntax errors
- ✗ in the system dynamics modeler, you can now control which plot is used using `set-current-plot`, and which stocks are plotted is controlled by `plot-pen-exists?`
- ✗ default maximum Java heap size raised to one gigabyte
- ✗ headless mode now works with IBM's Java
- ✗ the GoGo extension is now easier to use (no separate installation steps required)
- ✗ new checkbox allows you to turn off the world wire frame in the 3D view

✔ extensions API changes:
- ✗ the extensions API now has a version number, so extensions can be used with different NetLogo versions as long as the API version number hasn't changed
- ✗ instead of being just a jar, an extension is now a directory containing a jar, so that the directory can contain other supporting files
- ✗ there is now rudimentary support in the API for extensions to define new data types (the array and table extensions are examples)
- ✗ the random number generator is now accessible by extensions
- ✗ sample extensions are now installed with Java source code included (formerly it was separate downloads)

## Version 3.1.5 (December 2007)

✔ installer supports Windows Vista

- ✔ rewritten and expanded Tutorial #3
- ✔ new sound extension command `play-note-later` for playing musical phrases
- ✔ extensions not requiring additional jars work in applets
- ✔ `sort-by` on agentsets now randomizes the order of equal agents
- ✔ bugfixes

## Version 3.1 (April 2006)

- ✔ topologies (wrapping at world edges now optional)
- ✔ automatically randomized ordering of agentsets
- ✔ you now specify the singular form of a breed name as well as the plural
- ✔ `sort` and `sort-by` work on agentsets now
- ✔ link primitives for network and geometry models (experimental)
- ✔ `__tie` and `__untie` primitives (experimental)

## Version 3.0 (September 2005)

- ✔ 3D view (for 2D models)
- ✔ System Dynamics Modeler
- ✔ `follow`, `ride` and `watch` commands for tracking particular agents
- ✔ "drawing layer" for marks left by turtles
- ✔ more attractive colors
- ✔ more attractive Information tab
- ✔ GoGo extension for interfacing NetLogo with physical devices
- ✔ Color Swatches dialog helps you choose colors
- ✔ image file importing
- ✔ buttons take turns now (instead of interleaving their code with each other)

## Version 2.1 (December 2004)

- ✔ "headless" mode for command line operation
- ✔ editor highlights matching parentheses and brackets
- ✔ "action keys" let buttons be triggered by keypresses
- ✔ makes QuickTime movies of models
- ✔ add "output area" to models
- ✔ improved shapes editor and built-in shapes
- ✔ new primitives including `let` and `carefully`
- ✔ computer HubNet:
  - ✗ now more reliable
  - ✗ clients automatically find server
  - ✗ improved client interface and Control Center

## Version 2.0.2 (August 2004)

- ✔ extensions API for writing commands and reporters in Java
- ✔ controlling API for controlling NetLogo from external Java code
- ✔ sound extension for making sounds and music

## Version 2.0 (December 2003)

- ✔ full support for Mac OS X and Linux
- ✔ Windows 95, MacOS 8/9 no longer supported
- ✔ improved look and feel
- ✔ fast, flicker-free, non-grid-based graphics
- ✔ primitives for reading and writing external files
- ✔ strict math mode for identical results cross-platform
- ✔ export view or interface tab as image file
- ✔ improved BehaviorSpace
- ✔ computer HubNet (no longer alpha)

## Version 1.3 (June 2003)

- ✔ view control strip
- ✔ choosers
- ✔ new primitives including `run`, `runresult`, `map`, `foreach`, `filter`, `reduce`
- ✔ some primitives accept a variable number of inputs

## Version 1.2 (March 2003)

- ✔ much improved speed
- ✔ computer HubNet (alpha)

## Version 1.1 (July 2002)

- ✔ "Save as Applet" lets you embed your model in any web page
- ✔ printer support
- ✔ Procedures menu
- ✔ scrollable Interface tab

## Version 1.0 (April 2002)

- ✔ initial release (after a series of betas)

# Chapitre 4

# Configuration minimale

NetLogo tourne sur à peu près tous les ordinateurs courants.

Si vous avez des problèmes parce que NetLogo ne fonctionne pas, écrivez à bugs@ccl.northwestern.edu.

## Configuration minimale : Application

### Windows

NetLogo tourne sur Windows Vista, XP, 2000, NT, ME et 98.

L'installateur de NetLogo pour Windows installe la version 1.5.0 de Java pour son usage exclusif. Les autres programmes de l'ordinateur ne sont pas concernés.

### Mac OS X

Mac OS X 10.4 (ou plus) est fortement recommandé. Les versions 10.3 and 10.2 sont aussi supportées.

Effectuez une mise à jour du système (Menu "Pomme", "Mise à jour de logiciels…") pour être sûr d'avoir la dernière version de Java.

### Autres plateformes

NetLogo devrait fonctionner sur toute plateforme sur laquelle la Machine Virtuelle Java de Sun, version 1.4.1 ou plus récente, est installée. La version 1.5.0_13 (ou plus) est recommandée.

Vous démarrez NetLogo en lançant le script `netlogo.sh` fourni.

## Configuration minimale : NetLogo en applets

Les modèles NetLogo sauvegardés sous forme d'applets Java devraient fonctionner sur tout navigateur utilisant une version 1.4.1 ou plus récente de Java.

## Configuration minimale : Vue 3D

Parfois, un système plus ancien, moins puissant, ne peut pas faire fonctionner correctement la Vue 3D. Essayez et constatez.

Certains systèmes peuvent utiliser la Vue 3D mais ne peuvent commuter en mode plein écran. Cela dépend de la carte graphique et du contrôleur. Par exemple, les cartes ATI Radeon IGP 345 et Intel 82845 ne fonctionneront probablement pas.

## Détails techniques Java pour les utilisateurs de Windows

**La majorités des utilisateurs de Windows devraient choisir le téléchargement NetLogo qui comprend le paquet Java.**

Il y a deux raisons pour lesquelles vous pourriez vouloir utiliser le téléchargement qui ne comprend pas le paquet Java :

- ✔ Vous voulez télécharger un petit paquet de manière à ce qu'il arrive plus rapidement et occupe moins de place sur le disque.
- ✔ Pour des raisons techniques particulières qui vous sont propres, vous voulez faire tourner NetLogo avec un Java autre que celui fourni d'office.

Si vous pensez que le téléchargement alternatif répond à vos besoins, lisez s'il-vous-plaît les informations techniques détaillées ci-dessous.

**Même si Java est déjà installé sur votre machine, il ne pourra vraisemblablement *pas* fonctionner avec NetLogo**

Pour avoir une efficacité maximale, NetLogo utilise une option spéciale appelée "`server`" VM. L'installateur du Java Runtime Environment (JRE) de Sun n'a pas et donc n'installe par cette option par défaut. Elle ne se trouve que dans le Java Development Kit (JDK) de Sun.

Si vous n'êtes pas un développeur Java, vous avez probablement le JRE, pas le JDK.

Toutefois, si vous voulez faire tourner NetLogo avec votre propre machine virtuelle Java, vous avez deux options :

1. Assurez-vous bien d'avoir le JDK complet pour le Java que vous voulez utiliser, et pas seulement le JRE.
2. Ou éditez le fichier de configuration de manière à faire fonctionner NetLogo avec le JRE.

Nous ne recommandons pas la seconde option car NetLogo fonctionne nettement plus lentement sans l'option "`server`".

Si, malgré tout, vous choisissez la seconde option, voilà ce qu'il faut faire. Vous devez dire à NetLogo de ne pas essayer d'utiliser l'option "`server`" VM. Tout d'abord, installez NetLogo en utilisant le téléchargement sans Java. Puis ouvrez le fichier `NetLogo 4.0.4.lax` situé dans le répertoire où vous avez installé NetLogo (par défaut `c:/Program Files/NetLogo 4.0.4`) avec un éditeur de texte tel que Notepad. Vous devez effacer l'option `serveur` des options java additionnelles. Donc, modifiez la section

```
#   LAX.NL.JAVA.OPTION.ADDITIONAL
#   ----------------------------
#   don't load native libs from user dirs, only ours, also run server not client VM

lax.nl.java.option.additional=-Djava.ext.dirs= -server -Dsun.java2d.noddraw=true
```

pour qu'elle ressemble à

```
#   LAX.NL.JAVA.OPTION.ADDITIONAL
#   ----------------------------
#   don't load native libs from user dirs, only ours, also run server not client VM

lax.nl.java.option.additional=-Djava.ext.dirs= -Dsun.java2d.noddraw=true
```

Encore une fois, rappelez-vous que cette approche dégrade nettement les performances de NetLogo.

# Chapitre 5

# Problèmes connus

Si NetLogo fonctionne mal, envoyez-nous s'il-vous-plaît un rapport de bug. Voyez le chapitre Pour nous contacter.

## Problèmes connus (tous les systèmes)

### Bugs du langage/moteur

- Les extensions Table et Array ne sont que partiellement compatibles avec la fonctionnalité d'importation des mondes. Quand vous exportez un monde NetLogo (avec la commande `export-world` ou la commande de menu "Export World"), les tableaux et les tables de hachage sont exportés « par valeurs ». Cela signifie que si vous avez la même table ou le même tableau placé à plus d'un endroit dans le code, alors, quand elle/il sera exporté puis ré-importé, il y aura des tables ou des tableaux distincts à chacun des endroits où il y avait les tables ou les tableaux originaux. Ces copies auront au départ les mêmes valeurs, mais si l'une des copies est modifiée par la suite, l'autre ne le sera pas.
- `export-world` ne sauvegarde pas l'état des fichiers qui ont été ouverts en utilisant l'une des primitives de manipulations de fichiers. Si vous avez ouvert un fichier de cette manière puis exporté le monde, vous devrez ré-ouvrir le fichier avant de continuer avec la simulation quand vous importerez ce monde dans NetLogo,.
- Un bug de Java fait que les couleurs des patches importées avec `import-pcolors` sont plus claires que les couleurs originales si l'image originale a une palette de gris. Pour contourner ce problème, convertissez l'image contenue dans le fichier en une image utilisant une palette RGB.

### Autres bugs

- Les situations de débordement de mémoire ne sont pas gérées avec élégance.
- La fonctionnalité "Snap to Grid" est désactivée quand on fait un zoom avant ou arrière.
- Dessiner puis effacer une ligne avec la tortue dans la couche de dessin n'efface pas toujours au pixel près.
- Les extensions qui font appel à des jars externes supplémentaires ne fonctionnent pas pour les modèles qui on été sauvegardés sous forme d'applets (nous sommes déjà entrain de voir comment résoudre ce problème).

- La Vue 3D ne fonctionne pas avec certaines configurations graphiques; avec d'autres, la Vue 3D fonctionne, mais pas en mode plein écran.
- Quand il tourne en mode « headless » (à partir de la ligne de commande) et s'il a été créé avec une version antérieure de NetLogo, un modèle peut ne pas fonctionner correctement. Utilisez l'interface utilisateur (GUI) pour ouvrir le modèle avec la version courante de NetLogo puis enregistrez-le avant de le faire tourner « headless ».

## Bugs « Windows-only »

- Sur certaines machines, les primitives `distance`, `in-radius` et consorts retournent parfois de mauvaises réponses dues à un bug dans Java. Consultez la FAQ pour voir comment résoudre ce problème.
- La commande "User Manual" du menu "Help" ne fonctionne pas sur tous les ordinateurs (Windows 98 et ME sont les plus touchés, les nouvelles versions de Windows, moins).
- Sur certains ordinateurs portables, l'affichage des panneaux "Information" et "Procedures" peut être perturbé quand on le fait défiler. Pour éviter ce bug, réduisez la taille de la fenêtre NetLogo et/ou réduisez les couleurs de votre écran (par exemple passez d'un affichage couleurs 32-bit à un affichage couleurs 16-bit, voire 8-bit). C'est un bug de Java, pas de NetLogo. Pour les détails techniques concernant ce bug, consultez `http://developer.java.sun.com/developer/bugParade/bugs/4763448.html` (enregistrement gratuit nécessaire). Nous incitons les utilisateurs de NetLogo à visiter ce site et à demander à Sun de corriger ce bug.

## Bugs « Macintosh-only »

- Quand on ouvre un modèle à partir du Finder (en double-cliquant sur son icône ou en déposant son icône sur l'icône NetLogo) et que NetLogo ne tourne pas encore, il est possible que le modèle ne s'ouvre pas; ce bug est intermittent. (Si NetLogo tourne déjà, le modèle s'ouvre à tous les coups.)
- Avec Mac OS X 10.4 seulement, les commandes "Copy View" et "Copy Interface" peuvent ne pas fonctionner : l'image obtenue est déformée. La solution consiste à utiliser les commandes "Export -> View" et "Export -> Interface" du menu "File". Ce problème disparaît si vous mettez à jour vos logiciels pour obtenir le dernière version Java de Apple.
- Avec les versions Mac OS X antérieures à 10.4, il est possible que les menus de NetLogo s'embrouillent et que la commande "Quit" ne fonctionne plus. Si cela arrive, vous pouvez quitter NetLogo en cliquant le bouton rouge situé à gauche de la barre de titre de la fenêtre NetLogo.

## Bugs « Linux/UNIX-only »

- Le Manuel de l'utilisateur s'ouvre toujours dans Mozilla et non avec la navigateur par défaut. Une solution possible consiste à faire connaître le fichier `docs/index.html` à votre navigateur préféré. Une autre solution est de créer une symlink qui est appelée "`mozilla`" (c'est le nom de la commande que NetLogo tente d'exécuter), mais qui en réalité fait fonctionner un autre navigateur.
- Nous avons découvert un problème avec Linux où le reporter `exp` retourne parfois une réponse légèrement différente (la différence ne porte que sur la dernière décimale) pour la même entrée. D'après un ingénieur de chez Sun, cette erreur ne devrait se produire qu'avec le noyau (*kernel*) Linux 2.14.19 ou plus ancien, mais nous avons aussi observé ce problème avec des noyaux plus récents. Nous pensons donc que ce problème est spécifique à Linux et ne devrait pas apparaître avec les autres systèmes à base UNIX. Nous ne sommes pas sûrs si le problème apparaît toujours en pratique durant une vraie

simulation d'un modèle NetLogo ou s'il ne survient que dans le contexte des procédures de tests que nous faisons subir à NetLogo. Le bug se trouve dans la machine virtuelle Java (Java VM) de Sun et non dans NetLogo lui-même. Nous espérons que seul le reporter `exp` souffre de ce problème, mais nous ne pouvons pas en être certains. Nous encourageons les utilisateurs de NetLogo à visiter le site `http://developer.java.sun.com/developer/bugParade/bugs/5023712.html` (enregistrement gratuit nécessaire) et à demander à Sun de corriger ce bug.

- ✔ Si NetLogo ne peut pas trouver la police "Lucida", les menus sont illisibles. Nous savons que cette erreur survient avec Fedora Core 3, après mise à jours des paquets. Le redémarrage du X Font Server (xfs) a résolu le problème dans tous les cas qui nous ont été signalés.
- ✔ L'exécutable Java 1.5.0 de Sun a des problèmes d'affichage avec GTK 2.0 et NetLogo. Ces problèmes peuvent comprendre un mauvais rafraîchissement des fenêtres, des éléments d'interfaces avec des dimensions bizarres, des menus tronqués à la base et des caractères étranges qui apparaissent dans la vue. Pour éviter ces bizarreries, passez à Java 1.6.

## Problèmes connus avec Computer HubNet

Consultez le Guide HubNet pour la liste des problèmes connus avec Computer HubNet.

# Chapitre 6

# Pour nous contacter

Les retours d'informations de la part des utilisateurs nous sont très précieux pour nous aider (guider) dans la conception et le développement des versions futures de NetLogo. Nous aimerions avoir vos avis.

## Site web

Notre site web, ccl.northwestern.edu vous donne notre adresse e-mail et notre numéro de téléphone. Il contient également des informations concernant notre équipe et nos diverses activités de recherches.

## Retours d'informations, questions, etc.

Si vous avez besoin d'aide pour votre modèle, entrez en contact et posez vos questions au groupe d'utilisateurs de NetLogo à l'adresse http://groups.yahoo.com/group/netlogo-users/.

Nous avons également un groupe http://groups.yahoo.com/group/netlogo-educators/ spécialement destinés aux enseignants .

Si vous avez des informations en retour, des suggestions ou des questions, écrivez nous à l'adresse feedback@ccl.northwestern.edu.

## Rapport de bugs

Si vous voulez nous faire part d'un bug rencontré en utilisant NetLogo, écrivez à bugs@ccl.northwestern.edu. Quand vous nous soumettez un tel rapport, essayez d'y mettre le plus d'informations possibles concernant les sujets suivants :

- ✔ Une description complète du problème et de la manière dont il est apparu.
- ✔ Le modèle NetLogo ou le code qui vous a posé problème. Si possible, attachez le modèle complet à votre courrier électronique.
- ✔ Les informations concernant votre système : version de NetLogo, Système d'exploitation et sa version, version de Java, etc. (Ces informations peuvent être trouvées dans le panneau "System" de la fenêtre "About NetLogo" que l'on ouvre avec la commande de menu "About NetLogo". Dans les applets tournant dans un navigateur, les mêmes informations sont disponibles en faisant un Ctrl-clic (Mac) ou un clic-droit dans l'arrière-plan blanc de l'applet).
- ✔ Tous les messages d'erreurs qui ont été affichés.

# Chapitre 7

# Exemple de modèle : La réception

Cette activité a été conçue pour vous faire réfléchir à la modélisation par ordinateur et à la manière de l'utiliser. Il vous donne aussi idée du fonctionnement interne du logiciel NetLogo. Nous encourageons les utilisateurs débutants de commencer par cette activité.

## Au cours d'une réception

Avez-vous déjà participé à une réception (*party*) et observé comment les gens se ressemblent en groupes? Vous avez peut-être aussi remarqué qu'ils ne restent pas dans un groupe mais se déplacent d'un groupe à l'autre. Tant que les individus se déplacent, les groupes changent. Si vous observez ces changements au cours du temps, vous allez remarquer que ces échanges obéissent à certains schémas.

Par exemple, dans les relations sociales, les gens ont tendance à avoir des comportements différents quand ils sont au travail et quand ils sont à la maison. Des individus extravertis dans leur environnement de travail deviennent renfermés et timides dans des réunions publiques. Et d'autres si tranquilles et réservés au travail deviennent des « meneurs » quand ils sont avec leurs amis.

Les schémas observés peuvent aussi dépendre du genre de réunion. Dans certaines situations, les gens sont entraînés à s'organiser d'eux-mêmes en groupes mixtes, par exemple lors d'activités ludiques ou de type scolaire. Mais dans une atmosphère non structurée, ils ont tendance à se regrouper de manière beaucoup plus aléatoire.

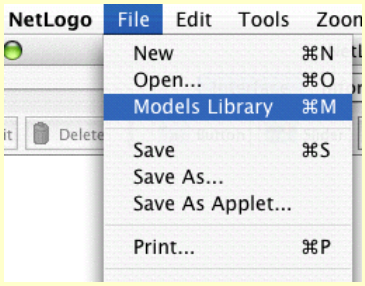
➔ ***Existe-t-il un schéma type pour ce genre de regroupements ?***

Examinons cette question de plus près en utilisant l'ordinateur pour modéliser le comportement humain au cours d'une réception. Le modèle " Party " de NetLogo s'intéresse plus spécialement au regroupement par sexe au cours des réceptions : pourquoi les groupes ont-ils tendance à n'être formés, certains presque que d'hommes et d'autres presque que de femmes?

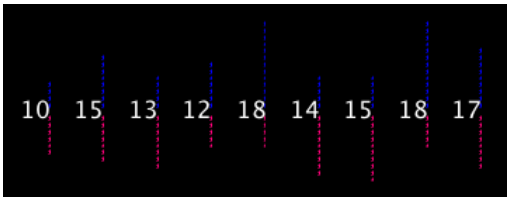
Utilisons NetLogo pour tenter de répondre à cette question.

**Marche à suivre :**

- Démarrez NetLogo.
- Sélectionnez " Models Library " dans le menu " File ".
- Ouvrez le dossier " Social Science ".
- Cliquez sur le modèle appelé " Party ".
- Pressez le bouton " open ".
- Attendre la fin du chargement de la simulation.
- (en option) Agrandir la fenêtre NetLogo afin de pouvoir tout voir.
- Pressez le bouton " setup ".



Dans la Vue, vous verrez apparaître des lignes roses et bleues accompagnées de nombres :




Ces lignes représentent les groupes mixtes formés lors de la réception. Les hommes sont représentés en bleu et les femmes en rose. Les nombres indiquent le total d'individus dans chaque groupe.

➔ ***Les groupes sont-ils tous formés d'à peu près le même nombre de personnes?***

➔ ***Les groupes ont-ils tous à peu près le même nombre d'individus de chaque sexe?***

Supposons que vous organisiez une réception et invitiez 150 personnes. Vous vous demandez comment vos invités vont se regrouper. Supposons qu'il se formera 10 groupes.

➔ ***Comment pensez-vous que les participants vont se regrouper?***

Plutôt que de demander à vos 150 plus proches amis de se réunir et de former des groupes au hasard, laissons l'ordinateur simuler cette situation.

**Marche à suivre :**

- Pressez le bouton " go ". (Presser " go " une seconde fois stoppe la simulation.)
- Observez les déplacements des gens jusqu'à ce que la simulation s'arrête.
- Observez les barres pour voir ce qui se passe d'un autre point de vue.

➔ ***Combien y a-t-il maintenant de personnes dans chaque groupe?***

Au départ, vous auriez pu penser que, 150 personnes ayant à se répartir en 10 groupes, le résultat serait d'environ 15 personnes dans chaque groupe. En consultant la simulation, nous voyons que les gens ne se répartissent pas régulièrement dans les 10 groupes – au contraire, certains groupes deviennent très petits alors que d'autres deviennent très grands. De plus, au cours du déroulement de la réception, on passe de groupes tous mixtes à des groupes unisexes.

➔ ***Comment expliquer ces résultats?***

La question de savoir ce qui se passe au cours de réceptions réelles peut recevoir toute une série de réponses possibles. Le concepteur de cette simulation part simplement du principe que les groupes ne se forment pas de manière totalement aléatoire. La composition des groupes dépend de la manière dont les individus se comportent au cours de la réception. Le concepteur a choisi de se focaliser sur un paramètre (une variable) particulier appelé « tolérance » :


La tolérance est définie ici comme le pourcentage de personnes de sexe opposé avec lesquelles un individu est « encore à l'aise ». Si un individu se trouve dans un groupe comportant un pourcentage de personnes du sexe opposé supérieur à sa tolérance, il ne se sent « plus à l'aise » et quitte le groupe pour en rejoindre un autre.

Par exemple, si le niveau de la tolérance est fixé à 25%, alors les hommes ne se sentent à l'aise que dans les groupes comportant moins de 25% de femmes et les femmes ne sont à l'aise que dans les groupes ne comportant que moins de 25% d'hommes.

Quand des individus ne se sentent « plus à l'aise » et quittent leur groupe, ils rejoignent d'autres groupes, ce qui a parfois comme conséquence que certaines personnes présentes dans ces groupes ne se sentent à leur tour « plus à l'aise » et quittent le groupe. Cette réaction en chaîne continue jusqu'à ce que chacun, dans la réception, se sente « à l'aise » dans son groupe.

Notez que, dans ce modèle, la « tolérance » n'est pas fixée. Vous, l'expérimentateur, pouvez utiliser le curseur " tolerance " pour tester différentes valeurs de ce paramètre et voir ce qu'il advient quand vous recommencez la simulation.

**Comment relancer la simulation :**

- Si le bouton " go " est enfoncé (noir), la simulation tourne encore. Pressez ce bouton pour l'arrêter.
- Placez le curseur " tolerance " sur une nouvelle valeur en tirant la poignée rouge.
- Pressez le bouton " setup " pour réinitialiser le modèle.
- Pressez le bouton " go " pour relancer la simulation.

## Défi

En tant qu'organisateur de la réception, vous aimeriez voir les hommes et les femmes se mélanger au sein des groupes. Ajustez le curseur de tolérance placé sur le côté de la Vue jusqu'à ce que vous arriviez à ce que tous les groupes soient mixtes à la fin de la réception.

➔ ***Quel doit être le niveau de tolérance afin d'être sûr que les 10 groupes soient mixtes?***

Testez vos prédictions à l'aide du modèle.

➔ ***Pouvez-vous imaginer d'autres facteurs ou d'autres variables pouvant influencer le rapport hommes/femmes au sein de chaque groupe?***

Faites des prédictions et testez vos idées à l'aide du modèle. Sentez-vous libre de modifier plus d'une variable à la fois.

Pendant que vous testez vos hypothèses, vous constaterez que des schémas émergent petit à petit des données récoltées. Par exemple, si vous gardez constant le nombre de participants à la réception mais que vous accroissez graduellement le niveau de tolérance, un plus grand nombre de groupes mixtes apparaît.

➔ ***Quel est le niveau de tolérance atteint juste avant que vous obteniez des groupes mixtes?***

➔ ***Établissez une relation entre pourcentage de tolérance et pourcentage de mixité.***

## Réfléchir à l'aide de modèles

Utiliser NetLogo pour modéliser puis simuler des situations telles que ce scénario de réception permet d'expérimenter rapidement et en souplesse un système, démarche qu'il serait difficile de réaliser dans une situation réelle. La modélisation et la simulation vous donnent aussi l'opportunité d'observer une situation ou une circonstance avec moins de préjudices – car vous pouvez examiner les dynamiques sous-jacentes à la situation. Vous constaterez que plus vous progressez dans vos modélisations-simulations, plus vos idées préconçues au sujet de différents phénomènes seront remises en question. Par exemple, un résultat surprenant du modèle " Party " est que même si la tolérance est relativement élevée, une grande tendance à la séparation des sexes persiste.

Cette simulation est un exemple classique du phénomène d'**émergence**, où un schéma de groupe (comportement de groupe) résulte de l'interaction d'un grand nombre d'individus. Ce concept de phénomène **émergent** peut être appliqué à pratiquement n'importe quel sujet.

➔ ***À quel autre phénomène émergeant pouvez-vous penser?***

Pour étudier d'autres exemples, obtenir une meilleure compréhension de ce concept et voir comment NetLogo peut aider les étudiants à l'explorer, nous vous conseillons d'explorer la Bibliothèque des Modèles de NetLogo. Elle contient des modèles qui démontrent ces idées appliquées à toutes sortes de systèmes.

Pour une plus longue discussion au sujet de l'émergence et de la manière dont NetLogo peut aider l'étudiant à l'explorer, voyez "Modeling Nature's Emergent Patterns with Multi-agent Languages" (Wilensky, 2001).

## Et ensuite?

La section du Manuel de l'utilisateur appelée Tutoriel 1 : Utiliser les modèles entre plus en détails dans la manière d'utiliser les autres modèles de la Bibliothèque des modèles (Models Library).

Si vous voulez apprendre comment explorer les modèles à un niveau plus profond, le Tutoriel 2 : Les commandes vous donnera une introduction au langage de modélisation de NetLogo.

En fin de compte, vous serez prêts pour le Tutoriel 3 : Les procédures, où vous apprendrez comment modifier et compléter les modèles existants pour leur donner de nouveaux comportements, et finalement construire vos propres modèles.

# Chapitre 8

# Tutoriel 1 : Les modèles

Si vous avez lu la section Exemple de modèle : La réception, vous avez eu une bref aperçu de ce qu'est une interaction avec un modèle NetLogo. Cette section va bien plus loin dans l'étude des fonctionnalités mises à votre disposition pour explorer les modèles de la Bibliothèque des modèles.

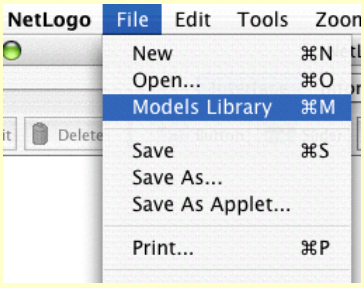
Dans tous ces tutoriels, il vous sera demandé de faire des prédictions sur les effets produits par les modifications apportées aux modèles. Gardez à l'esprit que ces effets sont souvent surprenants. Nous pensons que ces surprises sont stimulantes et offrent d'excellentes opportunités d'apprentissage.

Certaines personnes ont trouvé utile d'imprimer les tutoriels avant de les travailler. Quand les tutoriels sont imprimés, il y a plus de place sur l'écran de l'ordinateur pour le modèle NetLogo avec lequel vous êtes en train de travailler.

## Exemple de modèle : Prédation loups-moutons

Nous allons ouvrir l'un des exemples de modèles pour l'explorer en détails. Essayons un modèle biologique: la prédation loups-moutons, un modèle simulant les interactions prédateurs–proies entre une population de loups et une population de moutons.

➢ Ouvrez " Models Library " dans le menu " File ".




➢ Sélectionnez le modèle " Wolf Sheep Predation " dans la section " Biology " et pressez " Open ".

Le panneau "Interface" se remplit de tout un tas de boutons, de commutateurs, de curseurs et de moniteurs. Ces éléments de l'interface vous permettent d'interagir avec le modèle. Les boutons sont bleus : ils initialisent, démarrent et stoppent la simulation. Les curseurs et les commutateurs sont verts : ils modifient les réglages du modèle. Les moniteurs et les traceurs de courbes sont beiges : ils affichent des données.

Si vous voulez agrandir la fenêtre pour avoir une meilleure vision de tous ces éléments, vous pouvez utiliser les boutons de zoom de fenêtre habituels propres à chaque système d'exploitation.

Quand vous ouvrez le modèle, vous constatez que la Vue est vide (entièrement noire). Pour pouvoir commencer la simulation, vous devez d'abord l'initialiser.

- Pressez le bouton " setup ".

➔ ***Que voyez-vous apparaître dans la Vue?***

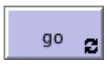
- Pressez le bouton " go " pour lancer la simulation.

➔ ***Que se passe-t-il avec les populations de loups et de moutons pendant la simulation?***

- Pressez le bouton " go " pour stopper la simulation.

## Contrôler la simulation : les boutons

Quand un bouton est pressé, le modèle répond par une action. Un bouton peut être un bouton « une fois » ou un bouton « pour toujours ». Vous pouvez différencier ces deux types de boutons grâce au symbole qui figure dans leur coin inférieur droit. Les boutons « pour toujours » sont signalés par deux flèches courbes, comme ici :




Les bouton « une fois » n'ont pas de telles flèches, comme ce bouton-ci :


Les boutons « une fois » déclenchent une action puis stoppent. Quand l'action est terminée, le bouton remonte en position déclenché.

Les boutons « pour toujours » déclenchent une action encore et encore. Quand vous voulez interrompre la simulation, pressez encore une fois ce bouton. L'action en cours se termine et le bouton remonte en position déclenché.

La plupart des modèles, y compris le modèle Prédation Loups-Moutons ont un bouton « une fois » appelé "setup" et un bouton « pour toujours » appelé "go". Plusieurs modèles possèdent aussi un bouton « une fois » appelé "go once" ou "step once" qui agit comme "go" sauf qu'il ne fait progresser (avancer) le modèle que d'un pas (d'une unité de temps) à la fois. L'utilisation d'un tel bouton permet d'observer la progression de la simulation de manière bien plus fine.

Déclencher un bouton « pour toujours » est la manière normale de stopper une simulation. Il n'y a aucun risque de mettre en pause une simulation en pressant le bouton « pour toujours » puis de la relancer en pressant encore une fois ce bouton car la simulation reprend là où elle avait été interrompue. Vous pouvez aussi stopper une simulation en pressant la commande "Halt" du menu "Tools", mais vous ne devriez le faire que si votre simulation se fige (plante) pour une raison ou une autre. La commande "Halt" peut interrompre la simulation au milieu d'une action mais il est alors fort probable que le modèle ne se comportera plus correctement si on reprend ensuite la simulation.

- Si vous en avez envie, testez l'effet des boutons " setup " et " go " dans le modèle Prédation Loups-Moutons.

➔ ***Obtenez-vous toujours des résultats différents si vous démarrez plusieurs fois la simulation avec les mêmes réglages initiaux?***

## Contrôler la vitesse de simulation : le curseur "Speed" (vitesse)

Le curseur vitesse permet de contrôler la vitesse de la simulation, c'est-à-dire la vitesse à laquelle les tortues se déplacent, les patches changent de couleur, et ainsi de suite.


Quand vous déplacez le curseur vers la gauche, la simulation ralentit car les pauses entre chaque pas de simulation sont plus longues. Ce qui permet de mieux voir ce qui se passe. Vous pouvez même ralentir la simulation à tel point qu'il est possible d'observer précisément ce que fait une tortue particulière.

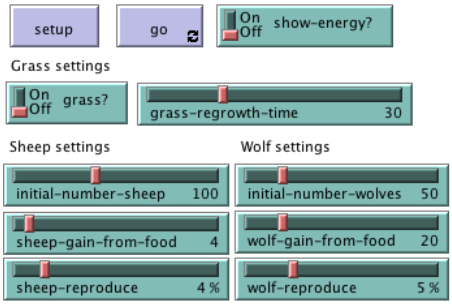
Quand vous déplacez le curseur vers la droite au-delà du milieu, ("normal speed") la simulation accélère. NetLogo commence de sauter des images, c'est-à-dire qu'il ne met plus l'affichage à jour à la fin de chaque pas de la simulation, mais seulement après quelques pas. Afficher l'état du monde prend du temps, donc l'afficher moins souvent signifie que la simulation peut progresser plus rapidement.

Notez que si vous déplacez le curseur assez loin vers la droite, le rafraîchissement de la vue peut se faire à une fréquence telle que le modèle semble ralentir. Ce n'est pourtant pas le cas, comme vous pouvez le constater en observant le compteur de pas. Seule la fréquence des rafraîchissements a diminué.

## Ajuster les réglages: curseurs et commutateurs

Les réglages d'un modèle vous donnent la possibilité de tester différents scénarios ou hypothèses. Modifier les réglages et relancer la simulation pour voir comment le modèle réagit à ces changements peut vous amener à une meilleure compréhension du phénomène simulé. Les commutateurs et les curseurs permettent de régler les valeurs des paramètres du modèle.

Voici les commutateurs et les curseurs du modèle Prédation Loups-Moutons :




Expérimentons leurs effets sur le comportement du modèle.

- Ouvrez le modèle " Wolf Sheep Predation " s'il ne l'est pas déjà.
- Pressez " setup " et " go " et laissez tourner la simulation pendant environ 100 pas.
  (Note : le nombre de pas écoulés est affiché en haut dans la barre de titre de la Vue " ticks: xx ".)
- Stoppez la simulation en pressant le bouton " go ".

➔ ***Qu'arrive-t-il aux moutons au cours du temps?***

Observez ce qui se passe avec les moutons si vous modifiez l'un des paramètres.

- Mettez le commutateur " grass? " sur " On ".
- Pressez " setup " et " go " et laissez tourner la simulation pendant le même temps qu'avant.

➔ ***Quel est l'effet de ce commutateur sur la simulation?***

➔ ***Le résultat est-il le même que lors de l'essai précédent?***

Tout comme les boutons, les **commutateurs** sont associés à des informations. Dans leur cas, cette information est de type " on/off ". Les commutateurs activent/désactivent des ensembles de directives différents. Ces directives ne sont généralement pas nécessaires pour le déroulement de la simulation, mais elles peuvent apporter une autre dimension au modèle. Activer le commutateur " grass? " affecte le résultat de la simulation. Dans la première simulation (commutateur sur " Off "), la croissance de l'herbe était constante. Ce qui ne correspond pas à une représentation réaliste de la relation prédateur-proie. Donc, en initialisant et en activant un taux de croissance de l'herbe, nous sommes capables de modéliser et de simuler les trois facteurs que sont les populations de moutons, de loups et d'herbe.

Les **curseurs** permettent un autre type de réglage des paramètres que les commutateurs. Un commutateur ne peut avoir que deux valeurs : enclenché ou déclenché. Un curseur permet le réglage d'un paramètre avec toute une série de valeurs numériques. Par exemple, le curseur " initial-number-sheep " (nombre initial de moutons) a pour valeur minimale 0 et pour valeur maximale 250. La simulation peut démarrer avec 0 mouton ou avec 250 moutons ou avec n'importe quel nombre de moutons entre ces deux valeurs. Essayez et voyez ce qui se passe. Quand vous déplacez la poignée de la valeur minimale à la valeur maximale, le nombre affiché à la droite du curseur change : il reflète la valeur spécifiée par la poignée du curseur.

Examinez les curseurs de la Prédation Loups-Moutons.

- Ouvrez puis lisez le contenu du panneau " Information " (dont l'onglet est placé au-dessus de la barre des outils) pour savoir à quel paramètre correspond chaque curseur du modèle.

Le panneau " Information " est un guide qui donne une information globale concernant le modèle. Dans ce panneau, vous trouverez une description et une explication du modèle, des suggestions concernant ce qu'il convient de tester, et d'autres informations encore. Vous pouvez soit lire le contenu du panneau " Information " avant de lancer la simulation, soit commencer tout de suite l'expérimentation puis jeter ensuite un oeil sur ce panneau.

➔ ***Qu'arriverait-il à la population de moutons s'il y avait plus de moutons et moins de loups au début de la simulation?***

- Placez le commutateur " grass? " sur " Off ".
- Placez le curseur " initial-number-sheep " sur 100.
- Placez le curseur " initial-number-wolves " sur 20.
- Pressez " setup " puis " go ".
- Laissez tourner la simulation pendant environ 100 cycles.
- Faites tourner la simulation plusieurs fois avec ces réglages.

➔ ***Qu'arrive-t-il à la population de moutons?***

➔ ***Le résultat vous surprend-il?***

➔ ***Quels autres curseurs ou commutateurs pourraient être ajustés pour aider la population de moutons?***

- Donnez une valeur de 80 au curseur " initial-number-sheep " et une valeur de 50 au curseur "i nitial-number-wolves ". (Ces valeurs sont proches de celles spécifiées par défaut lors de la première ouverture du modèle.)
- Mettez le curseur " sheep-reproduce " à 10.0%.
- Pressez " setup " puis " go ".
- Laissez tourner la simulation pendant environ 100 cycles.

➔ ***Qu'arrive-t-il aux loups au cours de cette simulation?***

Quand vous ouvrez un modèle, tous les curseurs et commutateurs sont réglés à leur valeur par défaut. Si

vous ouvrez un nouveau modèle ou quittez le programme, les paramètres que vous avez modifiés ne sont pas sauvegardés, à moins que vous ne choisissiez de le faire.

**Note** : en plus des curseurs et des commutateurs, certains modèles possèdent un troisième type de réglage, appelé sélecteur. Toutefois, le modèle Prédation Loups-Moutons n'en possède pas.)

## Recueillir des informations : Traceurs et Moniteurs

L'un des buts de la simulation à l'aide de modèles est la récolte d'informations (de données) concernant un sujet ou un thème qu'il serait très difficile d'obtenir par des expériences réelles. NetLogo peut afficher les données obtenues de deux manières : à l'aide de traceurs ou de moniteurs.

### Les traceurs

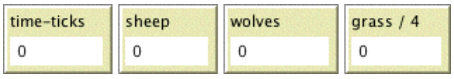
Le traceur de courbes de la Prédation Loups-Moutons contient trois courbes représentant les populations de moutons, de loups et d'herbe. (Le nombre représentant la quantité d'herbe est divisé par quatre pour qu'il ne soit pas trop grand et n'« écrase » pas les autres courbes dans l'affichage). Les courbes montrent ce qui se passe dans la simulation au cours du temps. (Pour savoir ce que représente chaque courbe, cliquez sur "Pens" dans le coin supérieur de la fenêtre du graphe pour afficher la légende des courbes. Une information s'affiche pour indiquer ce que chaque courbe représente. Dans notre cas, il s'agit du nombre d'individus de chaque population. — Obsolète avec les dernières versions de NetLogo car la légende des courbes est affichée par défaut.)

Quand la fenêtre du traceur est presque pleine, l'échelle horizontale (les abscisses) est modifiée (elle devient plus petite) et les données récoltées avant l'instant présent sont comprimées pour occuper moins d'espace, laissant ainsi de la place pour permettre aux courbes de représenter les données à venir.

Si vous voulez sauvegarder les données d'un graphique pour les visualiser ou les analyser dans un autre programme, vous pouvez utiliser le commande "Export Plot" du menu "File". Cette commande enregistre ces données sur votre mémoire de masse dans un fichier dont le format peut être lu par un tableur ou une base de données. Vous pouvez aussi exporter un graphe avec Ctrl-clic (Mac) ou clic-droit (Windows) puis sélectionner "Export...") dans le menu contextuel qui s'affiche.

### Les monitors

Les moniteurs sont un autre moyen d'afficher les informations d'une simulation. Voici les moniteurs utilisés dans la Prédation Loups-Moutons :




Le moniteur appelé " time-ticks " indique le temps écoulé (en pas ou cycles de simulation) depuis le lancement de la simulation. Ce moniteur n'est plus utilisés dans les versions récentes de cette simulation puisque cette information est fournie par défaut dans la barre de titre de la Vue.

Les autres moniteurs affichent la population de moutons ("sheeps"), celle des loups ("wolves") et la quantité d'herbe ("grass / 4"). (Rappelez-vous que la quantité d'herbe est divisée par quatre pour que le graphique ne devienne pas trop grand.)

Les valeurs affichées dans les moniteurs sont continuellement mises à jour pour refléter l'état actuel du système tant que la simulation tourne, alors que les graphes affichent les données pour toute la durée de la simulation.

Notez que NetLogo possède encore un autre type de moniteur appelé " agent monitors ". Il sera présenté dans le Tutoriel 2.

# Contrôler la Vue

Si vous examinez le panneau " Interface ", vous verrez toute une série de contrôles placés le long du bord supérieur de la barre d'outils. Ces contrôles permettent de modifier différents aspects de la Vue.

Testez les effets de ces contrôles :

- Pressez " setup " puis " go " pour démarrer une simulation.
- Pendant que la simulation tourne, déplacez le curseur vitesse (speed) vers la gauche.

➔ ***Que se passe-t-il?***

Ce curseur est utile si la simulation tourne trop vite pour vous permettre de voir ce qui se passe en détail.

- Déplacez le curseur vitesse vers le milieu (sur " normal speed ").
- Déplacez la poignée du curseur vers la droite.
- Essayez maintenant d'activer ou de désactiver le bouton de rafraîchissement de la vue " view updates ".

➔ ***Que se passe-t-il?***

Augmenter la vitesse de la simulation et désactiver le rafraîchissement de la vue est utile si vous êtes impatients et voulez que la simulation tourne plus vite. Augmenter la vitesse (déplacer la poignée du curseur vitesse vers la droite) fait sauter certaines mises à jour de la Vue de manière à ce que la simulation puisse tourner plus rapidement car le rafraîchissement de la vue prend du temps.

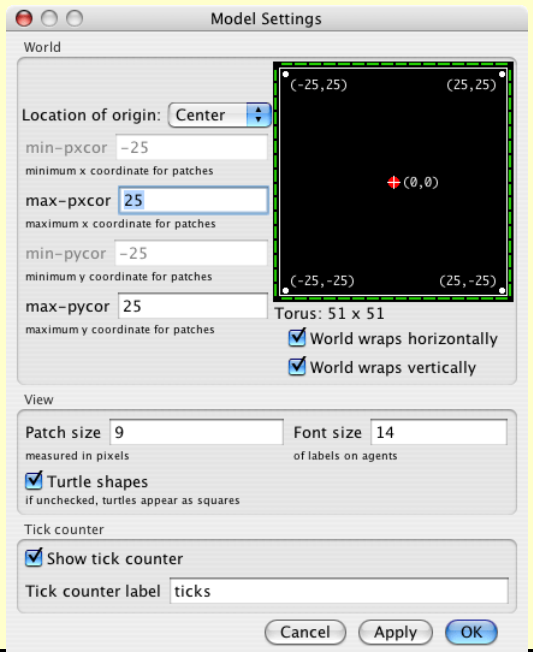
Quand le rafraîchissement de la vue est complètement désactivé, la simulation continue de tourner en arrière-plan, mais les traceurs et moniteurs sont toujours mis à jours. Si vous voulez voir ce qui se passe, vous devez réactiver le rafraîchissement de la vue en réactivant le bouton " view updates " prévu à cet effet. La plupart des modèles tournent beaucoup plus vite quand le rafraîchissement de la vue est désactivé.

La taille de la vue est déterminée par cinq réglages : Min et Max X, Min et Max Y et Patch Size. Observons ce qui se passe quand nous modifions les dimensions de la vue dans le modèle Prédation Loups-Moutons.

Il y a bien plus de réglages pour la Vue et le Monde qu'il n'y a de place pour eux dans la barre d'outils. Le bouton "Settings..." donne accès à tous ces autres réglages.

- Pressez le bouton " Settings... " de la barre d'outils.

La boîte de dialogue " Model settings " qui s'ouvre contient tous les réglages pour la Vue :



➔ ***Quels sont les valeurs actuelles pour "max-pxcor", "min-pxcor", "max-pycor", "min-pycor" et "Patch size" ?***

- Pressez le bouton " Cancel " pour refermer cette fenêtre sans modifier les réglages.
- Placez le pointeur de la souris près de, mais à l'extérieur, de la Vue.

Vous noterez que le pointeur se transforme en croix.

- Maintenez le bouton de la souris pressé et tirez la croix au-dessus de la vue.

La vue est maintenant sélectionnée, ce que NetLogo vous fait savoir en l'entourant d'une large bordure grise.

- Tirez l'une des «poignées» carrées noires. Ces poignées sont placées sur les côtés et aux coins de la vue.
- Dé-sélectionnez la vue en cliquant quelque part sur le fond blanc de panneau " Interface ".
- Pressez encore une fois le bouton " Settings... " et observez les nouvelles valeurs des réglages.

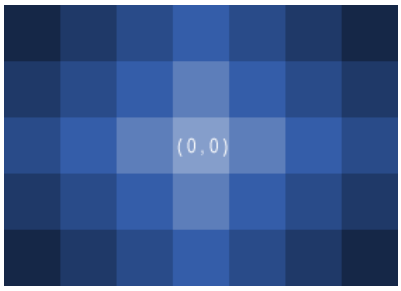
➔ ***Quelles sont les valeurs qui ont été modifiées?***

➔ ***Quelles sont les valeurs qui n'ont pas été modifiées?***

Le monde NetLogo est une grille de **patches** (plaques, carreaux, cases) à deux dimensions. Les patches sont les carrés de la grille.

Quand le commutateur " grass? " est activé dans le monde de la Prédation Loups-Moutons, les patches sont bien visibles car certains sont verts alors que les autres sont bruns.

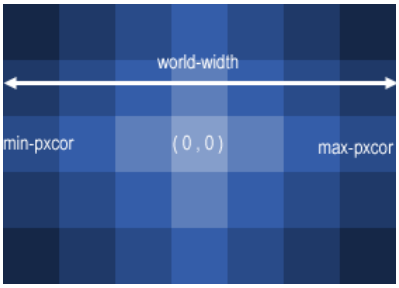
Considérez les patches comme étant des cases carrées formant le sol d'une chambre. Par défaut, la case placée au milieu de la pièce est désignée par (0,0), ce qui signifie que si la pièce est divisée en deux dans un sens puis aussi en deux dans l'autre sens, les deux lignes marquant les divisions se croisent au milieu, sur cette case. Nous avons maintenant un système de coordonnées qui pourra nous aider à localiser les objets dans la pièce :

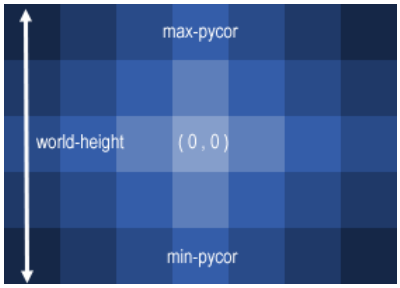



➔ ***De combien de patches le patch (0,0) est-il éloigné du bord droit de la pièce?***

➔ ***De combien de patches le patch (0,0) est-il éloigné du bord gauche de la pièce?***

En NetLogo, le nombre de patches du bord gauche au bord droit est la largeur du monde. Et le nombre de patches du bord supérieur au bord inférieur est la hauteur du monde. Ces nombres sont définis par les frontières supérieure, inférieure, gauche et droite.




Dans ces diagrammes, "`max-pxcor`" vaut 3 , "`min-pxcor`" -3, "`max-pycor`" 2 et "`min-pycor`" -2.

Quand vous modifiez la taille des patches, le nombre de patches (de carreaux) ne change pas, les patches sont dessinés soit plus grands, soit plus petits sur l'écran.

Observons maintenant les effets d'une modification des coordonnées minima et maxima du monde.

- Utilisez la boîte de dialogue " Settings " qui est encore toujours ouverte, donnez la valeur 30 à " max-pxcor " et 10 à " max-pycor ". Notez que les paramètres " min-pxcor " et " min-pycor " changent aussi. Ceci parce que, par défaut, l'origine (0,0) est au centre du monde.

➔ ***Qu'arrive-t-il à la forme de la Vue?***

- Pressez le bouton " setup ".

Vous pouvez voir maintenant que de nouveaux patches ont été créés.

- Éditez la vue en pressant encore une fois le bouton " Settings... ".
- Changez la taille des patches à 20 et pressez " OK ".

➔ ***Qu'arrive-t-il à la taille de la Vue? Est-ce que sa forme change?***

L'édition de la Vue permet aussi de modifier d'autres réglages, tels que la taille des caractères des étiquettes ou l'utilisation des formes. N'hésitez pas à essayer tous les réglages disponibles.

Une fois que vous aurez fini d'explorer le modèle Prédation Loup-Mouton, vous pourrez passez quelque temps à explorer quelques-uns des modèles de la Bibliothèque des modèles.

**Remarque** : quand vous quittez le modèle, cliquez le bouton Discard pour que les modifications que vous avez apportées au modèle ne soient pas enregistrées dans le modèle original. Si vous voulez conservez les modifications faites au modèle, sauvegardez-le sous un autre nom.

## La bibliothèque des modèles

Cette bibliothèque comporte cinq sections : Sample Models (Exemples de modèles), Perspective Demos (Démos de perspective), Curricular Models (Modèles de plan d'étude), Code Examples (Exemples de code) et HubNet Computer Activities (Activités informatiques HubNet).

### "Sample Models"

Cette section est organisée par sujets et contient actuellement plus de 210 modèles. Nous travaillons constamment pour y en ajouter d'autres, c'est pourquoi nous vous invitons à la visiter de temps en temps pour y découvrir les nouveautés.

Certains répertoires de cette section contiennent des sous-répertoires intitulés "(unverified)" (non vérifiés). Les modèles qui s'y trouvent sont complets et fonctionnent bien, mais ils sont encore en cours de vérifications quand à leur contenu, leur précision et la qualité de leur code.

### "Perspective Demos"

Ces modèles sont tous des modèles qui se trouvent dans la section "Sample Models" mais qui ont été légèrement modifiés pour démontrer les fonctionnalités d'affichage en perspective de NetLogo.

### "Curricular Models"

Ces modèles ont été développés pour être utilisés dans les écoles dans le contexte du plan d'études développé par le *CCl* de la *Northwestern University*. Certains de ces modèles se trouvent aussi dans la section "Sample Models", d'autres sont spécifiques à cette section. Consultez aussi le panneau "Information" de ces modèles pour avoir plus d'informations concernant le plan d'étude auquel ils sont rattachés.

### "Code Examples"

Ce sont de simples démonstrations de certaines fonctionnalités particulières de NetLogo. Elles vous seront utiles par la suite quand vous compléterez des modèles existants ou quand vous construirez vos propres modèles. Par exemple, si vous voulez afficher un histogramme dans l'un de vos modèles, étudiez le modèle "Histogram Example" pour voir comment vous y prendre.

### "HubNet Computer Activities"

Cette section contient des simulations à utiliser en réseau dans le cadre d'une classe. Pour en savoir plus au sujet de HubNet, voir le Guide HubNet.

## Et ensuite?

Si vous voulez apprendre comment explorer les modèles plus à fond, le Tutoriel 2 : Les commandes vous donnera une introduction au langage de modélisation de NetLogo.

Dans le Tutoriel 3 : Les procédures vous apprendrez comment modifier et compléter les modèles existants ou même comment créer vos propres modèles.

# Chapitre 9

# Tutoriel 2 : Les commandes

Dans le Tutoriel 1, vous avez eu l'occasion de voir quelques modèles NetLogo et avez achevé avec succès votre parcours à travers l'ouverture et la simulation des modèles en pressant des boutons, modifiant des valeurs à l'aide de curseurs et de commutateurs et en récoltant des données fournies par ces modèles à l'aide de graphes et de moniteurs. Dans cette section, l'accent sera mis au départ sur le passage de l'observation d'un modèle à la manipulation de ce modèle. Vous commencez à étudier le fonctionnement interne des modèles et serez alors capable de changer leurs apparences.

## Exemple de modèle : Traffic Basic

- Sélectionnez " Models Library " dans le menu " File ".
- Ouvrez " Traffic Basic " dans la section " Social Science " du dossier " Sample Models ".
- Testez le modèle pendant quelques minutes pour voir comment il fonctionne.
- Consultez le panneau " Information " pour toutes les questions concernant ce modèle.

Dans ce modèle, vous voyez une voiture rouge dans un flot de voitures bleues. Toutes les voitures vont dans la même direction. À intervalles assez réguliers, les voitures « s'entassent » et s'arrêtent. C'est une modélisation montrant comment des bouchons peuvent se former sans cause apparente telle qu'un accident, la rupture d'un pont ou le tête-à-queue d'un camion. Aucune « cause centralisée » n'est nécessaire pour qu'un bouchon se forme.

Modifiez certains réglages et observez quelques simulations pour bien comprendre le fonctionnement du modèle.

➔ ***Au cours des simulations de Traffic Basic, avez-vous pensé à des compléments que vous aimeriez apporter à ce modèle ?***

En observant le modèle Traffic Basic, vous avez pu constater que l'environnement est très simple : un arrière-plan noir avec une route claire sur laquelle circulent des voitures bleues et une voiture rouge. Les modifications pouvant être apportées au modèle comprennent : la modification de la couleur et de la forme des voitures, l'adjonction d'une maison ou d'un éclairage public, l'installation d'un feu rouge ou même la création d'une autre voie de circulation. Certaines de ces suggestions de modifications sont plutôt cosmétiques et ne feront qu'améliorer l'apparence du modèle alors que d'autres concernent avant tout le comportement du modèle. Au cours de la plus grande partie de ce tutoriel, nous nous concentrerons surtout sur les modifications « cosmétiques » ou les plus simples. (Le Tutoriel 3 entre bien plus en détail dans la modification des comportements, ce qui implique des modifications du code à faire dans le panneau " Procedures ".)

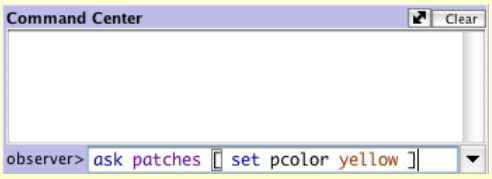
Nous utiliserons le Centre de commande (Command Center) pour apporter ces simples modifications.

# Le Centre de commande (Command Center)

Le Centre de commande (Command Center), qui se trouve dans le panneau " Interface ", permet d'entrer des commandes ou de donner des directives (des ordres) au modèle. Les commandes sont des instructions que vous pouvez donner aux agents de NetLogo : les tortues, les patches, les liens et l'observateur. (Référez-vous au Guide de l'interface pour une explication détaillée des différentes parties du Centre de commande.)

Dans le modèle Traffic Basic :

- Pressez le bouton " setup ".
- Localisez le Centre de commande " Command Center ".
- Cliquez avec la souris dans la boîte " observer> " située au bas du Centre de commande, c'est la Ligne de commandes.
- Entrez le texte montré ci-dessous :




- Pressez la touche Retour.

➔ ***Que s'est-il passé dans la Vue ?***

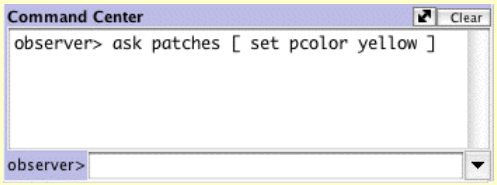
Vous avez certainement remarqué que l'arrière-plan noir de la vue est devenu jaune et que la rue a disparu.

➔ ***Pourquoi les voitures ne sont-elles pas, elles aussi, devenues jaunes ?***

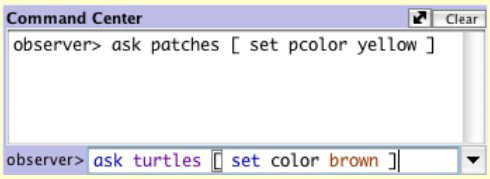
Revenons à la commande qui vous aviez donnée. Ce n'est qu'aux patches que nous avons demandé de changer de couleur. Dans ce modèle, les voitures sont représentées par un autre type d'agents, appelés « tortues » (turtles). C'est pourquoi les voitures n'ont pas reçu ces instructions et n'ont donc pas changé de couleur.

➔ ***Que s'est-il passé dans le centre de commande ?***

Vous devriez avoir remarqué que la commande que vous venez de taper est maintenant affichée dans la grande zone blanche au dessus de la Ligne de commandes, comme le montre l'illustration ci-dessous :




- Écrivez dans la Ligne de commandes au bas du Centre de commande le texte montré ci-dessous :



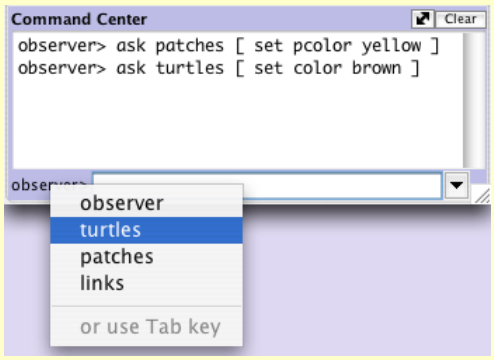

➔ ***Le résultat correspond-il à ce que vous attendiez ?***

La Vue devrait avoir un arrière-plan jaune avec une file de voitures brunes au milieu.

Le monde de NetLogo est un monde à deux dimensions qui comprend des tortues, des patches et un observateur. Les patches forment le sol sur lequel les tortues peuvent se déplacer et l'observateur est un être qui observe d'en haut tout ce qui se passe dans le monde. (Pour une description détaillée de ce monde et de ses spécificités, référez-vous au Guide de programmation de NetLogo.)

Dans le Centre de commande, nous avons la possibilité de donner une commande à l'observateur (observer), une commande aux tortues (turtles) ou une commande aux patches. Nous pouvons choisir l'une de ces options en utilisant le menu situé dans le coin inférieur gauche du Centre de commande. Vous pouvez aussi utiliser la touche Tab du clavier pour passer d'une option à l'autre.

- Dans le Centre de commande, cliquez sur " observer> " situé dans le coin inférieur gauche :




- Sélectionnez " turtles " dans le menu qui s'affiche.
- Écrivez `set color pink` et pressez la touche Retour.
- Pressez la touche Tab jusqu'à ce que vous voyez " patches " dans le coin inférieur gauche.
- Écrivez `set pcolor white` et pressez la touche Retour.

➔ ***Quel est maintenant l'aspect de la Vue ?***

➔ ***Avez-vous remarqué des différences entre ces deux commandes et les commandes passées auparavant à l'observateur ?***

L'observateur regarde le monde d'en haut et peut donc donner une commande aux patches ou aux tortues en utilisant `ask`. Comme dans le premier exemple (observer> `ask patches [set pcolor yellow]`), l'observateur doit demander aux patches de donner à leur paramètre couleur `pcolor` la valeur `yellow`. Mais quand une commande est donnée directement à un groupe d'agents comme dans le second exemple (patches> `set pcolor white`), vous ne devez que donner la commande elle-même (puisque NetLogo sait à qui vous vous adressez).

- Pressez " setup ".

➔ ***Que s'est-il passé ?***

Pourquoi la Vue est-elle repris son aspect d'origine, avec un arrière-plan noir et une route claire? En pressant le bouton "setup", le modèle se re-configure lui-même en utilisant les réglages spécifiés par le code du modèle dans le panneau "Procedures". La Centre de commande n'est que très rarement utilisé pour modifier un modèle de manière permanente. Il est par contre plus souvent utilisé pour personnaliser le modèle courant et permettre de manipuler le monde NetLogo pour répondre à de questions du genre « Et si » qui pourraient surgir lorsque vous explorez les modèles. (Le panneau "Procedures" est expliqué dans le prochain tutoriel et dans le Guide de la programmation.)

Maintenant que vous vous êtes familiarisés avec le Centre de commande, voyons quelques particularités concernant le fonctionnement des couleurs dans NetLogo.

# Travailler avec les couleurs

Vous aurez certainement remarqué, dans la section précédente, que nous avons utilisé deux termes différents pour changer les couleurs : `color` et `pcolor`.

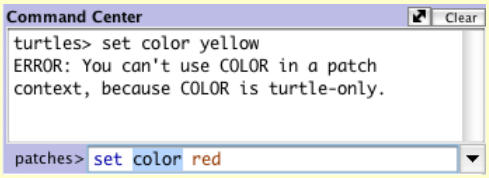
➔ ***Quelle différence y a-t-il entre color et pcolor ?***

- Sélectionnez " turtles " dans le menu du Centre de commande (ou utilisez la touche Tab pour y parvenir).
- Écrivez `set color blue` et pressez la touche Retour.

➔ ***Que se passe-t-il avec les voitures ?***

- Réfléchissez à ce que vous avez fait pour peindre les voitures en bleu et essayez de peindre les patches en rouge.

Si vous essayez de dire aux patches `set color red`, NetLogo affiche un message d'erreur disant : `Vous ne pouvez pas utiliser COLOR dans un contexte patch, car COLOR est réservé aux tortues.`



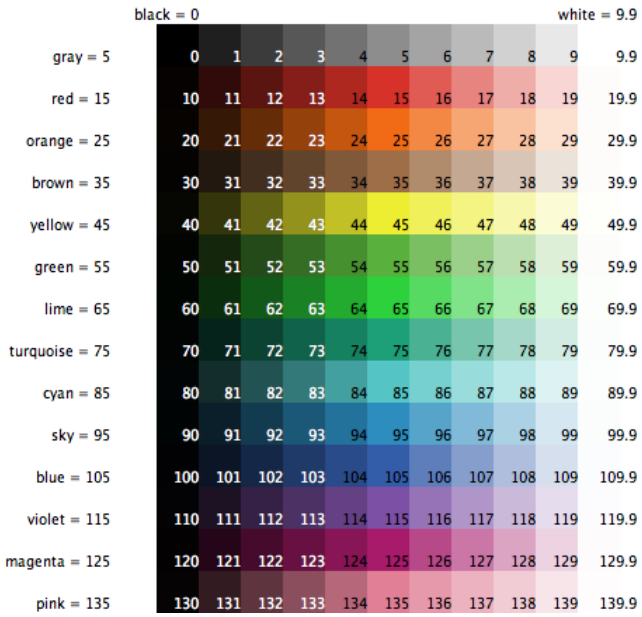

- Écrivez plutôt `set pcolor red` et pressez la touche Retour.

Nous disons que `color` et `pcolor` sont des **variables**. Certaines commandes et certaines variables sont spécifiques aux tortues alors que d'autres sont spécifiques aux patches. Par exemple, la variable `color` est une variable réservée aux tortues alors que la variable `pcolor` est réservée aux patches.

Allez de l'avant et exercez-vous en modifiant les couleurs des tortues et des patches au moyen de la commande `set` et de ces deux variables.

Afin d'avoir un plus grand choix de couleurs pour peindre les tortues et les patches (ou devrions-nous dire les voitures et l'arrière-plan), voyons un peu plus en détail comment les couleurs sont gérées par NetLogo.

Pour NetLogo, toutes les couleurs sont désignées par une valeur numérique. Si, dans tous les exercices précédents, nous avons utilisé les noms des couleurs, c'est d'une part parce que c'est plus simple et d'autre part parce que NetLogo connaît 16 couleurs par leur nom. Mais cela ne veux pas dire que NetLogo ne reconnaisse que 16 couleurs. De nombreuses nuances, situées entres ces 16 couleurs de base, peuvent aussi être utilisées. Voici une palette montrant tout l'espace de couleurs utilisable par NetLogo :



Pour obtenir une couleur qui n'a pas de nom, il suffit de s'y référer en utilisant son numéro, ou en additionnant ou en soustrayant un nombre d'un nom de couleur. Par exemple, quand vous écrivez `set color red`, cette commande fait la même chose que si vous aviez écrit `set color 15`. Et vous pouvez obtenir une version plus claire ou plus sombre de cette même couleur en utilisant un nombre qui est un peu plus petit ou un peu plus grand, comme le montrent les exemples suivants :

- Sélectionnez " patches " dans le menu du Centre de commande (ou utilisez la touche Tab).
- Écrivez `set pcolor red - 2` ( Les espaces entourant le signe " - " sont importants. )

En soustrayant un nombre de `red`, vous obtenez un rouge plus sombre.

- Écrivez `set pcolor red + 2`

En ajoutant un nombre à `red`, vous obtenez un rouge plus clair.

Vous pouvez utiliser cette technique pour n'importe quelle couleur listée dans la palette.

## Moniteurs d'agent et Agent Commanders

Dans l'activité précédente, nous avions utilisé la commande " `set` " pour modifier la couleur de toutes les voitures. Mais si vous vous en rappelez, il y avait dans le modèle original une voiture rouge au milieu des voitures bleues. Voyons comment changer la couleur d'une seule voiture.

- Pressez " setup " pour réinitialiser le modèle et faire réapparaître la voiture rouge.
- Si vous êtes sur Macintosh (souris à un bouton), maintenez la touche Ctrl enfoncée et cliquez la voiture rouge. Sur les autres systèmes (ou sur Mac avec souris à 2 boutons) cliquez la voiture rouge avec le bouton droit de la souris pour faire apparaître un menu contextuel.
- Si plusieurs tortues sont très proches de la tortue rouge, elles apparaîtront toutes dans le menu. Déplacez votre souris sur cette liste de tortues et remarquez que lorsque la souris met en évidence un nom de tortue dans le menu, la tortue correspondante est aussi mise en évidence dans la vue. Sélectionnez "inspect turtle" dans le sous-menu de la tortue `red`.

Une fenêtre « moniteur tortue » pour la voiture rouge apparaît :

En examinant ce moniteur tortue de plus près, nous pouvons voir toutes les variables qui appartiennent (qui définissent) la voiture rouge. Une variable est un emplacement (on pourrait dire un tiroir ou une boîte) qui contient une valeur qui peut être modifiée. Vous rappelez-vous qu'il a été dit que, dans les ordinateurs, toutes les couleurs sont représentées par des nombres? Il en va de même pour les agents. Par exemple, chaque tortue possède un numéro d'identification que nous appelons son "who number".

Examinons plus attentivement le moniteur tortue :

➔ ***Quel est le numéro "who number" de cette tortue ?***

➔ ***Quelle est la couleur de cette tortue ?***

➔ ***Quelle est la forme de cette tortue ?***

Ce moniteur tortue montre une tortue dont le numéro est 0, qui a une couleur 15 (rouge – voir la palette) et une forme " `car` " (voiture).

Il existe deux autres moyens d'ouvrir un moniteur tortue en plus du clic-droit (ou Ctrl-clic, en fonction du système d'opération). L'un de ces moyens consiste à sélectionner la commande " Turtle Monitor " dans le menu " Tools ", d'écrire le numéro de la tortue que vous voulez examiner dans le champ " who " puis de pressez la touche Retour. L'autre moyen est d'écrire `inspect turtle 0` (ou un autre numéro " who ") dans le Centre de commande.

Vous fermez un moniteur tortue en cliquant le bouton de fermeture situé dans le coin supérieur gauche (Macintosh) ou dans le coin supérieur droit (les autres systèmes d'exploitation) de la fenêtre du moniteur.

Maintenant que nous connaissons mieux les Moniteurs d'agent, nous avons trois possibilités pour modifier la couleur d'une seule tortue.

L'une de ces possibilités est d'utiliser la boîte de texte appelée " Agent Commander " (Commandant d'agent) située à la base de la fenêtre Moniteur d'agent. Là, vous y inscrivez vos commandes tout comme dans le Centre de commande, mais les commandes que vous y donnez ne sont valables, et donc exécutées, que par cette tortue particulière.

➢ Dans le Commandant d'agent du Moniteur de tortue pour la tortue 0, écrivez `set color pink`

➔ ***Que se passe-t-il dans la Vue ?***

➔ ***Y a-t-il eu des modifications dans le Moniteur de tortue ?***

Une autre manière de modifier la couleur d'une seule tortue est d'aller directement dans le champ "color" du Moniteur de tortue et d'y changer la valeur affichée.

➢ Sélectionnez le texte dans le champ situé à droite de "color" dans le Moniteur tortue.
➢ Écrivez le code d'une nouvelle couleur, par exemple `green + 2`

➔ ***Que se passe-t-il ?***

La troisième méthode pour changer la couleur d'une tortue ou d'un patch particulier consiste à utiliser l'observateur (observer). En effet, puisque l'observateur domine le monde NetLogo, il peut donner des commandes qui affectent aussi bien des tortues individuellement que des groupes de tortues.

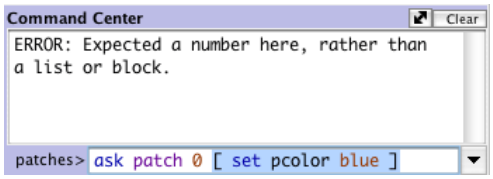
➢ Dans le Centre de commande, sélectionnez "observer" dans le menu (ou utilisez la touche Tab).
➢ Écrivez `ask turtle 0 [set color blue]` et pressez Retour.

➢ ***Que se passe-t-il ?***

Tout comme il y a des Moniteurs tortue, il existe aussi des Moniteurs patch. Les Moniteurs patch fonctionnent, à quelques détails près, comme les Moniteurs tortue.

➔ Pouvez-vous faire apparaître un Moniteur patch puis l'utiliser pour changer la couleur d'un seul patch ?

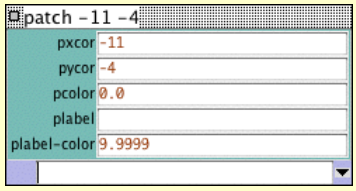
Si vous utilisez le Centre de commande avec la commande `ask patch 0 [set pcolor blue]`, vous obtiendrez le message d'erreur suivant : `ERREUR: Attend ici un nombre plutôt qu'une liste ou un bloc.`




Pour demander à une tortue particulière de faire quelque chose, nous utilisons son numéro personnel. Mais comme les patches n'ont pas de numéro personnel, nous devons nous adresser à eux d'une autre manière.

Rappelez-vous, les patches sont disposés dans un système de coordonnées. Deux nombres sont nécessaires pour pouvoir dessiner un point dans un graphique à deux dimensions. Les positions des patches sont déterminées de la même manière que les positions des points dans un graphique.

➢ Ouvrez le Moniteur patch d'un patch quelconque.




Le moniteur montre que, pour le patch de la figure ci-dessus, sa variable `pxcor` vaut -11 et sa variable `pycor` vaut -4. Si nous revenons à l'analogie avec les coordonnées du plan et que nous voulons dessiner ce point, il se trouvera dans le quadrant inférieur gauche où se situent les coordonnées x = -11 et y = -4.

Pour dire à ce patch particulier de changer de couleur, il faut utiliser ses coordonnées.

➢ Entrez `set pcolor blue` dans le champ texte situé au bas du Moniteur patch et pressez la touche Retour.

La commande donnée dans un Moniteur tortue ou patch ne concerne que cette tortue ou ce patch.

Vous pouvez aussi vous adresser à un seul patch via le Centre de commande :

➢ Dans le Centre de commande, écrivez `ask patch -11 -4 [set pcolor green]` et pressez Retour.

## Et ensuite ?

Arrivés à ce point, vous aurez très certainement envie de passer quelque temps à essayer ces techniques sur d'autres modèles de la Bibliothèque des modèles.

Dans le Tutoriel 3 : Les procédures vous pourrez apprendre comment modifier ou compléter des modèles existants ou même comment créer vos propres modèles.

# Chapitre 10

# Tutoriel 3 : Les procédures

Ce tutoriel vous conduit à travers le processus de réalisation d'un modèle complet, construit étape par étape, chaque étape étant expliquée pas à pas.

## Agents et procédures

Dans le Tutoriel 2, vous avez appris à utiliser le Centre de commande et les Moniteurs d'agent pour inspecter et modifier les agents et les obliger à faire certaines choses. Nous sommes maintenant prêts pour apprendre à utiliser le véritable coeur des modèles NetLogo : le panneau des Procédures "Procedures".

Vous avez déjà utilisé des types d'agents auxquels vous pouvez donner des commandes en NetLogo : les patches, les tortues, les liens et l'observateur. Les patches sont immobiles et disposés sur une grille. Les tortues se déplacent sur cette grille. Les liens établissent une connexion entre deux tortues. L'observateur surveille (et voit) tout ce qui se passe et fait tout ce que les tortues, les patches ou les liens ne peuvent pas faire eux-mêmes.

Tous les agents appartenant à ces quatre types peuvent exécuter des primitives NetLogo. Et tous les quatre peuvent aussi exécuter des procédures. Une **procédure**, que vous devez définir, réunit une série de commandes NetLogo pour n'en former plus qu'une seule.

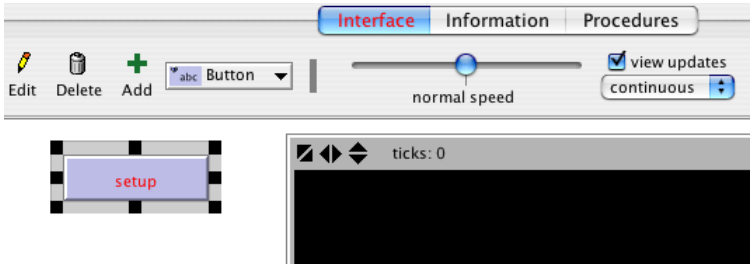
Vous allez maintenant apprendre à écrire des procédures qui font bouger les tortues, les font manger, se reproduire et mourir. Vous allez aussi apprendre comment créer des moniteurs, des curseurs et des traceurs de courbes. Le modèle que nous allons construire est un modèle d'un écosystème simple, un peu dans le genre du modèle Prédation Loups-Moutons du Tutoriel 1.

## Créer le bouton PREPARER (setup)

Pour créer un nouveau modèle, sélectionnez "New" dans le menu "File". Puis commencez par créer un bouton d'initialisation "PREPARER" :

- Cliquez l'icône " Button " dans la barre d'outils placée au sommet du panneau " Interface ".
- Cliquez dans la zone vide, blanche, du panneau "Interface", là où vous voulez mettre le bouton.
- Une boîte de dialogue d'édition de bouton s'ouvre. Écrivez `PREPARER` dans le champ texte appelé " Commands ".
- Pressez le bouton " OK " quand vous avez terminé. La boîte de dialogue se ferme.

Vous avez maintenant un bouton "PREPARER" (setup). Presser ce bouton devrait déclencher l'exécution d'une procédure PREPARER. Une procédure est une séquence de commandes NetLogo à laquelle un nouveau nom a été donné. Mais nous n'avons pas encore défini cette procédure (nous le ferons bientôt). Puisque le bouton fait référence à une procédure qui n'existe pas encore, son intitulé est rouge (tout comme le texte de l'onglet "Interface") :



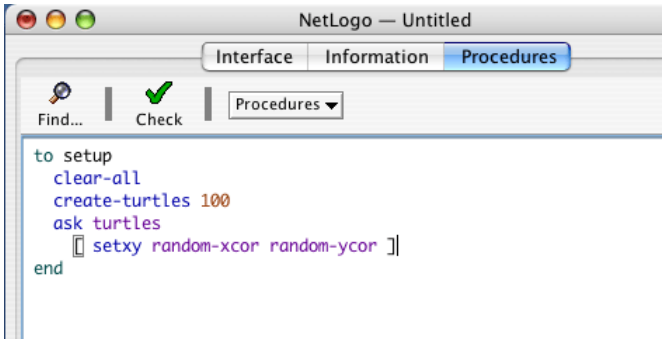

Si vous voulez voir le message d'erreur, cliquez le bouton.

Nous allons maintenant créer la procédure PREPARER, ce qui fera disparaître le message d'erreur :

- Allez dans le panneau " Procedures ".
- Écrivez les lignes de code suivantes :

```
to PREPARER
  clear-all
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
end
```

Quand vous avez terminé, le panneau "Procedures" ressemble à ceci :




Notez que certaines lignes sont indentées à différents degrés. La plupart des programmeurs trouvent utile d'indenter leur code de cette manière, mais ce n'est pas une obligation. Cela rend le code plus facile à lire (et donc à comprendre) et à modifier. La procédure commence par le mot `to` et se termine par le mot `end`. Toute nouvelle procédure que vous créez devra commencer et se terminer par ces deux mots.

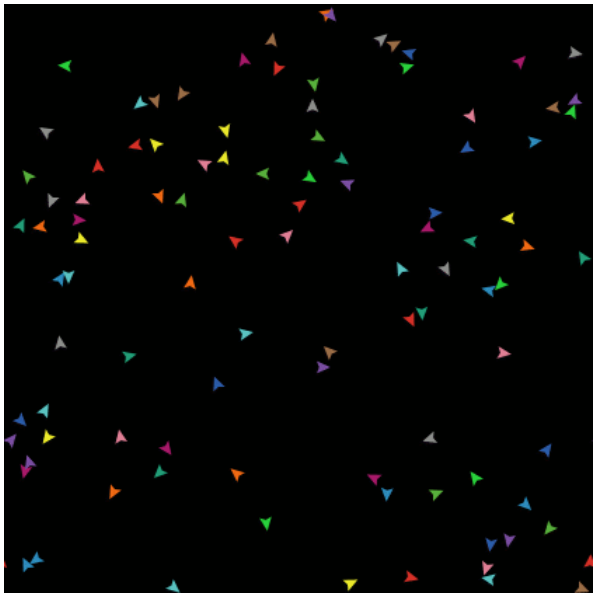
Examinons ce que vous avez écrit et voyons ce que fait chaque ligne de cette procédure :

- `to PREPARER` commence la définition d'une procédure appelée PREPARER.
- `clear-all` remet le monde dans son état initial, vide. Tous les patches deviennent noirs et toutes les tortues que vous auriez pu créer disparaissent. En fin de compte, cette commande nettoie tout (fait table rase) pour une nouvelle simulation du modèle.
- `create-turtles 100` crée 100 tortues. À la création, elles sont toutes placées à l'origine du monde, c'est-à-dire au centre du patch 0,0.

- ✔ `ask turtles [ ... ]` demande à chaque tortue d'exécuter indépendamment les commandes placées entre les crochets. (En NetLogo, chaque commande est exécutée par un agent. Et `ask` est aussi une commande. Ici, c'est l'observateur qui exécute la commande `ask` elle-même, qui demande à son tour aux tortues d'exécuter les commandes entre crochets.)
- ✔ `setxy random-xcor random-ycor` est une commande qui utilise des reporters. Un ***reporter***, à l'inverse d'une commande, rend compte de son action, autrement dit ***retourne*** un résultat. D'abord, chaque tortue exécute le reporter `random-xcor` qui retourne un nombre aléatoire appartenant à l'ensemble des coordonnées de tortue possibles le long de l'axe des X. Puis chaque tortue exécute le reporter `random-ycor`, qui fait la même chose pour l'axe des Y. Finalement, chaque tortue exécute la commande `setxy` en lui transmettant en entrée les deux nombres qu'elle vient de recevoir. Ce qui se traduit par un déplacement de la tortue sur le point désigné par ces coordonnées. En résumé, cette séquence de commandes disperse les 100 tortues de manière aléatoire sur tout le terrain disponible.
- ✔ `end` termine la définition de la procédure `PREPARER`.

Quand vous aurez fini d'écrire, revenez dans le panneau "Interface". Remarquez que les intitulés rouges sont redevenus noirs car le bouton "PREPARER" a maintenant une procédure `PREPARER` à laquelle s'adresser.

Pressez le bouton "PREPARER" et vous verrez toutes les tortues se disperser dans leur monde.



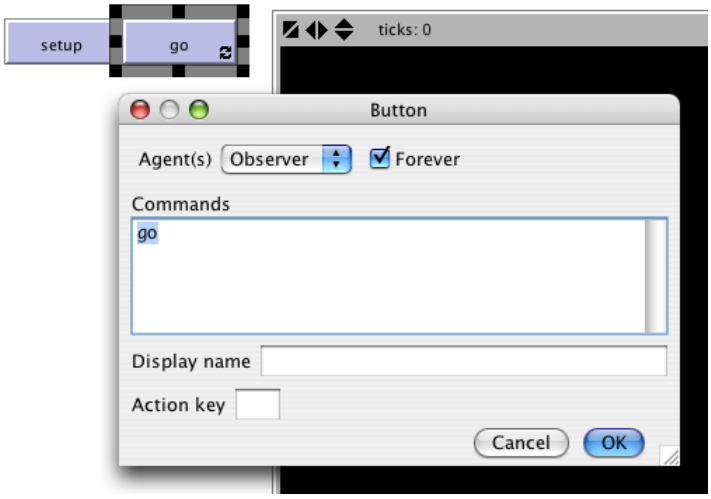
Pressez le bouton "PREPARER" encore quelques fois et observez comment les positions des tortues sont à chaque fois différentes. Notez que certaines tortues peuvent se retrouver « à cheval » sur d'autres tortues.

Réfléchissez un petit peu à ce que vous avez dû faire pour arriver à ce résultat. Vous avez dû placer un bouton dans l'interface et créer une procédure que ce bouton utilise (appelle). Le bouton ne fonctionne qu'une fois ces deux opérations distinctes achevées. Dans la suite de ce tutoriel, vous aurez souvent à accomplir deux ou plusieurs étapes semblables pour ajouter une nouvelle fonctionnalité au modèle. Si quelque chose semble ne pas fonctionner une fois terminé ce qui vous semble être la dernière étape de l'élaboration d'une nouvelle fonctionnalité, continuez de lire plus loin pour voir s'il n'y a pas encore quelque chose à faire. Une fois que vous aurez lu quelques paragraphes, reprenez en sens inverses les différentes directives pour voir si vous n'auriez pas oublié une étape.

## Créer le bouton GO

Créez maintenant un bouton appelé "GO". Suivez les mêmes étapes que celles accomplies pour créer le bouton "PREPARER", sauf :

- Dans " Commands " entrez `GO` au lieu de PREPARER.
- Cochez le bouton " Forever " dans la boîte de dialogue " Button ".




L'activation du bouton "Forever" force le bouton "GO" à rester enfoncé une fois qu'il a été pressé, ce qui fait que ses commandes seront exécutées en boucle (encore et encore) et pas seulement une fois.

- Ajoutez ensuite une procédure GO dans le panneau " Procedures " :

```
to GO
  bouge-tortues
end
```

Mais qu'est-ce que `bouge-tortues`? Est-ce une primitive (en d'autres mots, une commande prédéfinie de NetLogo), comme l'est `clear-all`? Non, il s'agit d'une autre procédure que vous devrez créer. Jusqu'ici, vous avez déjà ajouté deux procédures que vous avez créées : `PREPARER` et `GO`.

- Ajoutez la procédure `bouge-tortues` en dessous de la procédure `GO` :

```
to GO
  bouge-tortues
end

to bouge-tortues
  ask turtles [
    right random 360
    forward 1
  ]
end
```

Notez qu'il n'y a pas d'espaces autour du tiret dans `bouge-tortues`. Dans le Tutoriel 2 nous avions utilisé `red - 2` avec des espaces autour du signe moins dans le but de soustraire deux nombres, mais ici, nous devons écrire `bouge-tortues` sans espaces. Le tiret "-" réunit les mots `bouge` et `tortues` en un seul mot.

Voici ce que fait chaque commande de la procédure `bouge-tortues` :

- ✔ `ask turtles [ ... ]` demande à chaque tortue d'exécuter les commandes entre crochets.
- ✔ `right random 360` est une autre commande qui utilise un reporter. D'abord, chaque tortue prend au hasard un nombre entier entre 0 et 359 (`random` ne retourne pas le nombre qui lui a été donné). Puis la tortue tourne à droite du nombre de degrés correspondant au nombre retourné par `random`.
- ✔ `forward 1` fait avancer la tortue d'un pas.

Pourquoi ne pourrions-nous pas tout simplement écrire toutes ces commandes dans la procédure `GO` plutôt que dans une procédure séparée? Nous aurions pu le faire, mais au cours du développement du projet, il est presque sûr que vous y ajouterez bien d'autres parties. Nous aimerions garder `GO` aussi simple que possible de manière à ce que cette procédure soit facilement compréhensible. Par la suite, ce modèle comportera bien d'autres choses que vous voudrez voir se faire quand la simulation tournera, telles que le calcul de certaines valeurs ou l'affichage de résultats. Chacune de ces choses à faire aura sa propre procédure et chaque procédure devra avoir son propre nom, unique dans le programme.

Le bouton "GO" que vous avez créé dans le panneau interface est un bouton « pour toujours », ce qui signifie qu'il exécutera continuellement ses commandes jusqu'à ce que vous le déclenchiez (en le cliquant une seconde fois). Après avoir pressé le bouton "PREPARER" une fois pour créer les tortues, pressez le bouton "GO". Observez ce qui se passe. Désactivez-le, et vous verrez que toutes les tortues s'arrêtent instantanément sur leur chemin.

Notez que si une tortue se déplace au-delà du bord du monde, elle «s'enroule», ce qui signifie qu'elle réapparaît au côté opposé. (C'est le comportement des tortues par défaut qui peut être modifié. Consultez à ce sujet la section Topologie du Guide de la programmation pour en savoir plus.)

## Expérimenter avec les commandes

Nous vous suggérons d'essayer d'autres commandes de tortue.

- ✔ Écrivez des commandes (telles que `turtles> set color red`) dans le Centre de commande ou ajoutez des commandes aux procédures `PREPARER`, `GO`, ou `bouge-tortues`.

- ✔ Notez que quand vous entrez des commandes dans le Centre de commande, vous devez sélectionner "`turtles>`", "`patches>`", ou "`observer>`" dans le menu placé à gauche, en fonction du type d'agents qui devra exécuter ces commandes. Cette manière de faire revient au même que de taper `ask turtles` ou `ask patches`, mais épargne des frappes claviers. Vous pouvez aussi utiliser la touche Tab pour passer d'un type d'agents à l'autre, ce que vous trouverez certainement plus pratique que l'utilisation du menu.

- ✔ Essayer de taper `turtles> pen-down` dans le Centre de commande puis de pressez le bouton "GO".

- ✔ De même, dans la procédure `bouge-tortues` , essayez de remplacer `right random 360` par `right random 45`.

Amusez-vous. C'est facile et les résultats sont immédiatement visibles (caractéristique qui est l'un des nombreux points forts de NetLogo).

Quand vous aurez l'impression d'avoir fait (pour le moment) toutes les expérimentations que vous souhaitez, vous êtes prêts pour poursuivre l'amélioration du modèle que vous êtes en train de construire.

## Patches et variables

Nous avons maintenant 100 tortues qui errent sans but, ignorant absolument tout ce qui les entoure. Rendons les choses un peu plus intéressantes en donnant à ces tortues un joli arrière-plan sur lequel se déplacer.

➢ Revenez à la procédure PREPARER. Nous pouvons la modifier comme suit :

```
to PREPARER
  clear-all
  prepare-patches
  prepare-tortues
end
```

➢ La nouvelle définition de PREPARER fait appel à deux nouvelles procédures. Pour définir `prepare-patches`, ajoutez ceci :

```
to prepare-patches
  ask patches [ set pcolor green ]
end
```

La procédure `prepare-patches` donne d'abord la couleur verte à chaque patch. (La variable servant à spécifier la couleur d'une tortue est `color`, celle pour la couleur d'un patch est `pcolor`.)

La seule partie encore non définie de notre nouvelle procédure "PREPARER" est `prepare-tortues`.

➢ Ajoutez aussi cette procédure :

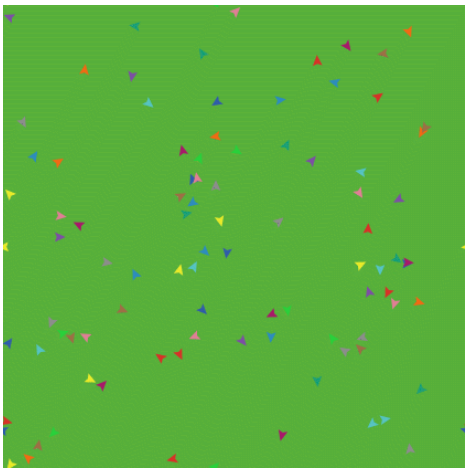
```
to prepare-turtles
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
end
```

➔ Avez-vous remarqué que la nouvelle procédure `prepare-tortues` a presque les même commandes que l'ancienne procédure PREPARER ?

➢ Revenez dans le panneau " Interface ".

➢ Pressez le bouton " PREPARER ".

Et voila! Un paysage NetLogo luxuriant avec des tortues colorées et des patches verts apparaît :



Après avoir observé quelques fois le fonctionnement de la nouvelle procédure PREPARER, il ne serait pas inutile de relire encore une fois les définitions de toutes les procédures.

## Les variables tortue

Nous avons maintenant quelques tortues qui courent encore et toujours sans buts dans toute la verte campagne, mais qui ne font rien de plus et ignorent complètement leur environnement. Ajoutons donc quelques interactions entre les tortues et les patches.

Nous allons faire en sorte que les tortues mangent de l'herbe (les patches verts), se reproduisent et meurent après un certain temps. L'herbe devra repousser petit à petit après avoir été mangée.

Nous avons besoin d'un moyen pour contrôler quand une tortue se reproduit et quand elle meurt. Nous allons déterminer ces moments en gardant la trace de la quantité « d'énergie » que possède chaque tortue. Cette valeur devra être stockée dans une variable tortue que nous appellerons `energie`.

Nous avons déjà rencontré des variables tortue prédéfinies telles que `color`. Toutefois, NetLogo ne connaît pas de variable tortue `energie` mais nous permet d'en créer une.

- Revenez dans le panneau " Procedures ".
- Définissez la nouvelle variable `energie` au moyen de la déclaration `turtle-own` à la première ligne du code.

```
turtles-own [energie]
```

Utilisons cette nouvelle variable (`energie`) pour permettre aux tortue de manger.

- Récrivez la procédure GO de la manière suivante :

```
to GO
  bouge-tortues
  mange-herbe
end
```

- Ajoutez une nouvelle procédure `mange-herbe` :

```
to mange-herbe
  ask turtles [
  if pcolor = green [
    set pcolor black
    set energie (energie + 10)
    ]
  ]
end
```

Nous utilisons la commande `if` pour la première fois. Examinez attentivement le code. Chaque tortue, quand elle exécute cette commande, compare la valeur de la couleur du patch (`pcolor`) sur lequel elle se trouve à la valeur de `green`, valeur numérique spécifiant la couleur verte de base dans NetLogo. (Une tortue a un accès direct aux variables du patch sur lequel elle se trouve.) Si la couleur du patch est `green`, la comparaison retourne `true`, et dans ce cas seulement la tortue exécutera les commandes qui sont entre les crochets (sinon, elle les saute). Ces commandes font que la tortue change la couleur du patch en noir et augmente sa réserve d'énergie de 10 unités. La patch devient noir pour montrer que l'herbe qu'il portait a été mangée et que la tortue a gagné de l'énergie car elle vient de manger l'herbe.

Puis faisons en sorte que les mouvements des tortues consomment un peu de cette énergie.

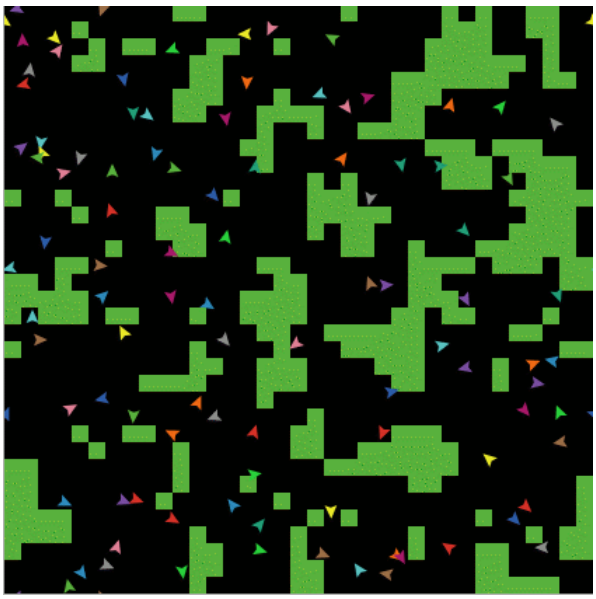
➢ Réécrivez `bouge-tortues` de la manière suivante :

```
to bouge-tortues
  ask turtles [
    right random 360
    forward 1
    set energie energie – 1
  ]
end
```

Maintenant, quand une tortue se déplace, elle perd une unité d'énergie à chaque pas.

➢ Passez dans le panneau " Interface " et pressez les boutons " PREPARER " puis " GO ".

Vous verrez les patches devenir noirs quand les tortues leur passent dessus.



## Les moniteurs

Nous allons ensuite placer deux moniteurs dans le panneau "Interface" à l'aide de la barre d'outils. (Ils se construisent exactement de la même manière que les boutons et les curseurs, en utilisant le menu déroulant de la barre d'outils). Créons le premier moniteur.

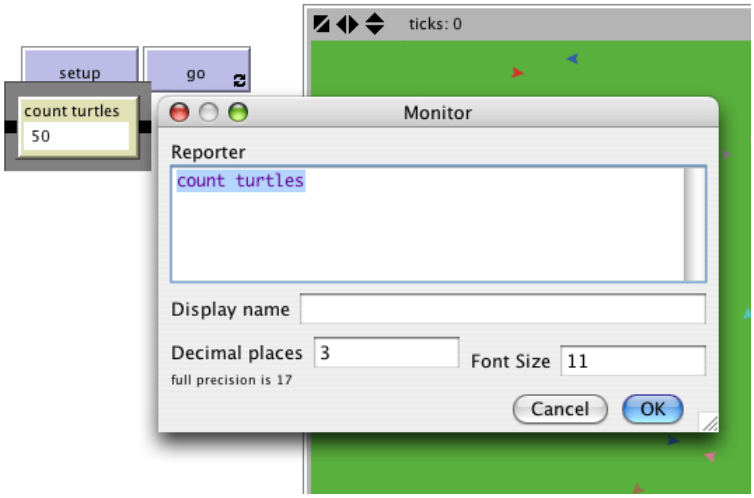
➢ Créez un moniteur en ouvrant le menu déroulant de la barre d'outils et en y sélectionnant la commande " Monitor ", puis cliquez dans une zone non occupée de l'Interface pour le placer.

Une boîte de dialogue apparaît.

➢ Écrivez `count turtles` dans la zone " Reporter " de cette boîte de dialogue (voir l'image ci-dessous).

➢ Écrivez `Nombre d'animaux` dans le champ " Display name "

➢ Pressez le bouton " OK " pour fermer la boîte de dialogue.



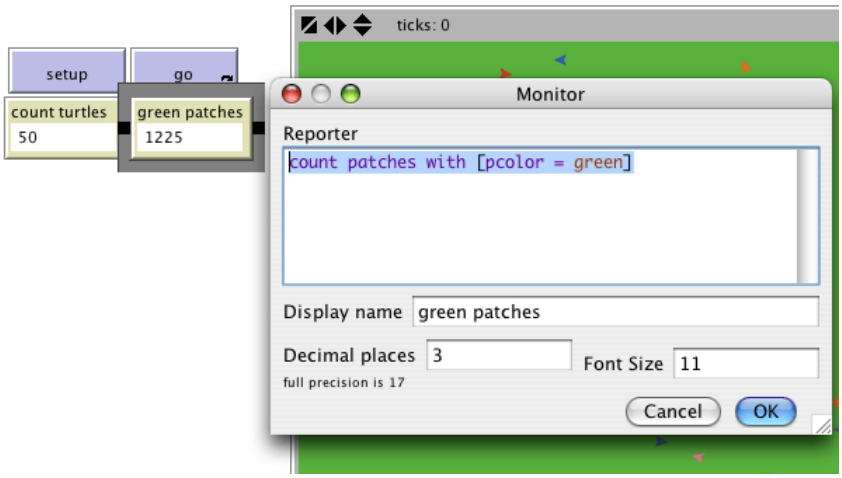

Dans ce que vous venez d'écrire, `turtles` est un « ensemble d'agents », ici, l'ensemble de toutes les tortues et `count` nous indique combien d'agents il y a dans cet ensemble.

Construisons maintenant le second moniteur :

- Créez un moniteur en utilisant le bouton " Add " de la barre d'outils et cliquez dans une zone non occupée de l' " Interface " pour le placer (Ce bouton permet d'ajouter un contrôle de type affiché dans le menu placé à sa droite.).

Une boîte de dialogue apparaît.

- Dans la zone " Reporter " de la boîte de dialogue, écrivez : `count patches with [pcolor = green]`
- Dans la zone " Display name " de la boîte de dialogue, écrivez : `Carrés d'herbe`
- Pressez le bouton " OK " pour fermer la boîte de dialogue.




Ici, nous utilisons encore une fois `count` pour voir combien il y a d'agents dans un ensemble d'agents. `patches` est l'ensemble de tous les patches, mais nous ne voulons justement pas savoir combien il y a de patches au total; nous voulons savoir combien de patches ont de l'herbe, donc combien il y a de patches verts. C'est ce que fait `with`. Il construit un sous-ensemble d'agents constitué uniquement des agents pour lesquels la condition formulée entre crochets est vraie. Cette condition étant `pcolor = green`, `with` nous retourne bien le nombre de patches verts.

Il y a maintenant deux moniteurs qui vont nous indiquer combien de tortues et combien de patches verts nous avons dans le but de nous aider à suivre ce qui se passe au cours de la simulation. Les valeurs affichées dans les moniteurs sont automatiquement mises à jours tout au long de la simulation.

➢ Utilisez les boutons " PREPARER " et " GO " et observez comment l'affichage des valeurs change dans les moniteurs.

## Commutateurs et étiquettes

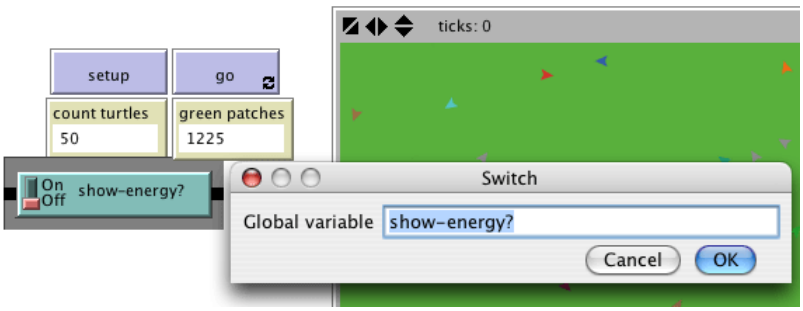
Les tortues ne se contentent pas de changer la couleur des patches en noir, elles gagnent et perdent aussi de l'énergie. Essayez d'utiliser un moniteur tortue pour voir l'augmentation ou la diminution d'énergie d'une tortue au cours de la simulation.

Ce serait une bonne chose de pouvoir prendre connaissance à tout moment de l'énergie propre à chaque tortue. Et c'est exactement ce que nous allons faire en ajoutant un commutateur (switch) afin de pouvoir activer ou désactiver l'affichage de ces informations.

➢ Pour créez un commutateur, sélectionnez la commande " Switch " dans le menu déroulant de la barre d'outils du panneau " Interface " et cliquez dans une zone non occupée de l'interface pour placer ce commutateur.

Une boîte de dialogue apparaît.

➢ Écrivez `AFFICHER_ENERGIE?` dans le champ " Global variable " de la boîte de dialogue et n'oubliez pas de mettre le point d'interrogation à la suite du nom. (Voir l'image ci-dessous.)




➢ Retournez maintenant à la procédure `mange-herbe` (dans le panneau " Procedures ").
➢ Modifiez la procédure `mange-herbe` comme suit :

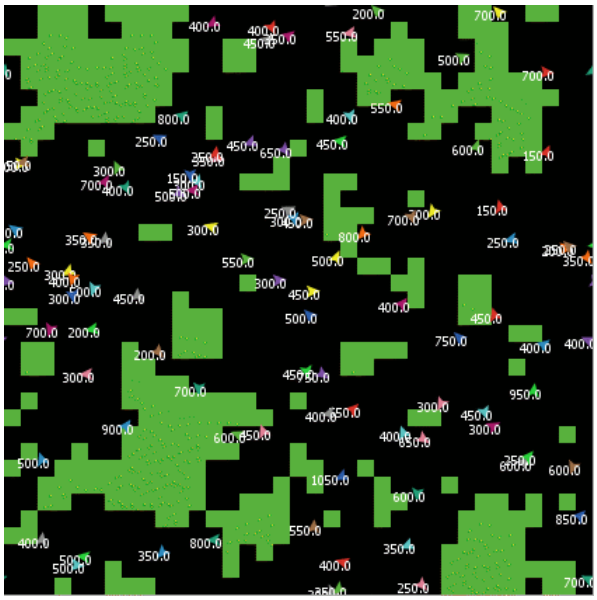
```
to mange-herbe
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energie (energie + 10)
      ]
      ifelse AFFICHER_ENERGIE?
      [ set label energie ]
      [ set label "" ]
    ]
end
```

La procédure `mange-herbe` utilise la commande `ifelse`. Examinez soigneusement le code. Chaque tortue, quand elle exécute cette nouvelle commande, teste la valeur de `AFFICHER_ENERGIE?` (qui est déterminée par la position du commutateur). Si le commutateur est enclenché, la comparaison est vraie et la tortue exécute les commandes qui sont à l'intérieur de la première paire de crochets. Dans ce cas, elle transmet la valeur de l'énergie de la tortue à l'étiquette de cette tortue. Si la comparaison est fausse (le commutateur est déclenché), la tortue exécute les commandes placées dans le seconde paire de crochets. Dans ce cas, elle efface le texte de l'étiquette (en demandant à l'étiquette d'afficher une chaîne de caractères vide)

En NetLogo, un bout de texte est appelé une ***chaîne de caractères*** ou plus simplement une ***chaîne*** (string). Une chaîne est une suite de lettres et/ou d'autres caractères, écrits entre des guillemets anglais doubles. Ici, nous avons deux doubles guillemets qui se suivent sans aucun espace entre eux. C'est une ***chaîne vide***. Si une étiquette de tortue reçoit une chaîne vide, elle n'affiche rien et aucun texte n'est attaché à la tortue.

- Testez ces modifications dans le panneau " Interface " en faisant tourner la simulation (utilisez les boutons " PREPARER " et " GO ", puis enclenchez et déclenchez le commutateur `AFFICHER_ENERGIE?`).

Quand ce commutateur est enclenché, vous verrez l'énergie de chaque tortue augmenter chaque fois qu'elle mange de l'herbe. Vous verrez aussi son énergie diminuer chaque fois qu'elle se déplace.



## Encore des procédures

Maintenant que nos tortues peuvent manger, faisons en sorte qu'elles puissent aussi se reproduire et même mourir. Il serait bien aussi que l'herbe puisse repousser. Nous allons ajouter ces trois comportements en créant trois procédures distinctes, une pour chaque comportement.

- Allez dans le panneau " Procedures ".
- Complétez la procédure GO de la manière suivante :

```
to GO
  bouge-tortues
  mange-herbe
  reproduit-mouton
  teste-mort
  repousse-herbe
end
```

- Ajouter les procédures `reproduit-mouton`, `teste-mort` et `repousse-herbe` ci-dessous :

```
to reproduit-mouton
  ask turtles [
    if energie > 50 [
```

```
      set energie energie – 50
      hatch 1 [ set energie 50 ]
    ]
  ]
end

to teste-mort
  ask turtles [
    if energie <= 0 [ die ]
  ]
end

to repousse-herbe
  ask patches [
    if random 100 < 3 [ set pcolor green ]
  ]
end
```

Chacune de ces procédures utilise la commande `if`. Chaque tortue, quand elle exécute `reproduire-mouton`, teste la valeur de la variable `energie` de la tortue. Si cette valeur est plus grande que 50, alors la tortue exécute les commandes placées à l'intérieur de la première paire de crochets. Dans ce cas, ces commandes diminuent l'énergie de la tortue de 50 unités puis font naître une nouvelle tortue dotée de 50 unités d'énergie. La commande `hatch` est une primitive NetLogo qui s'utilise selon le schéma : `hatch nombre [ commandes ]`. Cette tortue crée `nombre` nouvelles tortues, chacune identique à son parent, et demande à la (aux) nouvelle(s) tortue(s) qui a (ont) été créée(s) d'exécuter les `commandes`. Vous pouvez utiliser ces commandes pour donner aux nouvelles tortues d'autres couleurs, d'autres caps ou n'importe quoi d'autre. Dans notre cas, nous n'exécutons qu'une seule commande : nous donnons à la nouvelle tortue 50 unités d'énergie.

Quand chaque tortue exécute `teste-mort`, elle fait un test pour voir si son énergie est inférieure ou égale à 0. Si le test est vrai, la tortue doit exécuter `die` (c'est-à-dire mourir). (`die` est une primitive de NetLogo.)

Quand chaque patch exécute `repousse-herbe`, il fait un test pour voir si un nombre entier, généré aléatoirement entre 0 et 99, est inférieur à 3. Si c'est le cas, le patch retrouve une couleur verte. La probabilité de ce changement de couleur est en moyenne de 3%. Un patch « passe » 3% de son temps à verdir, puisqu'il y a trois nombres (0, 1 et 2), sur les 100 possibles, qui sont inférieurs à 3.

➢ Revenez dans le panneau " Interface " et pressez les boutons " PREPARER " et " GO ".

Vous devriez maintenant observer un certain nombre de comportements intéressants au cours de la simulation. Certaines tortues meurent, d'autres sont créées (naissent) et l'herbe de certains patches repousse. C'est exactement ce que nous avons essayé d'obtenir.

Si vous observez les moniteurs pendant un certain temps, vous verrez que les valeurs affichées dans "Nombre d'animaux" et dans "Carrés d'herbe" varient au cours de la simulation. Est-ce que ces fluctuations obéissent à un certain schéma et ce schéma est-il prévisible? Y a-t-il un lien de cause à effet entre ces deux variables?

Il serait intéressant d'avoir un moyen plus pratique de suivre les fluctuations de certaines variables du modèle au cours du temps. NetLogo nous offre la possibilité d'afficher les données sous forme de courbes en fonction du temps, et ce pendant tout le déroulement de la simulation. Ce sera notre prochaine étape.

## Représentation graphique

Pour pouvoir afficher des graphiques, nous devons d'abord créer un **traceur** dans le panneau "Interface" et y faire quelques réglages. Nous devrons aussi ajouter une procédure supplémentaire dans le panneau "Procedures", procédure dont la fonction sera de tenir le graphique à jour.

Commençons par le panneau "Procedures" :

- Modifiez PREPARER pour qu'elle appelle une nouvelle procédure, `dessine-graphes`, que nous créerons dans un moment :

```
to PREPARER
  clear-all
  prepare-patches
  prepare-tortues
  dessine-graphes
end
```

- Modifiez aussi GO pour appeler la procédure `dessine-graphes` :

```
to GO
  bouge-tortues
  mange-herbe
  reproduit-mouton
  teste-mort
  repousse-herbe
  dessine-graphes
end
```

- Ajoutez maintenant la nouvelle procédure. Nous allons représenter graphiquement le nombre de tortues et le nombre de patches verts en fonction du temps. À chaque cycle (un cycle correspondant à une seule exécution de la procédure GO ), les valeurs calculées sont ajoutées aux graphiques du traceur.

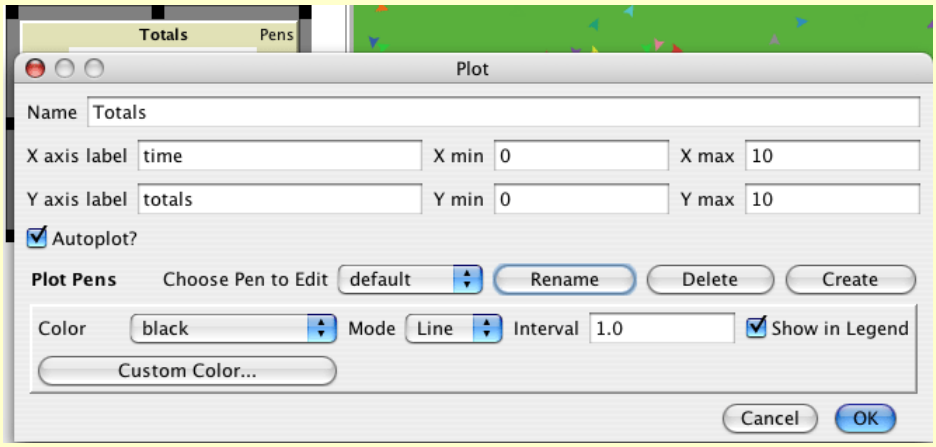
```
to dessine-graphes
  set-current-plot "Totaux"                  ;; le traceur
  set-current-plot-pen "turtles"             ;; le crayon pour les tortues
  plot count turtles                         ;; dessiner la courbe des tortues
  set-current-plot-pen "grass"               ;; le crayon pour l'herbe
  plot count patches with [pcolor = green]   ;; dessiner la « quantité » d'herbe
end
```

Notez que nous utilisons la commande `plot` pour ajouter le point suivant au graphique. Toutefois, avant de pouvoir le faire, nous devons dire deux choses à NetLogo. Premièrement, nous devons spécifier quel traceur nous voulons utiliser (puisque par la suite notre modèle pourrait avoir plus d'un traceur), et deuxièmement nous devons spécifier quel crayon nous voulons utiliser pour tracer la courbe (nous utiliserons deux crayons dans ce traceur).

La commande `plot` déplace le crayon de la courbe courante sur le point dont la coordonnée X est d'une unité supérieure à la coordonnée X du point précédent et dont la coordonnée Y correspondant à la valeur transmise à la commande `plot` (dans le premier cas, le nombre de tortues, dans le second cas, le nombres de patches verts). Quand les crayons se déplacent, ils dessinent chacun leur propre ligne.

Pour que la commande `set-current-plot "Totaux"` puisse fonctionner, vous devez ajouter un traceur dans le panneau "Interface" du modèle, puis le paramétrer de manière à ce que son nom soit le même que le nom utilisé dans la procédure, ici `Totaux`. Même un espace non prévu dans le nom suffit à le faire rejeter : il doit être orthographié exactement de la même manière aux deux endroits.

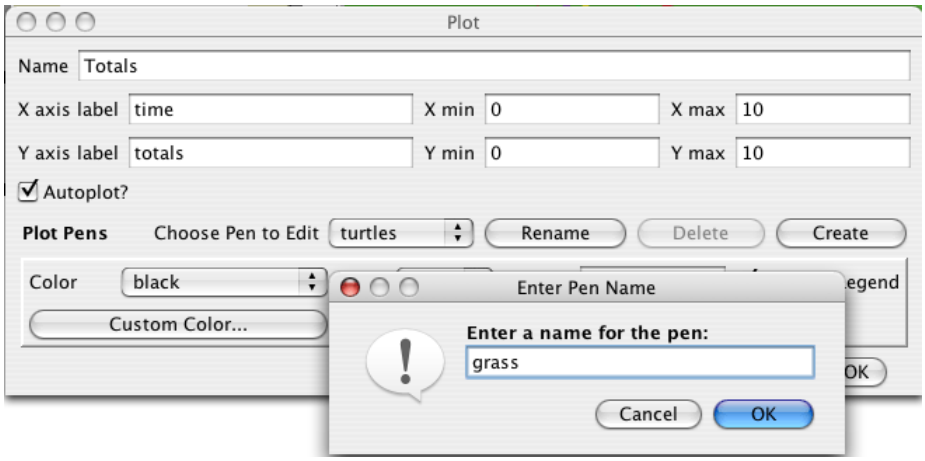
- Créez un traceur en utilisant la commande " Plot " du menu " Add " de la barre d'outils puis cliquez dans un endroit libre de l'interface.
- Écrivez le nom `Totaux` dans le champ " Name " (voir l'image ci-dessous).




- Écrivez `temps` dans le champ " X axis label ".
- Écrivez `totaux` dans le champ " Y axis label ".

Vous devez ensuite créer les deux crayons.

- Dans la boîte de dialogue " Plot " toujours ouverte, pressez le bouton " Create " pour créer un nouveau crayon.
- Entrez `turtles`, le nom de ce crayon, dans le champ " Enter a name for the pen: " de la boîte " Enter Pen Name " et pressez " OK " (voir image ci-dessous).
- Pressez encore une fois le bouton " Create " de la boîte de dialogue " Plot " pour créer un second crayon.
- Entrez `grass`, le nom de ce crayon, dans le champ " Enter a name for the pen: " de la boîte " Enter Pen Name " et pressez " OK " (voir image ci-dessous).
- Cliquez le menu " Color " de la boîte " Plot " et sélectionnez " green " dans le menu pour que la courbe de l'herbe soit verte.
- Cliquez " OK " pour valider et fermer la boîte de dialogue " Plot ".

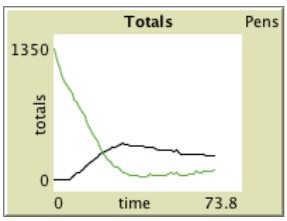



Notez que, quand vous créez et paramétrez un traceur, vous pouvez aussi spécifier les valeurs maximales et maximales pour les axes X et Y. Vous pouvez aussi laisser la coche dans la case à cocher "Autoplot?" de manière à ce que si les valeurs à représenter dépassent les valeurs minimales et/ou maximales de ces axes, l'échelle de ceux-ci s'adapte automatiquement afin que toutes les données puissent êtres affichées.

Initialisez (bouton " PREPARER ") et lancez (bouton " GO ") la simulation encore une fois.

Vous pouvez maintenant voir les graphiques se dessiner au fur et à mesure que la simulation se déroule. Le traceur devrait ressembler à celui montré ci-dessous, mais il ne doit pas en être la copie conforme.

Nous avions laissé " Autoplot? " actif, ce qui permet au graphe de se réajuster lorsque les valeurs deviennent trop grandes.




**Remarque** : La case "Show legend" de la boîte de dialogue "Plot" permet d'activer ou de désactiver l'affichage des légendes des courbes dans la marge droite du traceur. Cette boîte de dialogue peut en tout temps être ouverte par un clic droit sur le traceur et la sélection de la commande "Edit..." dans le menu qui s'affiche.

Faites tourner la simulation plusieurs fois pour voir quels aspects des courbes sont constants et quels aspects changent au gré des simulations.

## Le compteur de cycles

Pour pouvoir comparer les graphes obtenus par plusieurs simulations successives, il est souvent utile de comparer des simulations de même durée. Savoir comment stopper ou démarrer une action à un moment spécifique peut y contribuer en arrêtant la simulation au même endroit lors chaque passe. Garder la trace du nombre de fois que la procédure `GO` a été exécutée est un moyen utile pour intercaler de telles actions.

Pour garder cette trace, nous utiliserons le compteur de cycles interne de NetLogo.

➢ Modifiez la procédure `GO` de la manière suivante :

```
to go
  if ticks >= 500 [ stop ]
  bouge-tortues
  mange-herbe
  reproduit-mouton
  teste-mort
  repousse-herbe
  tick                ;; faire avancer d'un pas le compteur de cycle
  dessine-graphes
end
```

➢ Initialisez (bouton " PREPARER" ) et lancez (bouton " GO ") la simulation.

Maintenant, la simulation et le dessin des graphiques ne se poursuivent plus jusqu'à l'infini. Ils s'arrêtent automatiquement lorsque le compteur de cycles de la barre d'outils du panneau "Interface" atteint 500.

La commande `tick` avance le compteur de cycles par pas de 1. `ticks` est un reporter qui retourne la valeur courante du compteur de cycle. `clear-all` de la procédure `PREPARER` se charge de remettre de compteur de cycles à 0 quand on démarre une nouvelle simulation.

Notez que nous avons mis `tick` avant `dessine-graphes`. Nous procédons ainsi car si nous écrivons du code pour un traceur qui utilise la valeur du compteur de cycle, il verra la nouvelle valeur, pas l'ancienne. (Dans ce tutoriel, nous ne voulons pas écrire un code de ce type, mais quoi qu'il en soit, c'est une bonne habitude que d'appeler `tick` après que les agents aient fait leur travail mais avant que vous n'actualisiez les graphes.)

Maintenant que votre modèle utilise les cycles, vous voudrez probablement utiliser le menu placé dans le haut du panneau "Interface" pour passer d'une mise à jour continue ("continuous") à une mise à jour basée sur les cycles ("on ticks"). Cela signifie que NetLogo ne mettra à jour (c'est-à-dire ne redessinera) la Vue (autrement dit la zone dans laquelle vous voyez les agents) qu'à la fin de chaque cycle, jamais au milieu d'un

cycle. Ce réglage permet une simulation plus rapide et assure une apparence consistante des résultats (puisque les rafraîchissements ne se font qu'à des moments précis). Voir le Guide de Programmation pour une discussion complète et approfondie du rafraîchissement de la vue.

## Encore quelques détails

Plutôt que d'avoir toujours 100 tortues, vous pouvez spécifier un nombre de tortues variable.

- Créez un curseur de variable appelé " Nombre_tortues ", au moyen de la commande " Monitor " de la barre d'outils, puis cliquez dans une zone libre de l'interface. Essayez de modifier les valeurs maximales et minimales du curseur.
- Puis, dans le procédure `prepare-tortues`, remplacez `create-turtles 100` de la manière suivante :

```
to prepare-tortues
  create-turtles Nombre_tortues
  ask turtles [ setxy random-xcor random-ycor ]
end
```

Testez cette modification et comparez comment le fait d'avoir plus ou moins de tortues initiales affecte les courbes en fonction du temps.

Ne serait-il pas intéressant de pouvoir modifier les quantités d'énergie que les tortues gagnent ou perdent quand elles mangent de l'herbe ou se reproduisent?

- Créez un curseur appelé `energie-herbe`.
- Créez un autre curseur appelé `energie-reproduction`.
- Puis, dans la procédure `mange-herbe,` apportez les modifications suivantes :

```
to mange-herbe
  ask turtles
  [
    if pcolor = green
    [
      set pcolor black
      set energie (energie + energie-herbe)
    ]
    ifelse AFFICHER_ENERGIE?
    [ set label energie ]
    [ set label "" ]
  ]
end
```

- Et modifiez la procédure `reproduit-mouton` de la manière suivante :

```
to reproduit-mouton
  ask turtles [
    if energie > energie-reproduction [
      set energie energie – energie-reproduction
      hatch 1 [ set energie energie-reproduction ]
    ]
  ]
end
```

Finalement, quel autre curseur pourriez-vous ajouter pour agir sur la repousse de l'herbe? Y a-t-il des règles que vous pourriez ajouter aux déplacements des tortues ou au comportement des tortues nouveaux-nés qui ne s'appliqueraient qu'à certains moments? Essayez de les écrire.

## Et ensuite ?

Vous avez maintenant un modèle simple d'un écosystème. Les patches font pousser de l'herbe, les tortues se promènent, mangent de l'herbe, se reproduisent et meurent. Vous avez créé une interface contenant des boutons, des curseurs, des commutateurs et un traceur. Vous avez même écrit une série de procédures pour donner quelque chose à faire aux tortues.

C'est donc ici que ce tutoriel s'achève.

Si vous voulez consulter plus de documentation au sujet de NetLogo, la section Guide de l'interface de ce manuel vous emmène visiter tous les éléments de l'interface de NetLogo et vous en explique le fonctionnement. Pour une description détaillée sur la manière d'écrire des procédures, référez-vous au Guide de la programmation. Toutes les primitives sont décrites dans le Dictionnaire NetLogo.

Vous pouvez aussi continuer d'expérimenter et de compléter ce modèle si vous le désirez, en testant différentes variables et différents comportements pour les agents.

Mais vous pouvez aussi revoir le premier modèle de ces tutoriels, la Prédation Loups-Moutons. Il s'agit du modèle utilisé dans le `Tutoriel 1`. Dans ce modèle Prédation Loups-Moutons, vous avez vu des moutons se déplacer, consommer des ressources qui sont occasionnellement renouvelées (l'herbe), se reproduire dans certaines conditions, et mourir quand ils viennent à manquer de ressources. Mais ce modèle possède aussi un autre type de créatures, les loups. L'entrée en scène des loups demande quelques procédures supplémentaires et l'utilisation de quelques nouvelles primitives. Les loups et les moutons sont deux différentes « races » de tortues. Pour voir comment utiliser les races, étudiez le modèle Prédation Loups-Moutons.

Vous avez encore la possibilité d'étudier d'autres modèles (y compris les nombreux modèles de la section "Code Examples" de la Bibliothèque des modèles) ou même d'aller de l'avant et de construire vos propres modèles. Vous n'avez même pas besoin de modéliser quoi que ce soit. Il peut aussi être intéressant de simplement observer les patches et les tortues former des dessins, d'essayer de créer des jeux, ou encore de faire « n'importe quoi ».

Espérons que vous avez appris certaines choses, aussi bien en langage NetLogo que dans la manière de s'y prendre pour créer des modèles. Toutes les procédures créées au cours de ce tutoriel sont regroupées (et commentées) ci-dessous.

## Appendice : le code complet

Le code complet de ce modèle est aussi disponible dans la Bibliothèque des modèles NetLogo, dans la section "Code Examples". Il est intitulé "Tutorial 3". Sa version française "`Tutorial 3_fr.nlogo`"se trouve dans le dossier "`Code Examples-fr`".

Notez que ce code est plein de commentaires, qui commencent par un point-virgule. Les commentaires permettent de placer des explications dans le code lui-même, aux endroits où elles sont le plus utiles. Vous pouvez utiliser les commentaires pour aider les autres à comprendre le fonctionnement de votre modèle ou tout simplement comme notes pour votre usage personnel.

Dans le panneau "Procedures", les commentaires sont affichés en gris afin qu'ils puissent être facilement repérés.

```
turtles-own [energie] ;; pour savoir quand la tortue est prête
                      ;; pour se reproduire et quand elle doit mourir

to PREPARER
  clear-all
  prepare-patches
  prepare-tortues
  dessine-graphes
end

to prepare-patches
  ask patches [ set pcolor green ]
end

to prepare-tortues
  create-turtles Nombre_tortues ;; utilise la valeur du curseur Nombre_tortues pour créer les tortues
  ask turtles [ setxy random-xcor random-ycor ]
end

to GO
  if ticks >= 500 [ stop ]  ;; stoppe après 500 cycles
  bouge-tortues
  mange-herbe
  reproduit-tortue
  teste-mort
  repousse-herbe
  tick                    ;; augmente le compteur de cycle de 1 à chaque cycle
  dessine-graphes
end

to bouge-tortues
  ask turtles [
    right random 360
    forward 1
    set energie energie - 1  ;; quand la tortue se déplace, elle perd 1 unité d'énergie
  ]
end

to mange-herbe
  ask turtles [
    if pcolor = green [
      set pcolor black
           ;; la valeur du curseur energie-herbe est ajoutée à energie
      set energie (energie + energie-herbe)
    ]
  ifelse AFFICHER_ENERGIE?
    [ set label energie ] ;; l'étiquette est réglée pour afficher la valeur de energie
    [ set label "" ]      ;; l'étiquette reçoit une chaîne vide pour s'effacer
  ]
end

to reproduit-tortue
  ask turtles [
    if energie > energie-reproduction [
      set energie energie - energie-reproduction   ;; ôte energie-reproduction pour une naissance
      hatch 1 [ set energie energie-reproduction ] ;; donne cette energie-reproduction au nouveau-né
    ]
  ]
end

to teste-mort
  ask turtles [
    if energie <= 0 [ die ] ;; enlève la tortue si elle n'a plus d'énergie
  ]
end
```

```
to repousse-herbe
  ask patches [                            ;; 3 cycles sur 100, la patch reçoit une couleur verte
    if random 100 < 3 [ set pcolor green ]
  ]
end

to dessine-graphes
  set-current-plot "Totaux"                ;; quel traceur utiliser
  set-current-plot-pen "tortues"           ;; quel crayon utiliser
  plot count turtles                       ;; ce qui sera dessiné avec le crayon courant
  set-current-plot-pen "herbe"             ;; quel crayon utiliser
  plot count patches with [pcolor = green] ;; ce qui sera dessiné avec le crayon courant
end
```

# Chapitre 11

# Guide de l'interface

Cette section de Manuel de l'utilisateur présente tous les éléments de l'interface NetLogo et en explique les fonctions.

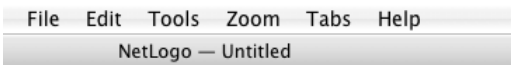
Avec NetLogo, vous avez la possibilité de travailler avec les modèles qui se trouvent dans la Bibliothèque des modèles (Models Library), de modifier ou compléter des modèles existants ou même de créer vos propres modèles. L'interface de NetLogo a été conçue pour remplir toutes ces tâches.

L'interface peut être divisée en deux parties : les menus NetLogo et la fenêtre NetLogo, cette dernière étant elle-même composée de trois panneaux.



## Les menus

Quand vous faites tourner l'application NetLogo sur un système Mac, la barre des menus est située au haut de l'écran. Sur les autres plates-formes, la barre des menus est au sommet de la fenêtre NetLogo.




Remarque : les illustrations proviennent d'un NetLogo tournant sur Mac OS X.

Les commandes proposées par les menus de la barre des menus sont listées dans le tableau suivant.

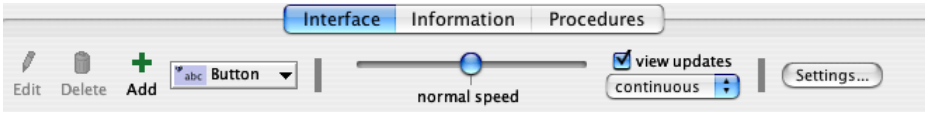
## Tableau : les menus de NetLogo

| Menu | Commande | Action |
|---|---|---|
| **File** | New | Commencer un nouveau modèle. |
| | Open | Ouvrir tout modèle NetLogo présent dans l'ordinateur. |
| | Models Library | Une collection de modèles de démonstration. |
| | Save | Enregistrer le modèle courant. |
| | Save As | Enregistrer le modèle courant sous un autre nom. |
| | Save As Applet | Pour sauvegarder une page au format HTML avec le modèle sous forme d'« applet » Java. |
| | Print | Envoyer le contenu du panneau courant à l'imprimante. |
| | Export World | Enregistrer toutes les variables, l'état actuel des tortues et des patches, le dessin, les traceurs de courbes, la zone de sortie et l'état du générateur de nombres aléatoires dans un fichier. |
| | Export Plot | Enregistrer les données du traceur de courbes dans un fichier. |
| | Export All Plots | Enregistrer les données de tous les traceur de courbes dans un fichier. |
| | Export View | Enregistrer une image de la vue courante (2D / 3D) dans un ficher (format .png). |
| | Export Interface | Enregistrer une image de panneau Interface courant (format .png). |
| | Export Output | Enregistrer le contenu de la zone de sortie ou la zone de sortie du Centre de commande dans un fichier. |
| | Import World | Charger un fichier qui avait été enregistré avec "Export World". |
| | Import Patch Colors | Charger une image dans la couche des patches; voir la commande import-pcolors. |
| | Import Patch Colors RGB | Charger une image dans la couche des patches en utilisant les couleurs RGB; voir la commande import-pcolors-rgb. |
| | Import Drawing | Charger une image dans la couche dessin, voir la commande import-drawing. |
| | Import HubNet Client Interface | Charger l'interface d'un autre modèle dans l'éditeur "HubNet Client". |
| | Quit | Quitter NetLogo. (Sur Mac, cette commande se trouve dans le menu NetLogo.) |
| **Edit** | Cut | Couper le texte sélectionné et le sauvegarder dans le presse-papier. |
| | Copy | Copier le texte sélectionné dans le presse-papier. |
| | Paste | Coller le texte du presse-papier à l'emplacement du curseur. |
| | Delete | Supprimer le texte sélectionné. |
| | Undo | Annuler la dernière action d'édition de texte. |
| | Redo | Annuler la dernière action d'annulation. |
| | Select All | Sélectionner tout le texte dans la fenêtre active. |
| | Find | Chercher un mot ou une séquence de caractères dans le panneau "Information" ou "Procedures." |
| | Find Next | Chercher la prochaine occurrence du dernier mot ou de la dernière séquence utilisée avec "Find". |
| | Shift Left / Shift Right | À utiliser dans le panneau "Procedures" pour modifier le niveau d'indentation du code. |
| | Comment / Uncomment | À Utiliser dans le panneau "Procedures" pour ajouter ou supprimer des points-virgules au code (les points-virgules sont utilisés en NetLogo pour marquer les commentaires). |
| | Snap To Grid | Si activé, les nouveaux contrôles (dans le panneau "Interface") sont fixés sur une grille de 5 pixels, ce qui facilite leur alignement. (Note: cette fonctionnalité est désactivée quand on zoome avant ou arrière.) |

| Menu | Commande | Action |
|---|---|---|
| **Tools** | Halt | Stopper l'exécution du code, y compris celui des boutons et du Centre de commande. (Attention : puisque le code est interrompu quoi qu'il soit en train d'exécuter, vous pouvez obtenir des résultats bizarres si vous tentez de relancer le code sans d'abord tout réinitialiser pour repartir avec une nouvelle simulation.) |
| | Globals Monitor | Afficher les valeurs de toutes les variables globales. |
| | Turtle Monitor | Afficher les valeurs de toutes les variables d'une tortue particulière. Vous pouvez aussi éditer les valeurs des variables de la tortue et lui passer des commandes. (Vous pouvez aussi ouvrir un moniteur tortue à partie de la Vue : voir la section Vue ci-dessous.) |
| | Patch Monitor | Afficher les valeurs de toutes les variables d'un patch particulier. Vous pouvez aussi éditer les valeurs des variables du patch et lui passer des commandes. (Vous pouvez aussi ouvrir un moniteur de patch à partie de la vue : voir la section Vue ci-dessous.) |
| | Link Monitor | Afficher les valeurs de toutes les variables d'un lien particulier. Vous pouvez aussi éditer les valeurs des variables du lien et lui passer des commandes. (Vous pouvez aussi ouvrir un moniteur de lien à partie de la vue : voir la section Vue ci-dessous.) |
| | Hide/Show Command Center | Afficher ou cacher le Centre de commande. (Notez que le Centre de commande peut aussi être rendu visible ou caché et redimensionné au moyen de la souris.) |
| | 3D View | Ouvrir la vue 3D. Voir la section Les vues 2D et 3D pour plus d'informations. |
| | Color Swatches | Ouvrir la Palette des couleur. Voir la section Les couleurs du Guide de la programmation pour plus de détails. |
| | Turtle Shapes Editor | Ouvrir l'éditeur des formes de tortues. Voir le Guide de l'éditeur de formes pour plus d'informations. |
| | Link Shapes Editor | Ouvrir l'éditeur des formes de liens. Voir le Guide de l'éditeur de formes pour plus d'informations. |
| | BehaviorSpace | Faire tourner la simulation encore et encore avec différents réglages. Voir le Guide de BehaviorSpace pour plus d'informations. |
| | System Dynamics Modeler | Ouvrir le Modélisateur de systèmes dynamiques. Voir le Guide du modélisateur de systèmes dynamiques pour plus de détails. |
| | HubNet Client Editor | Ouvrir l'éditeur de client HubNet. Voir le Guide de programmation HubNet pour plus de détails. |
| | HubNet Control Center | Ouvrir le Centre de contrôle d'HubNet. (Désactivé s'il n'y a pas d'activité HubNet.). Voir le Guide HubNet pour plus d'informations. |
| **Zoom** | Larger | Accroître la taille de l'écran du modèle. Utile sur un grand moniteur ou avec une projecteur vidéo (*beamer*). |
| | Normal Size | Remettre l'écran du modèle à sa taille normale. |
| | Smaller | Diminuer la taille de l'écran du modèle |
| **Tabs** | | Ce menu présente les combinaisons de touches pour ouvrir les différents panneaux (sur Mac, **Commande+ 1** à **Commande +3**, sur Windows, **Ctrl + 1** à **Ctrl + 3**). |
| **Help** | About NetLogo | Informations concernant la version de NetLogo utilisée (Sur Mac, cette commande se trouve dans le menu NetLogo.) |
| | Look Up In Dictionary | Ouvrir le Dictionnaire dans le navigateur par défaut et affiche les informations concernant la commande ou le reporter sélectionné. (Seulement pour les primitives NetLogo, pas pour celles des extensions.) |
| | NetLogo User Manual | Ouvrir le Manuel de l'utilisateur dans le navigateur web par défaut. |
| | NetLogo Dictionary | Ouvrir le Dictionnaire NetLogo dans le navigateur web par défaut. |

# Les panneaux

Dans le haut de la fenêtre NetLogo se trouvent trois boutons allongés (ou onglets) intitulés "Interface", "Information" et "Procedures" qui permettent d'accéder aux panneaux correspondants. Un seul panneau est visible à la fois, mais vous pouvez passer de l'un à l'autre au moyen de ces boutons-onglets.




Juste en dessous de cette rangée de boutons se trouve une barre d'outils contenant un rangée de boutons. Les boutons disponibles varient en fonction du panneau affiché.

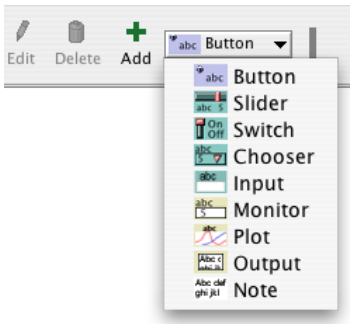
# Le panneau Interface

Le panneau "**Interface**" est l'endroit où vous pouvez voir fonctionner votre modèle. Il offre en plus des outils que vous pouvez utiliser pour inspecter et modifier ce qui se passe dans le modèle.

Quand vous démarrez NetLogo, le panneau Interface ne contient que la **Vue** (View), dans laquelle apparaîtront les tortues et les patches, et le **Centre de commande** (Command Center) qui permet de commander le programme au moyen d'une ligne de commandes.

## Travailler avec les objets de l'interface

La barre d'outils du panneau Interface contient des boutons qui permettent d'éditer, de supprimer et de créer des éléments dans le panneau Interface et d'un menu qui permet de sélectionner les différents types d'objets à créer (tels que des boutons ou des curseurs).




L'utilisation des boutons de la barre d'outils est expliquée ci-dessous.

**Ajouter un élément :** Pour ajouter un élément d'interface, sélectionnez l'élément désiré dans le menu déroulant ("Button" dans la figure ci-dessus). Notez qu'une fois l'élément choisi, le bouton "Add" (ajouter) devient actif. Cliquez ensuite dans la zone blanche située au-dessous de la barre d'outils, là où vous voulez placer l'élément. (Si le menu affiche déjà le type d'élément désiré, il suffit de presser le bouton "Add" plutôt que d'utiliser encore une fois le menu.)

**Sélectionner un élément :** Pour sélectionner un élément de l'interface, dessinez un rectangle de sélection sur au moins une partie de l'élément avec le bouton gauche de la souris. Une large bordure grise munie de poignées de redimensionnement s'affiche autour de l'élément pour montrer qu'il est sélectionné. On peut aussi sélectionner un élément en le cliquant avec le bouton droit (Ctrl-clic avec une souris Mac à un seul bouton) puis choisir "Select" dans le menu contextuel qui s'affiche.

**Sélectionner plusieurs éléments :** Vous pouvez aussi sélectionner plusieurs éléments de l'interface en même temps en les incluant tous dans le rectangle dessiné avec la souris (très utile pour les déplacer d'un seul coup quand on dessine une interface complexe). Si plusieurs éléments sont sélectionnés, l'un d'entre eux (en général le dernier créé parmi les sélectionnés) est l'élément « clé », ce qui signifie que si vous utilisez les boutons "Delete" ou "Edit" de la barre d'outils de l'interface, seul l'élément clé est concerné. La bordure de l'élément clé est plus sombre que la bordure des autres éléments sélectionnés. Pour sélectionner seulement quelques éléments dispersés, cliquez-droit sur le premier, sélectionnez "Select" dans le menu contextuel, pressez "Maj + clic-droit" sur l'élément suivant puis "Select" dans le menu contextuel, et ainsi de suite.

**Désélectionner :** Pour désélectionner tous les éléments de l'interface, cliquez-gauche sur l'arrière-plan blanc du panneau Interface. Pour désélectionner un seul élément, "Ctrl-clic" (Mac) ou "clic-droit" (autres systèmes) puis choisissez "Unselect" dans le menu contextuel qui s'affiche.

**Éditer un élément :** Pour modifier les caractéristiques d'un élément de l'interface, sélectionnez l'élément et pressez le bouton "Edit" de la barre d'outils de l'interface. Vous pouvez aussi double-cliquer sur l'élément une fois qu'il est sélectionné. Une troisième possibilité pour éditer un élément consiste à faire un "Ctrl-clic" (Mac) ou un "clic-droit" (autres systèmes) sur l'élément puis à choisir "Edit" dans le menu contextuel qui s'affiche. Si vous utilisez cette méthode, l'élément n'a pas besoin d'être préalablement sélectionné.

**Déplacer un élément :** Sélectionnez d'abord l'élément sur l'interface puis déplacez-le à l'aide du bouton gauche de la souris. Vous pouvez limiter le déplacement horizontalement ou verticalement en maintenant la touche "Maj" pendant le déplacement.

**Redimensionner un élément :** Sélectionnez d'abord l'élément sur l'interface puis déplacer l'une des « poignées » noires de sa bordure jusqu'à obtenir les dimensions voulues.

**Supprimer un élément :** Sélectionnez l'élément ou les éléments que vous voulez supprimer puis pressez le bouton "Delete" dans la barre d'outils de l'interface. Vous pouvez aussi supprimer un élément par un "Ctrl-clic" (Mac) ou un "clic-droit" (autres systèmes) puis choisir "Delete" dans le menu contextuel qui s'affiche. Si vous utilisez cette dernière méthode, il n'est pas nécessaire de d'abord sélectionner l'élément.

Pour en savoir plus sur les différents éléments de l'interface, consultez le tableau ci-dessous.

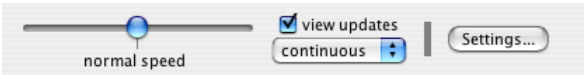
**Tableau : Les différents éléments du panneau Interface**

| Icône & nom | Description |
|---|---|
| Button | Les ***boutons*** (Button) peuvent être soit des boutons « une-fois » (*once-only*), soit des boutons « pour-toujours » (*forever*). Quand l'utilisateur clique un bouton "once-only", celui-ci exécute ses instructions une seule fois. Un bouton "pour-toujours" (signalé par deux flèches courbes tournant en rond dans son coin inférieur droit) reste enfoncé une fois cliqué (il est alors noir) et exécute ses instructions encore et encore jusqu'à ce que l'utilisateur le clique une seconde fois pour stopper l'action ou jusqu'à ce que l'action soit stoppée par le modèle lui-même. (Si les instructions sont nombreuses et prennent du temps, le programme termine les actions en cours avant de remettre le bouton en position déclenchée.) Si vous avez assigné une touche à un bouton, une pression sur la touche correspondante du clavier (quand l'interface à le focus) produit le même résultat qu'un clic sur le bouton. Les boutons associés à une touche affichent la lettre correspondante dans leur coin supérieur droit. Si le curseur d'entrée se trouve dans un autre élément de l'interface, par exemple dans la ligne de commande du Centre de commande, une pression sur la touche correspondante du clavier n'enclenche (ou ne déclenche) pas le bouton. Dans ce cas, la lettre dans le coin supérieur du bouton est gris pâle. Pour activer l'action des touches, il suffit de cliquer dans l'arrière-plan blanc du panneau Interface. |
| Slider | Les ***curseurs*** (Sliders) définissent des variables globales, variables qui sont accessibles à tous les agents. Ils sont utilisés dans les modèles en tant que moyen permettant de modifier rapidement la valeur d'une variable sans avoir à re-coder la procédure à chaque fois. Ils permettent à l'utilisateur de simplement déplacer le curseur sur la valeur désirée et d'observer ce qui se passe au cours de la simulation. (*) |
| Switch | Les ***commutateurs*** (Switches) sont la représentation visuelle d'une variable booléenne vrai/faux (true/false). L'utilisateur peut mettre cette variable à `true` ou à `false` en cliquant l'interrupteur ("On" = `true`, "Off" = `false`). (*) |

| Icône & nom | Description |
|---|---|
| Chooser | Les ***sélecteurs*** (Choosers) permettent à l'utilisateur de sélectionner une valeur pour une variable globale, valeur à choisir dans une liste de valeurs prédéfinies. La liste est présentée sous la forme d'un menu déroulant. (*) |
| Input | Les ***boîtes d'entrée*** (Input) représentent des variables globales pouvant contenir des chaînes ou des nombres. L'auteur du modèle choisit, à l'aide du bouton-menu "Type" quel type de valeurs l'utilisateur pourra entrer. Ces types sont : "`Number`" (un nombre ou une expression numérique constante), "`String`" (une chaîne de caractères normale), "`String (reporter)`" (une chaîne d'instructions pour un reporter), "`String (commands)`" (une chaîne de commandes) et "`Color`" (un choix de couleurs). La syntaxe des chaînes destinées aux reporters ou aux commandes est testée avant qu'elles ne soient envoyées à leur destinataire. Les boîtes d'entrée numériques lisent n'importe quel type d'expression numérique constante, ce qui offre une plus grande liberté pour spécifier des nombres précis qu'un curseur. L'entrée de type "`Color`" permet à l'utilisateur d'ouvrir la palette des couleurs de NetLogo pour y choisir la couleur désirée. |
| Monitor | Les ***moniteurs*** (Monitor) affichent la valeur de n'importe quelle expression. L'expression peut être une variable, une expression complexe ou un appel à un reporter. Le contenu des moniteurs est remis automatiquement à jour plusieurs fois par seconde. |
| Plot | Les ***traceurs*** (Plots) permettent de représenter graphiquement, en temps réel, des données calculées par le programme (autrement dit l'état et l'évolution de certaines variables) au cours de la simulation. |
| Output | La ***zone de sortie*** (Output) est une zone de texte avec défilement qui peut être utilisée pour créer un journal résumant l'activité du modèle. Un modèle ne peut avoir qu'une seule zone de sortie. |
| Note | Les ***notes*** (Note) permettent d'ajouter des informations écrites (telles que des instructions pour l'utilisateur) dans le panneau Interface. Le contenu des notes n'est pas modifié (et ne peut pas être modifié) pendant que le modèle tourne. |

(*) **ATTENTION** : le nom de la variable globale, qui sert d'intitulé à l'élément, est utilisé tel quel dans le programme (donc pas d'accent ni d'espace).

Les autres contrôles de la barre d'outils de l'interface permettent de contrôler les mises à jours de la vue ainsi que diverses autres propriétés du modèle.




- Le curseur "normal speed" permet de contrôler la vitesse du déroulement de la simulation, ce qui est très utile car certains modèles « tournent » si vite qu'il est difficile de voir ce qui s'y passe. Vous pouvez ralentir la simulation en déplaçant le curseur vers la gauche (voire la stopper s'il est tout à gauche) ou l'accélérer en le déplaçant vers la droite, ce qui diminue aussi la fréquence des mises à jour de la vue.
- La case à cocher "view updates" (mise à jour de la vue) active ou désactive les mises à jours de la Vue.
- Le bouton-menu du mode de mise à jour permet de sélectionner le mode de mise à jour de la Vue, soit "continuous" (en continu), soit "on ticks" (basée sur les cycles).
- Le bouton "Settings..." (réglages) ouvre la boîte de dialogue "Model settings" qui permet d'éditer plusieurs propriétés du modèle, notamment celles concernant le monde "World", la vue "View" et le compteur de cycles "Tick counter".

La mise à jour continue, "Continuous", signifie que NetLogo met à jour (c'est-à-dire redessine) la Vue plusieurs fois par seconde, sans tenir compte de ce qui se passe dans le modèle. La mise à jour basée sur les cycles, "on ticks", fait que la Vue n'est mise à jour que lorsque le compteur de cycles avance. (Pour une discussion complète des mises à jour de la Vue, voyez le Guide de la programmation.)

## Les Vues 2D et 3D

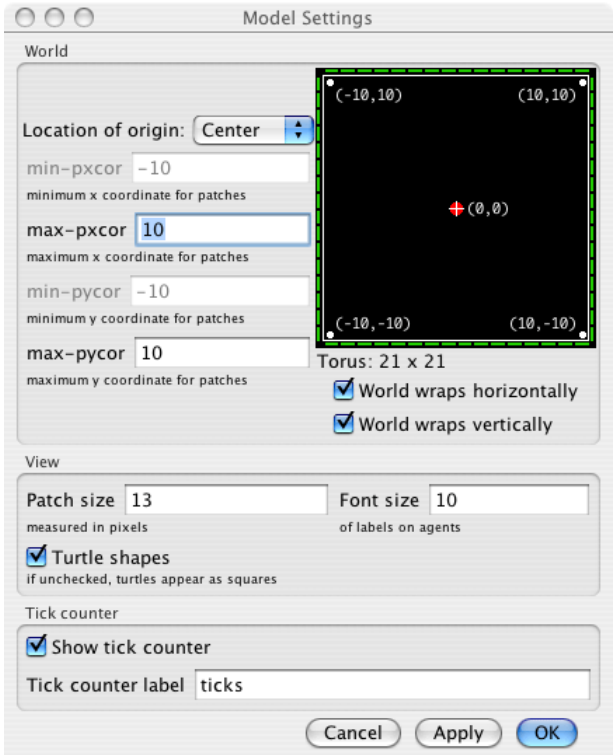
La grand carré noir dans le panneau Interface est la Vue 2D. C'est une représentation visuelle du monde NetLogo des tortues et des patches. Au départ, la Vue 2D est toute noire car les patches sont noirs par défaut et il n'y a pas encore de tortues. Vous pouvez ouvrir la Vue 3D, une autre représentation du monde NetLogo en cliquant le bouton "3D" dans la barre de contrôle de la vue.


Les trois paires de boutons noirs situées dans la partie gauche de la Barre de contrôle permettent de modifier la taille de la Vue. Quand l'origine du monde est au centre du monde, le monde s'accroît par pas de deux, ajoutant une unité au maximum et retranchant une unité au minimum. Si l'un des bords a la coordonnée 0, le monde croît d'une unité dans la l'autre direction, ceci afin de laisser l'origine le long de ce bord. Si l'origine est à un autre endroit (Custom), les flèches noires sont désactivées.

Un certain nombre de réglages sont associés aux Vues. Ils permettent de modifier certains paramètres en utilisant la Barre de contrôle placée dans le haut de la Vue, en ouvrant l'éditeur de la Vue 2D comme expliqué ci-dessus dans la section « Travailler avec les élément de l'Interface » ou en pressant le bouton "Settings..." dans la barre d'outils du panneau Interface.

Notez que la barre de contrôle de la Vue 3D combine le compteur de cycles (ticks) de la barre de contrôle de la Vue 2D avec les contrôles de la partie droite de la barre d'outils du panneau Interface.


Voici la fenêtre des réglages du modèle "Model Settings", accessible en éditant la Vue ou en pressant le bouton "Settings..." dans le barre d'outils du panneau Interface) :




Ces réglages sont répartis en trois groupes : les réglages du Monde "World", de la vue "View" et du Compteur de cycles "Tick counter". Les réglages du Monde affectent les propriétés du monde dans lequel vivent les tortues (les modifier peut nécessiter une ré-initialisation du monde). Les réglages de la Vue et du Compteur de cycles n'affectent que l'apparence, leurs modifications n'influençant pas la suite de la simulation.

Les réglages du Monde permettent de définir les frontières (les limites) et la topologie du monde. Dans la partie supérieure gauche du panneau "World", vous pouvez sélectionner l'emplacement de l'origine du monde "Location of origin:", à choix : "Center" (centre), "Corner" (coin), "Edge" (bord) ou "Custom" (personnalisé). Par défaut, le centre est configuré de telle manière que (0,0) soit au centre du monde et l'utilisateur définit le nombre de patches de ce centre vers les frontières gauche, "min-pxcor", et droite, "max-pxcor", ainsi que le nombre de patches de ce centre vers les frontières inférieure, "min-pycor", et supérieure, "max-pycor", du monde. Par exemple, si vous spécifiez "`max-pxcor = 10`", "min-pxcor" prendra automatiquement la valeur "`-10`" car il y a 10 patches à gauche et 10 patches à droite du patch (0,0).

La configuration "Corner" (coin) permet à l'utilisateur de placer (à l'aide d'un bouton-menu qui apparaît quand on sélectionne "Corner" dans le bouton-menu "Location of origin:") l'origine du monde à l'un de ses quatre coins, à savoir, "Bottom Left" (en bas à gauche), "Top Left" (en haut à gauche), "Top Right" (en haut à droite) et "Bottom Right" (en bas à droite). Vous définissez ensuite les frontières opposées dans les directions x et y. Par exemple, si vous choisissez de mettre l'origine dans le coin inférieur gauche, vous devez spécifier les frontières droite et supérieure (nombres positifs).

La configuration "Edge" (bord) permet de placer (à l'aide du bouton-menu qui apparaît quand on sélectionne "Edge" dans le bouton "Location of origin:") l'origine le long d'une des frontières du monde, à savoir : "Bottom" (en bas), "Top" (en haut), "Right" (à droite) et "Left" (à gauche). Vous devez ensuite définir la frontière qui lui est opposée et les deux frontières sur ses côtés. Par exemple, si vous sélectionnez le bord inférieur du monde, vous devez spécifiez sa frontière supérieure et ses frontières gauche et droite.

Finalement, le mode "Custom" (personnalisé) permet à l'utilisateur de placer l'origine à n'importe quel endroit du monde, mais il faut le que patch 0 0 soit bien placé dans les limites « physiques » de ce monde.

Quand vous modifiez ces réglages, les changements que vous apportez sont automatiquement retranscrits dans le panneau de prévisualisation situé à droite, panneau qui montre le centre du monde (cible rouge), l'état des frontières et les coordonnées des quatre coins par rapport à l'origine spécifiée. La topologie et les dimensions horizontale et verticale sont affichées en dessous de ce panneau.

Sous ce panneau de prévisualisation se trouvent deux cases à cocher, "World wraps horizontally" et "World wraps vertically", permettant de spécifier la topologie (autrement dit l'« enroulement » du monde). Les états de ces deux cases définissent les quatre topologies disponibles, soit : "Torus" (tore) où l'enroulement est possible aussi bien verticalement qu'horizontalement, "Horizontal Cylinder" (cylindre horizontal) où l'enroulement n'est possible que verticalement, "Vertical cylinder" (cylindre vertical) où l'enroulement n'est possible qu'horizontalement et "Box" (boîte) où aucun enroulement n'est possible (autrement dit les quatre frontières sont imperméables). Le nom de la topologie est affiché devant les dimensions du monde et l'état des frontières (franchissables en vert et infranchissables en rouge) est affiché le long des quatre côtés de la prévisualisation. Voir la section Topologie du Guide de la programmation pour des informations supplémentaires.

Les réglages "View" permettent de personnaliser l'aspect de la vue sans modifier le monde. La modification de la vue ne force jamais la ré-initialisation du monde. Pour changer la taille de la vue, ajuster la valeur "Patch size" (taille du patch), mesurée en pixels. Cette action ne modifie pas le nombre de patches, seulement la taille des patches dans la Vue 2D. C'est en quelque sorte une espèce de « zoom » sur le monde. (Notez que ce réglage n'influence pas la Vue 3D, vous pouvez agrandir la vue 3D simplement en augmentant la taille de sa fenêtre.)

La case à cocher "Turtle Shapes" (formes des tortues) permet d'activer ou de désactiver l'affichage des formes des tortues. Quand ces formes sont désactivées, les tortues apparaissent toujours sous la forme de carrés colorés plutôt que sous les formes spéciales que vous avez pu leur donner. Le dessin de carrés demande moins de travail à l'ordinateur, ce qui fait que les programmes tournent plus vite.

Les deux cases à cocher suivantes n'apparaissent que lorsque "Model Settings" est appelé depuis la vue 3D :

- "Smooth edges" (lisser les bords) contrôle l'anti-aliasing dans la Vue 3D. Quand il est activé, les lignes sont lissées (atténuation des bords en escalier), mais la simulation tourne plus lentement.
- "Show wire frame" (afficher en fil de fer) dessine les formes 3D sous forme de fil de fer (seules les arêtes sont affichées, les faces sont transparentes).

Le contrôle "Font size" (taille de la police) permet de spécifier la taille des caractères utilisés dans les étiquettes des agents.

Les réglages du compteur de cycles "Tick counter" permettent d'afficher ou non (case à cocher "Show tick counter") le compteur de cycles et de spécifier le contenu de l'étiquette de ce compteur ("Tick counter label"), compteur qui se trouve dans la barre de titre de la Vue.

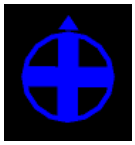
Les moniteurs des tortues, des patches et des liens sont facilement atteignables depuis la Vue. Il suffit de faire "Ctrl-clic" (Mac) ou "clic-droit" (autres systèmes) sur la tortue ou le patch que vous voulez inspecter puis de choisir "turtle n > inspect turtle n" ou "inspect patch x y" dans le menu contextuel qui s'affiche. Le sous-menu "turtle n" vous permet aussi de suivre ou de chevaucher la tortue en sélectionnant "watch turtle n" ou "follow turtle n" dans le sous-menu. (Les moniteurs de tortues, de patches ou de liens peuvent aussi être ouverts à partir du menu "Tools" ou en utilisant la commande `inspect`.)

Certains modèles NetLogo permettent d'interagir avec les tortues et les patches en cliquant et tirant dans la vue avec la souris.

### Utilisation de la la Vue 3D

Au bas de la fenêtre de la Vue 3D se trouvent des boutons permettant de déplacer l'observateur ou de changer le point de vue à partie duquel vous regardez le monde.


Une croix bleue apparaît au point focal courant quand vous manipulez ces réglages. Le petit triangle bleu pointe toujours dans la direction positive de l'axe Y, ceci afin de vous permettre de vous orienter si vous êtes perdus. Ce qu'il est facile de faire!



Pour observer le monde sous un angle différent, pressez le bouton "Orbit" (orbiter) puis cliquez-tirez vers le haut, le bas, la gauche ou la droite. L'observateur continue de regarder le même point (là où se trouve la croix bleue), mais sa position par rapport au plan XY change.

Pour se rapprocher ou s'éloigner du monde ou de l'agent que vous êtes en train de regarder, de suivre ou de chevaucher, pressez le bouton "Zoom" et cliquez-tirez verticalement (de haut en bas et de bas en haut) dans la Vue 3D (la molette centrale fait aussi l'affaire). (Notez que quand vous êtes en mode poursuite ou chevauchée, zoomer vous fait passer de chevauchée à poursuite, puisque chevauchée n'est qu'un cas spécial de poursuite où la distance qui vous sépare de celui que vous suivez est nulle.)

Pour changer la position de l'observateur sans changer la direction d'observation, cliquez le bouton "Move" et déplacez la souris vers le haut, le bas, la gauche ou la droite dans la vue 3D tout en maintenant le bouton gauche de la souris pressé.

Pour activer de transmission au modèle de la position et l'état de la souris dans la Vue, sélectionnez le bouton "Interact" et tout fonctionne comme avec la souris dans la Vue 2D.

Pour remettre l'observateur et le point focal à leurs positions par défaut, pressez le bouton "Reset Perspective" (ou utilisez la commande `reset-perspective`).

### Mode plein-écran

Pour entrer en mode plein-écran, pressez le bouton "Full Screen", pour quitter le mode plein-écran, pressez la touche "Esc".

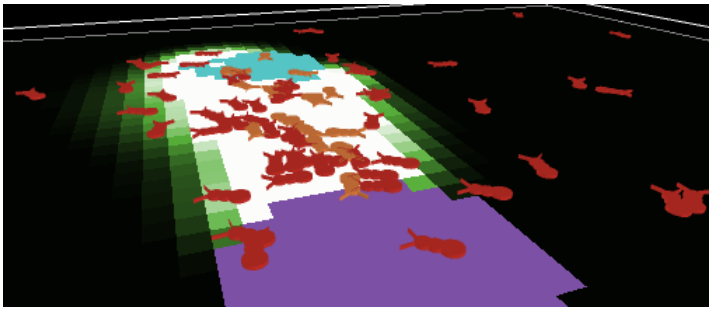
**Note :** Le mode plein-écran ne fonctionne pas sur certains ordinateurs. Cela dépend du type de carte graphique que vous possédez. Voir le chapitre 4 Configuration minimale pour plus de détails.

### Les formes 3D

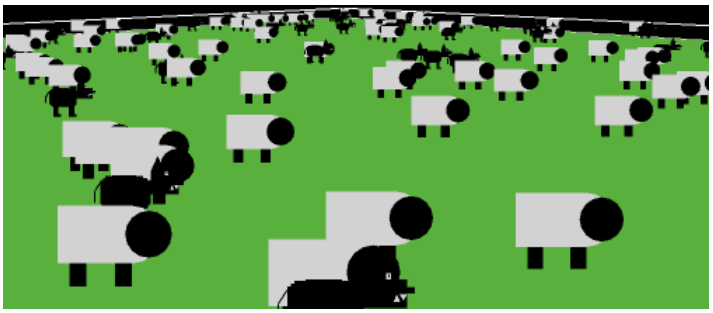
Certaines formes ont une version 3D (un cercle 3D est en réalité une sphère) dans la vue 3D et sont automatiquement transformées dans cette version.

| Nom de la forme | Forme en 3D |
|---|---|
| `default` | tortue en 3D |
| `circle` | sphère |
| `dot` | petite sphère |
| `square` | cube |
| `triangle` | cône |
| `line` | ligne 3D |
| `cylinder` | cylindre 3D |
| `line-half` | demi-ligne 3D |
| `car` | voiture 3D |

Toutes les autres formes sont interprétées à partir de leurs formes 2D. Si une forme est une forme « pivotable », elle est considérée comme vue de dessus et est extrudée verticalement (le long de l'axe Z) comme à travers une forme pour biscuits et orientée parallèlement au plan XY, comme dans le modèle "Ants".



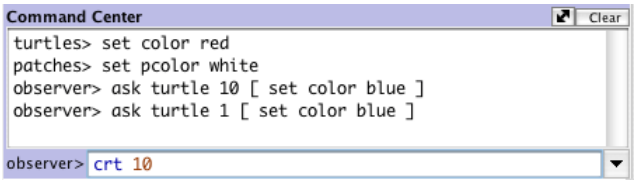
Si une forme n'est pas « pivotable », elle est considérée comme vue de profil et est par conséquent toujours dessinée pour faire face à l'observateur. Elle est donc sans épaisseur, comme les loups et les moutons dans le modèle "Wolf Sheep Predation".



## Le Centre de commande

Le Centre de commande permet à l'utilisateur de commander directement les agents NetLogo sans devoir passer par les procédures du modèle. (Les commandes sont les instructions que vous donnez aux agents du modèle). Cette manière de faire est très utile pour inspecter et manipuler les agents « à la volée ». Il est même possible de travailler avec la Ligne de commandes (zone du Centre de commande dans laquelle l'utilisateur entre les commandes) sans aucun modèle, ce qui est le meilleur moyen de comprendre comment fonctionne NetLogo.

Le Tutoriel 2 : Les Commandes est une introduction à l'utilisation des commandes dans le Centre de commande. Observons plus attentivement ce Centre de commande :




La petite boîte blanche, en dessous de la grande, est la **Ligne de commandes** dans laquelle vous écrivez les commandes. Une fois la (ou les ) commande entrée, pressez Retour ou Entrée pour lancer son exécution.

À gauche de la ligne de commandes se trouve un menu local qui affiche initialement `"observer>"` et qui permet de choisir à quel type d'agents sont destinées les commandes entrées. Les choix possibles sont : `"observer>"` (observateur), `"turtles>"` (tortues), `"patches>"` et `"links>"` (liens).

**Astuce** : on passe rapidement d'un type d'agents à l'autre au moyen de la touche "Tab" du clavier.

### Accéder aux commandes précédentes

Une fois qu'une commande entrée a été envoyée par Retour, elle apparaît dans la grande boîte d'affichage à défilement placée au-dessus de la Ligne de commandes. Vous pouvez utiliser la commande "Copy" du menu "Edit" pour y copier des commandes puis les coller ailleurs, par exemple dans le panneau "Procedures". (Les combinaisons de touches habituelles pour le copier-coller sont aussi utilisables.)

Vous pouvez aussi accéder aux commandes précédentes en utilisant le menu historique (représenté par la petite flèche noire pointant vers la bas) placé à droite de la Ligne de commandes. Un clic sur ce triangle déroule un menu affichant toutes les commandes passées. Il suffit de sélectionner celle qui vous intéresse pour qu'elle ré-apparaîsse dans la ligne de commande.

**Astuce** : une manière rapide de passer d'une ancienne commande à l'autre est d'utiliser les "Flèche haut" et "Flèche bas" du clavier (mais il faut que le curseur de la souris soit dans la ligne de commande).

### Effacer

Pour effacer le contenu de la grande zone de texte (contenant les commandes précédentes et les messages du programme), cliquez le petit bouton "Clear" (nettoyer) placé dans son coin supérieur droit.

Pour effacer l'historique des commandes du menu local, sélectionnez la commande "Clear History" (effacer l'historique) de ce menu.

### Affichage du Centre de commandes

Vous pouvez cacher ou montrer le Centre de commande au moyen des commandes "Hide Command Center" (cacher le Centre de commande) et "Show Command Center" (montrer le Centre de commande) du menu "Tools" (outils).

Pour modifier la taille du Centre de commande, tirez la barre qui le sépare de l'interface du modèle. Ou cliquez l'une des deux toutes petites flèches grises situées l'extrême droite de cette barre (juste au-dessus du bouton "Clear") pour agrandir le Centre de commande au maximum, la ramener à sa taille précédente ou le cacher complètement.

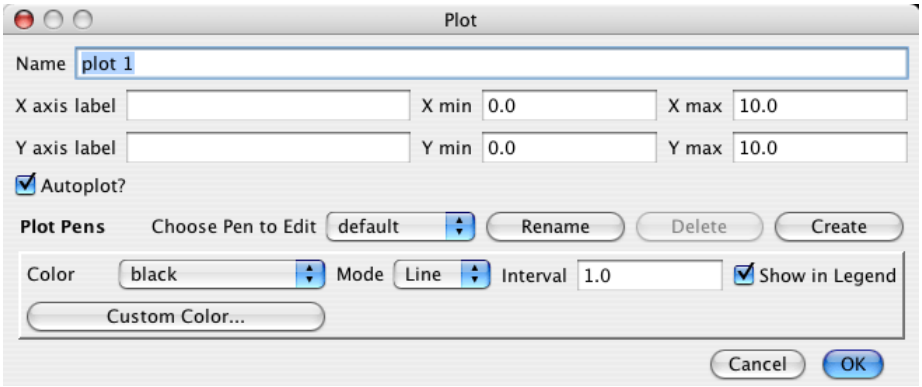
Pour faire passer le Centre de commande de la position horizontale (position par défaut) à une position verticale (ou l'inverse), cliquez sur la double flèche oblique placée à gauche du bouton "Clear".

## Les traceurs de courbes

Pour afficher ou cacher la légende des crayons du traceur, il faut ouvrir la boîte de dialogues "Plot" par un "Ctrl-clic" (Mac) ou un "clic-droit" (autres système) et sélectionner la commande "Edit..." dans le menu qui s'affiche puis cocher ou décocher le bouton "Show legend" de cette boîte.

Si vous déplacez la souris au-dessus de la zone de dessin d'un traceur, celui-ci affiche le long de ses axes les coordonnées X et Y de la position du pointeur de la souris. (Notez que la position de la souris peut ne pas correspondre exactement aux coordonnées des points du graphe. Si vous avez besoin de connaître les coordonnées exactes des points dessinés, utilisez la commande "Export..." du menu contextuel du traceur ("Ctrl-clic" ou "clic-droit") ou la commande "Plot..." du sous-menu "Export" du menu "File", puis examiner le fichier résultant dans un autre programme, un tableur par exemple.)

Quand vous créez un traceur, une boîte de dialogue d'édition de traceur "Plot" apparaît automatiquement (comme c'est le cas lors de la création de n'importe quel contrôle de l'interface).




De nombreux champs de cette boîte sont suffisamment explicites, tels que le nom ("Name") du traceur, les étiquettes des axes x ("X axis label") et y ("Y axis label"), les valeurs minimales "X min" et "Y min" et maximales "X max" et "Y max" pour ces axes.

Si la case "Autoplot?" est cochée, les échelles des axes X et Y changent automatiquement lorsque des points sont ajoutés et qu'ils se trouvent en dehors des limites fixées en cours.

Si la case "Show legend" (Montrer légende) est cochée, une légende, comprenant les noms et les couleurs des crayons dont la case à cocher "Show in Legend" est cochée, est affichée sur la droite du traceur.

Dans la zone "Plot Pens" de ce dialogue, vous pouvez créer ou personnaliser les différents crayons du traceur. Chaque traceur doit avoir au moins un crayon. Quand le traceur est créé, il possède déjà un crayon nommé "default", que vous voudrez probablement renommer pour lui donner un nom plus significatif en fonction de votre modèle.

Tous les contrôles placés en dessous du nom du crayon servent aux réglages des paramètre de ce crayon.

- Le menu "Color" (Couleur) permet de choisir l'une des couleurs de base de NetLogo et si vous n'y trouvez pas celle que vous désirez, pressez le bouton "Custom Colors" situé juste en dessous pour afficher la palette de toutes les couleurs de NetLogo. La couleur sélectionnée apparaît alors dans le bouton du menu "Color".
- Le menu "Mode" permet de sélectionner l'aspect de la courbe tracée par le crayon. Sont disponibles : "Line" (ligne), "Bar" (barre, pour des diagrammes en barres) et "Point" (les points calculés par le modèle ne sont pas reliés par des lignes comme en mode "Line").
- La valeur "Interval" est le nombre ajouté à X chaque fois que vous dessinez un point du graphe (autrement dit, elle fixe l'intervalle horizontal entre deux points du graphique).

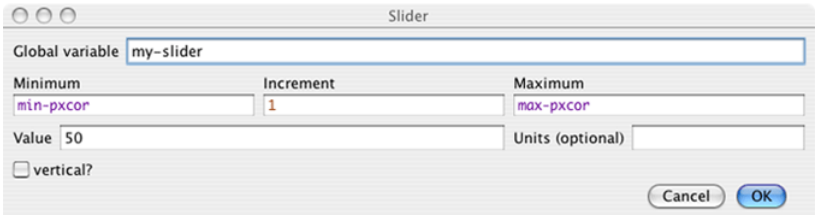
- ✔ Si la case "Show in Legend" (Montrer dans la légende) est cochée, le nom du crayon et sa couleur seront affichés dans la légende placée dans la partie supérieure droite du traceur (légende qui peut être affichée ou cachée au moyen de la case à cocher "Show legend").

Pour des informations plus détaillées sur la manière dont fonctionnent les traceurs, consultez la section Traceur du Guide de la programmation.

### Les curseurs

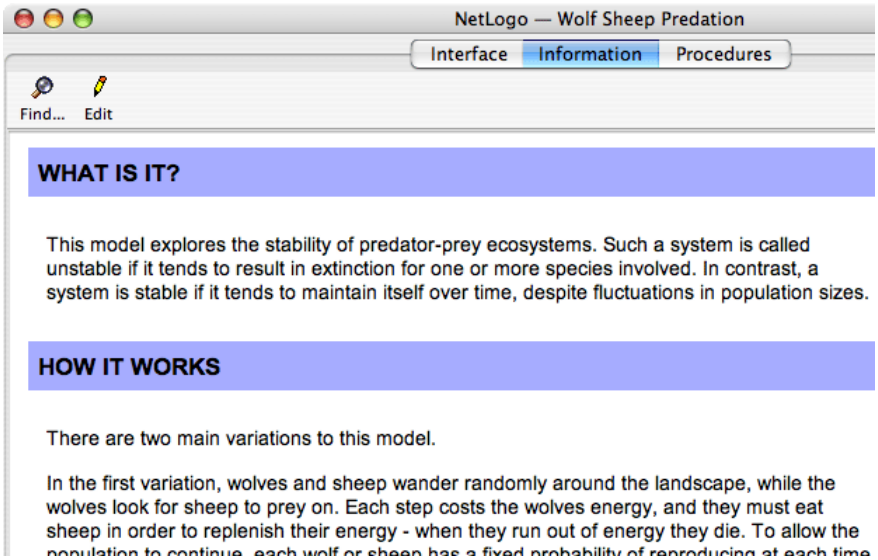
Les curseurs ("Sliders") définissent des variables globales. Ils offrent un moyen facile de changer la valeur de ces variables sans devoir modifier le code sous-jacent. Quand vous placez un curseur dans le panneau "Interface", la boîte de dialogue d'édition correspondante, "Slider", s'affiche automatiquement, comme pour tous les contrôles de l'interface. La plupart de ses champs sont suffisamment explicites. Toutefois, il est important de noter que les champs "Minimum", "Maximum" et "Increment" peuvent recevoir n'importe quelle expression reporter (expression retournant une valeur numérique) et pas uniquement des constantes. Ainsi, par exemple, vous pouvez donner à "Minimum" la valeur `min-pxcor` et à Maximum la valeur `max-pxcor` et les bornes du curseur s'ajusteront automatiquement quand vous modifierez les dimensions du monde.

**Attention** : le nom de la variable globale "Global variable" doit être écrit exactement de la même manière que dans le code.



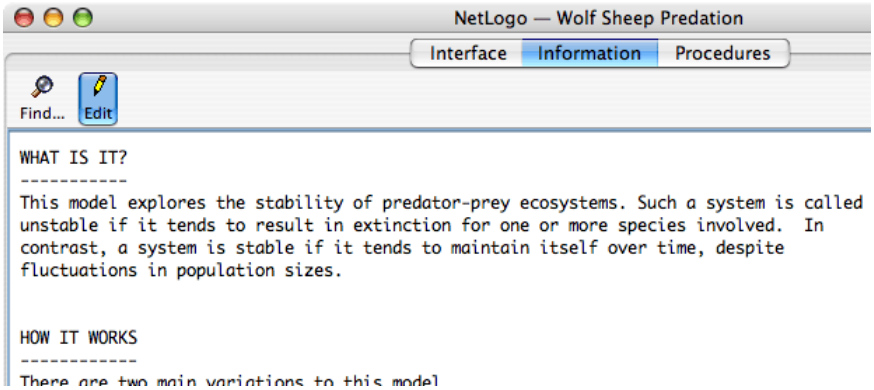

## Le panneau Information

Le panneau Information présente le modèle, explique comment l'utiliser, indique ce qu'il y a à explorer, suggère des extensions possibles et décrit les fonctionnalités NetLogo particulières utilisées dans le modèle. Ces informations sont très utiles quand on essaie le modèle la première fois (et, souvent aussi, par la suite).




Nous vous recommandons de lire le contenu du panneau Information avant de lancer la simulation du modèle. Ce panneau explique le principe (la loi, le phénomène, etc.) qui a été modélisé et comment le modèle a

été créé. Si l'aspect graphique du panneau Information n'est pas éditable, son contenu l'est. Pour éditer ce contenu, cliquez le bouton "Edit" ou double-cliquez un mot pour entrer en mode édition. Le double-clic sur un mot fait défiler le contenu jusqu'au mot cliqué et le mot est mis en évidence.




Vous pouvez éditer le texte dans ce panneau comme dans n'importe quel éditeur de texte. Toutefois, quelques options de formatage sont disponibles, formatage qui apparaîtra au moment où vous quittez le mode édition en cliquant sur le bouton "Edit" qui est alors actif.

## Formatage du texte dans le panneau Information

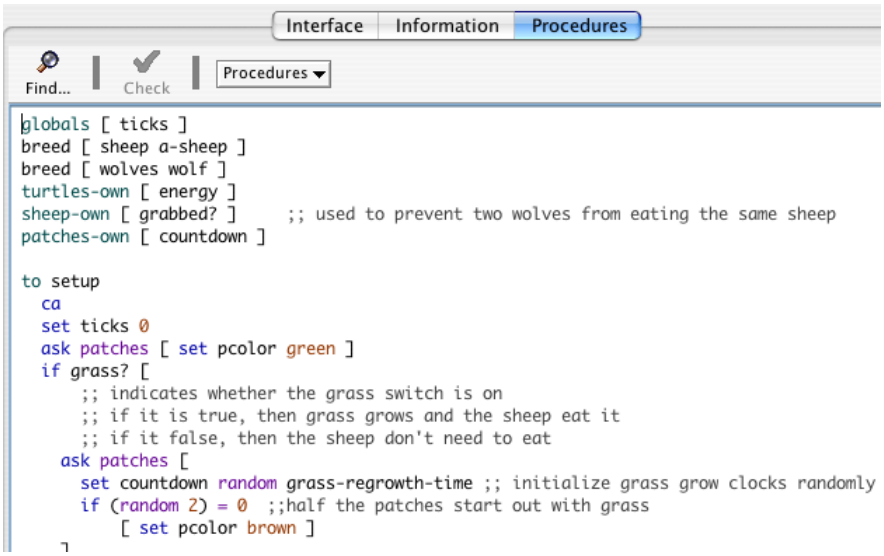
| Description | Mode édition | Mode lecture |
|---|---|---|
| Les lignes qui suivent une ligne vide et ne contiennent "que" des lettres majuscules deviennent des titres de section. | DE QUOI S'AGIT-IL ? | **DE QUOI S'AGIT-IL ?** |
| Une ligne qui ne comprend que des tirets n'est pas affichée. | ---------- | |
| Tout ce qui commence par " http:// " devient un hyperlien cliquable. | http://ccl.northwestern.edu | http://ccl.northwestern.edu |
| Les adresse E-mail deviennent aussi des liens " mailto: " cliquables. | bugs@ccl.northwestern.edu | bugs@ccl.northwestern.edu |
| Les lignes qui commencent par le caractère ligne verticale ' \| ' ("maj + backslash" '\' sur PC, "alt-7" sur Mac) deviennent du texte à chasse constante. Ce qui est utile pour des diagrammes et des formules complexes, entre autres choses. | \| c'est un texte préformaté<br>\| il y a des espaces | `c'est un texte préformaté`<br>`il y a des espaces` |

Pour quitter le mode édition et revenir à la vue normale, cliquez le bouton "Edit".

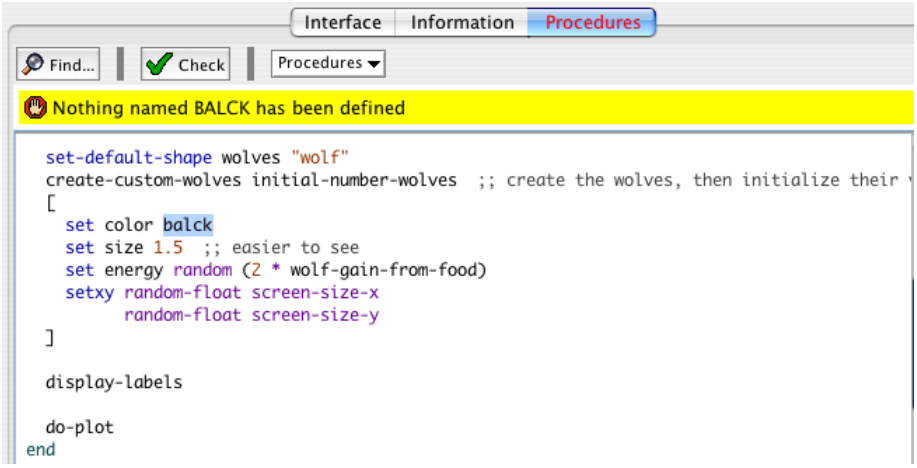
**Remarque** (pour les traducteurs) : Contrairement au code écrit dans le panneau "Procedures", le texte du panneau "Information" peut contenir des lettres accentuées et des lettres spéciales telles que le ç par exemple.

## Le panneau Procedures

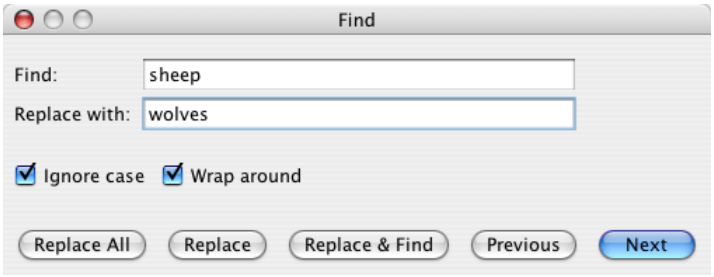
Ce panneau est l'espace de travail où le code du modèle est édité et stocké. Les commandes que vous voulez faire exécuter immédiatement sont données dans le Centre de commande, les commandes que vous voulez conserver (sauvegarder) et utiliser plus tard, encore et encore, se trouvent dans le panneau Procedures.




Pour savoir si le code contient des erreurs, pressez le bouton "Check" (Tester). S'il y a des erreurs de syntaxe, le titre du panneau Procedure devient rouge, la portion de code erronée est mise en évidence et un commentaire apparaît au-dessus de la zone de texte (sur fond jaune). Passer d'un panneau à l'autre déclenche aussi la recherche d'erreurs, la mise en évidence de l'erreur et l'affichage d'un message d'erreur, ce qui fait que si vous changer de panneau, il n'est pas nécessaire de presser d'abord le bouton "Check".




Pour trouver une fragment de code dans les procédures, cliquez de bouton "Find" (Trouver) de la barre d'outils du panneau Procedure pour afficher le dialogue "Find".




On peut entrer le mot ou la phrase à trouver (dans le champ "Find:"), ou le mot ou la phrase à trouver dans ce champ et le mot ou la phrase de remplacement (dans le champ "Replace with:"). La case à cocher "Ignore case" (Ignorer la casse) spécifie si la casse (majuscules et minuscules) doit être identique dans le texte de "Find:" et dans le texte à trouver dans le code pour que la correspondance soit valable. Si la case "Wrap around" (Enroulement) est cochée, tout le texte du panneau Procedures sera contrôlée, de la position du curseur à la fin du code, puis du début à la position du curseur, sinon la recherche ne se fait que de la position du curseur à la fin du code. Les boutons "Next" (Suivant) et "Previous" (Précédent) dirigent la recherche de la prochaine occurrence de "Find:" vers le bas ou vers le haut. "Replace" (Remplacer) remplace le texte actuellement sélectionné par le texte de remplacement et "Replace & Find" (Remplacer et trouver) remplace la phrase sélectionnée et recherche la prochaine occurrence. "Replace all" (Tout remplacer) remplace dans la zone de recherche toutes les occurrences de la phrase à trouver par la phrase de remplacement.

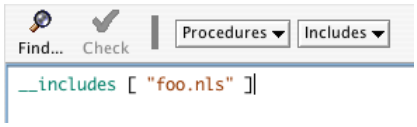
Pour trouver la définition d'une procédure particulière dans votre code, utilisez le bouton-menu "Procedures" de la barre d'outils. Ce menu liste toutes les procédures par ordre alphabétique.

Les commandes "Shift Left" (Décaler à gauche), "Shift Right" (Décaler à droite), "Comment" (Commenter) et "Uncomment" (Dé-commenter) du menu "Edit" sont utilisées dans le panneau Procedures pour, dans la section de code sélectionnée, changer le niveau d'indentation du code et pour ajouter ou enlever des points-virgules qui marquent les commentaires.

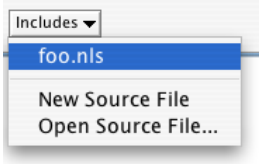
Pour plus d'informations sur la création des procédures, consultez le Tutoriel 3 : les procédures et le Guide de la programmation.

## Le menu Includes

Quand vous ajoutez le mot-clé __includes dans un modèle, un menu supplémentaire apparaît à droite du menu "Procedures". C'est le menu "Includes" qui liste tous les fichiers sources (`.nls`) NetLogo inclus dans ce fichier (soit `.nlogo`, soit `.nls`).

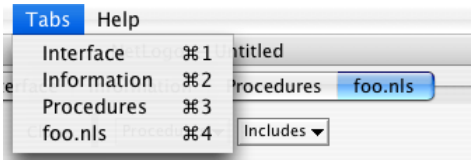



Vous pouvez cliquer sur les noms de fichiers listés dans ce menu pour ouvrir un nouveau panneau contenant le fichier sélectionné, mais vous pouvez aussi ouvrir de nouveaux fichiers ou des fichiers se trouvant dans le système à l'aide des commandes "New Source File" (Nouveau fichier source) et "Open Source File..." (Ouvrir fichier source...) offertes par ce menu.




Une fois de nouveaux panneaux ouverts, vous pouvez y accéder de la même manière que pour les autres panneaux. Ils sont tous accessibles dans le menu "Tabs" et vous pouvez aussi utiliser le clavier pour passer

d'un panneau à l'autre ("command + numéro" sur Mac, "ctrl + numéro" sur les autres systèmes).




**Attention** : Les possibilités d'inclusion sont nouvelles et expérimentales. Toutefois, nous pensons que certains utilisateurs les trouveront utiles.

# Chapitre 12

# Guide de la programmation

Le texte qui suit explique quelques importantes caractéristiques de la programmation en NetLogo.

Les modèles d'exemples de code cités au cours de ce chapitre se trouvent dans la section "Code Examples" de la Bibliothèques des modèles.

**Remarque** : tous les exemples de code cités dans ce chapitre ont été traduits et se trouvent dans le fichier `Code Examples_fr.zip` à télécharger, décompresser et installer dans le dossier "models" de votre installation NetLogo.

## Les agents

Le monde de NetLogo est constitué d'**agents**. Les agents sont des êtres capables d'exécuter des instructions. Chaque agent peut avoir sa propre activité, mais tous agissent simultanément.

NetLogo connaît quatre types d'agents : des tortues, des patches, des liens et l'observateur (observer). Les **tortues** (`"turtles"`) sont les agents qui peuvent se déplacer dans le monde. Le **monde** (`"world"`) est à deux dimensions et est divisé en une grille de patches. Chaque **patch** est un morceau de « sol » carré sur lequel les tortues peuvent se déplacer. Les **liens** (`"links"`) sont des agents qui relient deux tortues. L'**observateur** (`"observer"`) n'a pas de position déterminée – vous pouvez l'imaginer regardant d'en haut (comme un dieu) le monde des tortues et des patches.

Quand NetLogo démarre, il n'y a pas encore de tortues. L'observateur peut créer de nouvelles tortues. Les patches peuvent aussi créer de nouvelles tortues. (Les patches ne peuvent se déplacer, mais sinon, ils sont « vivants » tout comme l'est aussi l'observateur.)

Les patches sont désignés par leurs coordonnées dans le plan du monde. Le patch de coordonnées (0,0) est appelé **origine**, et les coordonnées des autres patches correspondent aux distances horizontale et verticale par rapport à ce patch. Les coordonnées d'un patch sont appelées `pxcor` et `pycor`. Tout comme pour les coordonnées du plan mathématique standard, la valeur de `pxcor` augmente quand on se déplace vers la droite et celle de `pycor` augmente quand on se déplace vers le haut.

Le nombre total de patches est déterminé par les paramètres `min-pxcor`, `max-pxcor`, `min-pycor` et `max-pycor`. Quand NetLogo démarre, `min-pxcor`, `max-pxcor`, `min-pycor` et `max-pycor` valent respectivement -16, 16, -16 et 16. Ce qui signifie que les valeurs de `pxcor` et de `pycor` peuvent aller de -16 à 16, et donc qu'il y a 33 fois 33, soit 1089 patches au total. (Vous pouvez modifier ces valeurs au moyen du bouton "Settings..." du panneau "Interface".)

Les tortues ont aussi des coordonnées : `xcor` et `ycor`. Les coordonnées d'un patch sont toujours des nombres entiers, alors que les coordonnées des tortues peuvent être des nombres décimaux. Ce qui signifie qu'une tortue peut être placée en n'importe quel point d'un patch, elle n'a pas besoin d'être toujours au milieu du patch.

Les liens n'ont pas de coordonnées, ils ont par contre deux points terminaux (chacun correspondant à une tortue). Les liens sont dessinés entre ces deux points terminaux en suivant le plus court chemin possible, même si cela signifie qu'il faut « tourner autour du monde ».

La manière dont le monde des patches se referme sur lui-même peut être modifiée. Par défaut, le monde est un tore, ce qui signifie qu'il n'a pas de limites, mais s'enroule (*wraps*) sur lui-même. Ainsi, lorsqu'une tortue traverse le bord du monde, elle disparaît puis réapparaît immédiatement au bord opposé. Sur ce tore, chaque patch possède aussi le même nombre de voisins : si vous êtes sur un patch situé sur un bord du monde, certains de vos « voisins » sont sur le bord opposé. Toutefois, vous pouvez modifier les paramètres de l'enroulement du monde au moyen des réglages de la boîte de dialogues "Model Settings" appelée par le bouton "Settings" du panneau "Interface". Si l'enroulement n'est pas autorisé dans une direction donnée, le monde est limité dans cette direction (X ou Y). Les patches placés le long de ce bord ont donc moins de huit voisins et les tortues ne peuvent pas franchir cette limite. Voir la Topologie pour plus d'informations.

## Les procédures

En NetLogo, les commandes et les reporters disent aux agents ce qu'ils doivent faire. Une **command**e est une action que l'agent doit exécuter. Un **reporter** calcule un résultat et en fait part à celui qui l'a appelé (on dit qu'il **retourne** un résultat). Les autres langages de programmation l'appelleraient une **fonction**.

La plupart des commandes commencent par un verbe, par exemple `create` (créer), `die` (mourir), `jump` (sauter), `inspect` (inspecter), `clear` (nettoyer), alors que les plupart des reporters sont des substantifs ou des expressions formées de substantifs.

Les commandes et les reporters prédéfinis sont appelés des **primitives**. Le Dictionnaire NetLogo contient et définit toutes les commandes et les reporters prédéfinis du langage NetLogo.

Les commandes et les reporters définis par l'utilisateur du langage (le programmeur de modèle) sont appelés des **procédures**. Chaque procédure commence par son nom qui doit être précédé du mot-clé `to`. Le mot-clé `end` doit marquer la fin des commandes placées dans la procédure, donc la fin de la procédure. Une fois définie, la procédure peut être utilisée n'importe où dans le programme.

Nombre de commandes et de reporters doivent recevoir des **entrées** (*inputs*), appelées **arguments**, qui contiennent des valeurs que la commande ou le reporter utilise pour accomplir ses actions (et/ou ses calculs).

**Exemples** : Voici deux procédures de type commande :

```
to setup
  clear-all        ;; effacer le monde
  crt 10           ;; créer 10 nouvelles tortues
end

to go
  ask turtles
  [ fd 1           ;; toutes les tortues avancent d'un pas
    lt random 10   ;; ... et pivotent d'un angle aléatoire
    lt random 10 ]
end
```

Notez l'utilisation des points-virgules "`;`" pour ajouter des **commentaires** au programme. Les commentaires facilitent la lecture et, surtout, la compréhension du programme.

Dans ce programme,

- ✔ `setup` et `go` sont des commandes (procédures) définies par l'utilisateur. On aurait donc pu les appeler autrement, par exemple `preparer` et `demarrer` (notez que nous n'avons pas utilisé de lettres accentuées car l'interpréteur ne les « aime » pas, il ne sait pas quoi en faire.
- ✔ `clear-all`, `crt` (pour `create turtles` = créer tortues), `ask`, `lt` (pour `left turn` = tourner à gauche) et `rt` (pour `right turn` = tourner à droite) sont des primitives de type commande.
- ✔ `random` et `turtles` sont des primitives de type reporter. `random` reçoit un nombre en entrée et retourne un nombre entier qui est plus petit que le nombre reçu (dans ce cas, entre 0 et 9). `turtles` retourne une ensemble d'agents contenant toutes les tortues. (Nous expliquerons les ensembles d'agents plus loin.)

Les procédures `setup` et `go` peuvent être appelées par d'autres procédures ou par des boutons. Bien des modèles NetLogo possèdent un bouton « une fois » (*once button*) qui appelle une procédure nommée `setup` et un bouton « pout toujours » (*forever*) qui appelle une procédure nommée `go`.

En NetLogo, vous devez spécifier quel type d'agents – turtles, patches, links ou observer – doit exécuter chaque commande. (Si vous ne spécifiez rien, le code est exécuté par l'observateur.) Dans le code ci-dessus, l'observateur utilise `ask` pour s'adresser à l'ensemble de toutes les tortues qui devront alors exécuter les commandes placées entre crochets.

`clear-all` et `crt` ne peuvent être exécutées que par l'observateur. De la même façon, `fd` ne peut être exécutée que par les tortues. Par contre, certaines autres commandes (et reporters), telles que `set`, peuvent être exécutées par différents types d'agents.

Voici quelques autres fonctionnalités avancées dont vous pouvez tirer parti quand vous définissez vos propres procédures.

### Les procédures avec arguments (ou entrées)

Les procédures que vous créez peuvent recevoir des informations au cours du déroulement du programme, exactement comme peuvent le faire les primitives. Pour créer une procédure qui puisse accepter des valeurs

en entrée (ces valeurs sont appelées des **paramètres**), il faut inclure dans la définition de la procédure une liste d'**arguments** (qui sont les noms des variables qui contiendront les valeurs que l'on veut passer à la procédure). Cette liste est à placer entre crochets à la suite du nom de la procédure. Par exemple :

```
to dessiner-polygone [nombre-cotes longueur]
  pen-down
  repeat nombre-cotes
    [ fd longueur
    rt 360 / nombre-cotes ]
end
```

Quelque part ailleurs dans le programme, vous pouvez demander aux tortues de dessiner chacune un octogone dont la longueur d'un côté est égale à sa propriété `who` (numéro reçu par la tortue à sa création) :

```
ask turtles [ dessiner-polygone 8 who ]
```

### Les procédures reporter (ou fonctions)

Tout comme vous pouvez définir vos propres commandes, vous pouvez aussi définir vos propres reporters. Mais vous avez deux choses particulières à faire. D'abord, utiliser le mot-clé `to-report` à la place de `to` pour marquer le début de la procédure. Puis, dans le corps de la procédure, utiliser le mot-clé `report` pour retourner la valeur que vous voulez exporter.

```
to-report valeur-absolue [ nombre ]
  ifelse nombre >= 0
    [ report nombre ]
    [ report (- nombre) ]
end
```

## Les variables

Les **variables** sont des endroits où ranger des valeurs (telles que des nombres). Une variable peut être une **variable globale**, une **variable tortue**, une **variable patch** ou une variable **lien**.

Si une variable est une **variable globale**, il n'existe qu'une seule valeur pour cette variable et chaque agent peut y avoir accès. Mais chaque tortue a sa propre valeur pour chaque **variable tortue** et chaque patch sa propre valeur pour chaque **variable patch**.

Un certain nombre de variables sont prédéfinies dans NetLogo. Par exemple, toutes les tortues ont une variable `color` et tous les patches ont une variable `pcolor`. (La variable patch commence par un "p" de manière à ce qu'on ne la confonde pas avec la variable tortue.) Si vous **initialisez** la variable (donc si vous lui donnez une valeur), la tortue ou le patch change de couleur. (Voir ci-dessous pour plus de détails.)

Les variables tortue prédéfinies comprennent entre autres `xcor`, `ycor` et `heading`. Les variables patch comprennent par exemple `pxcor` et `pycor`. (La liste complète des variables préprogrammées se trouve dans le Dictionnaire.)

Vous pouvez aussi définir vos propres variables.

Vous pouvez créer une **variable globale** en ajoutant un commutateur ou un curseur au modèle, ou en utilisant le mot-clé `globals` au début du code du modèle, comme ceci :

```
globals [ score ]
```

Vous pouvez également définir de nouvelles **variable tortue**, **variable patch** et **variable lien** avec les mots-clés `turtles-own` , `patches-own` et `links-own` de la manière suivante :

```
turtles-own [energie vitesse]
patches-own [frottement]
links-own [force]
```

Ces variables peuvent ensuite être utilisées librement dans le modèle. Il faut utiliser la commande `set` pour les initialiser (leur transmettre une valeur de départ). (Si vous ne le faites pas, ces variables reçoivent la valeur zéro par défaut.)

Les **variables globales** peuvent être lues et modifiées en tout temps par n'importe quel agent. De même, une tortue peut lire et modifier les **variables patch** du patch sur lequel elle se trouve. Par exemple, le code suivant

```
ask turtles [ set pcolor red ]
```

fait que chaque tortue peint en rouge le patch sur lequel elle se trouve. (C'est parce que les variables patch sont accessibles aux tortues de cette manière que vous **ne** pouvez **pas** avoir une variable patch et une variable tortue portant le même nom).

Dans d'autres situations, où vous voudriez qu'un agent lise la variable d'un autre agent, vous pouvez utiliser le mot-clé `of`. Exemple :

```
show [color] of turtle 5
;; affiche la couleur de la tortue dont le numéro who est 5
```

Le mot-clé `of` peut aussi être utilisé dans une expression plus complexe qu'un simple nom de variable, par exemple :

```
show [xcor + ycor] of turtle 5
;; affiche la somme des coordonnées X et Y
;; de la tortue dont le numéro who est 5
```

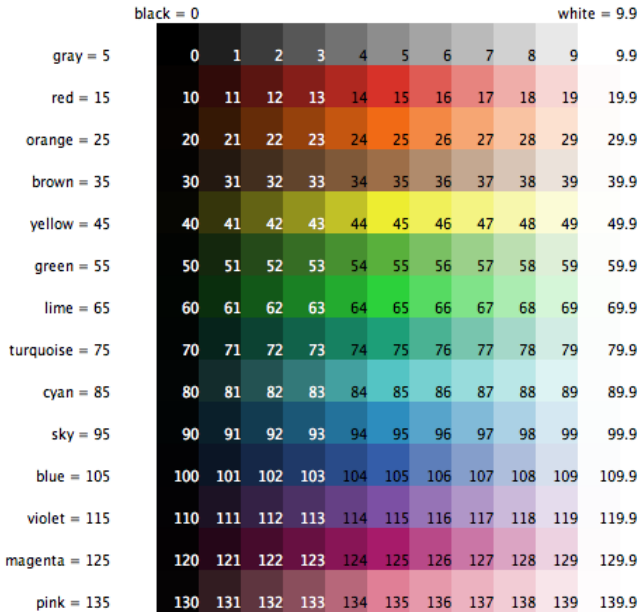
### Les variables locales

Une **variable locale** est définie et utilisée uniquement dans le contexte d'une procédure particulière ou même dans celui d'une partie seulement d'une procédure particulière. Pour créer une variable locale, il faut utiliser la commande `let`. Vous pouvez utiliser cette commande n'importe où. Si vous la mettez au début d'une procédure, elle existera dans tout le corps de cette procédure. Si vous la déclarez à l'intérieur d'une paire de crochets, par exemple dans une commande `ask`, elle n'existera alors qu'à l'intérieur de cette paire de crochets.

```
to permuter-couleurs [tortue1 tortue2]
  let temporaire [color] of tortue1
  ask tortue1 [ set color [color] of tortue2 ]
  ask tortue2 [ set color temporaire ]
end
```

# Les couleurs

Dans le langage NetLogo, les couleurs sont représentées de deux manières différentes. Premièrement en tant que nombres de 0 à 140, à l'exception de 140 lui-même. Les couleurs, avec leur numéro (que vous pouvez utiliser dans le code) sont présentées dans la figure ci-dessous :




Ce tableau montre que :

- Certaines couleurs ont un nom. (Vous pouvez utiliser ces noms dans votre code.)
- Chaque couleur portant un nom, sauf le noir et le blanc, a un numéro se terminant pas 5.
- De chaque côté de la couleur « nommée » se trouvent des teintes plus sombres et plus claires de cette couleur.
- 0 est le numéro du noir pur, 9.9 celui du blanc pur.
- Les couleurs 10, 20, etc. sont toutes si sombres qu'elles paraissent être noires.
- Les couleurs 19.9, 29.9 sont toutes si claires qu'elles paraissent être blanches.

**Exemple de code** : La palette de couleurs montrée ci-dessus a été faite en NetLogo à l'aide du modèle " Color Chart Example ".

Si vous utilisez un nombre n'appartenant pas à l'intervalle 0 à 140, NetLogo ajoute ou soustrait 140 au nombre fourni jusqu'à ce que le résultat tombe dans la fourchette désirée. Par exemple, 25 est orange, de même que 165, 305, 445, etc. sont aussi orange, il en va de même pour -115, -255, -395, etc. Ces calculs se font automatiquement chaque fois que vous donnez une valeur à la variable tortue `color` ou à la variable patch `pcolor`. Si vous deviez faire ce type de calcul dans un autre contexte, utilisez la primitive `wrap-color`.

Si vous voulez une couleur ne figurant pas dans le tableau, vos pouvez la trouver entre les nombres entiers. Par exemple, 26.5 est une teinte orange qui se trouve à mi-chemin entre l'orange 26 et l'orange 27. Mais tout ceci ne veut pas dire que vous pouvez créer n'importe quelle couleur en NetLogo. L'espace de couleurs de NetLogo n'est qu'un sous-ensemble de toutes les couleurs possibles. Il ne comporte qu'un ensemble déterminé de teintes discrètes (une teinte par ligne du tableau). En commençant par l'une de ces teintes, vous pouvez soit diminuer sa luminance (l'assombrir) soit diminuer sa saturation (l'éclaircir), mais vous ne pouvez pas diminuer à la fois la luminance et la saturation. Il s'ensuit que les valeurs des couleurs sont arrondies vers le bas à 0.1 près. Il n'y a, par exemple, pas de différences visibles entre 26.5 et 26.52 ou 26.58.

### Les primitives pour les couleurs

NetLogo possède quelques primitives utiles pour travailler avec les couleurs.

- ✔ Nous avons déjà mentionné la primitive `wrap-color`.
- ✔ La primitive `scale-color` est utile pour convertir des données numériques en couleurs.
- ✔ La primitive `shade-of?` nous indique si deux couleurs sont des nuances de la même teinte de base. Par exemple, `shade-of? orange 27` est vrai, car 27 est une nuance plus claire de la teinte orange.

**Exemple de code** : " Scale-color Example " montre l'utilisation du reporter `scale-color`.

### Les couleurs RGB

La seconde représentation des couleurs en NetLogo est une liste RGB (red/green/blue). L'utilisation du système des couleurs RGB donne accès à toute les couleurs. Les listes RGB sont formées de trois nombres entiers allant de 0 à 255. Si un nombre sort de cet intervalle, 255 en est soustrait jusqu'à ce qu'il puisse y entrer. Vous pouvez passer une liste RGB à n'importe quelle variable couleur de NetLogo (`color` pour les tortues et les liens et `pcolor` pour les patches) et l'agent sera affiché avec la couleur désirée. Ainsi, pour obtenir un patch d'un beau rouge pur, vous pouvez utiliser le code suivant :

```
set pcolor [255 0 0]
```

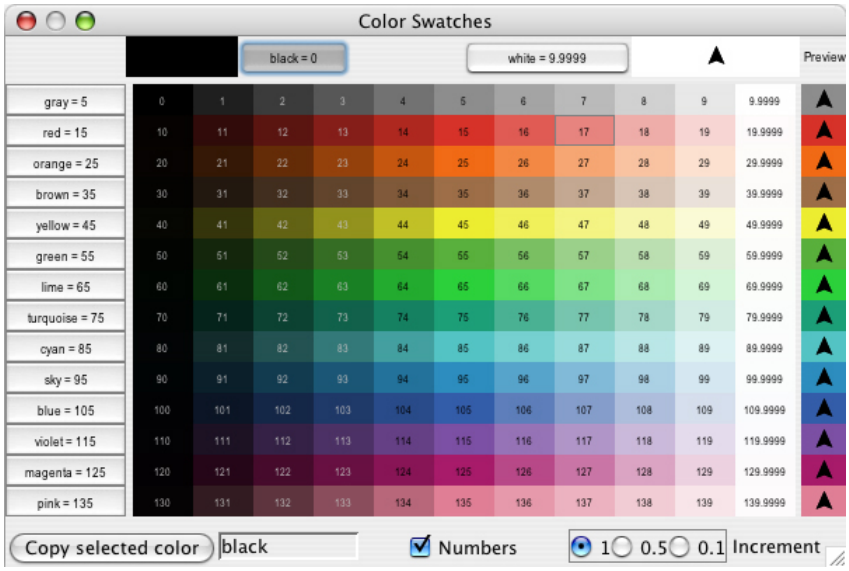
Vous pouvez convertir les couleurs des systèmes de couleurs RGB, HSB (hue/saturation/brightness) et NetLogo en utilisant `approximate-hsb` et `approximate-rgb` pour passer des couleurs HSB et RGB aux couleurs NetLogo, et `extract-hsb` et `extract-rgb` pour les convertir dans l'autre sens. Vous pouvez utiliser `rgb` pour générer des listes RGB et `hsb` pour convertir une liste HSB en liste RGB.

Puisque de nombreuses couleurs manquent dans l'espace de couleurs NetLogo, `approximate-hsb` et `approximate-rgb` ne peuvent pas toujours donner la couleur exacte demandée, mais ils tentent d'en arriver le plus près possible.

**Exemple de code** : Le modèle " HSB and RGB Example " permet de se familiariser avec les systèmes de couleurs HSB et RGB.

### La boîte de dialogue "Color Swatches"

La boîte de dialogue "Color Swatches" permet de tester et de choisir les couleurs. Vous pouvez l'ouvrir en sélectionnant "Color Swatches" dans le menu "Tools".



Quand vous cliquez sur un échantillon de couleur (ou sur un bouton de couleur), cette couleur est montrée (sous forme de pointe de flèche) au-dessus des différentes couleurs de base dans la colonne "Preview", sur un fond noir dans le coin supérieur gauche et sur un fond blanc dans le coin supérieur droit. Le nom (code) de la couleur sélectionnée (exemple `red+2`) est affiché dans le bas de la fenêtre, juste à droite du bouton "Copy selected color" qui vous permet de copier ce code pour le coller ensuite dans votre code. En bas à droite se trouvent trois options d'incrémentation : 1, 0.5 et 0.1. Ces nombres indiquent l'incrément utilisé pour l'affichage de la palette entre deux échantillons de couleurs successifs. Quand l'incrément vaut 1, il y a 10 nuances différentes dans chaque rangée, quand l'incrément vaut 0.1, il y a 100 nuances différentes dans chaque rangée. 0.5 est un réglage intermédiaire. La case à cocher "Numbers" active/désactive l'affichage des numéros de couleur NetLogo dans chaque échantillon de couleur. Les deux boutons placés dans le haut et les quatorze boutons placés à gauche de la fenêtre permettent de sélectionner directement l'une des seize couleurs de base. Enfin, si les échantillons vous paraissent trop petits (surtout avec des incrément de 0.5 ou 0.1) vous pouvez agrandir cette fenêtre à l'aide des boutons prévu à cet effet par le système sur lequel tourne NetLogo.

## La commande `ask`

NetLogo utilise la commande `ask` pour transmettre les ordres aux tortues, aux patches et aux liens. Tout le code devant être exécuté par des tortues **doit** être placé dans un « contexte » tortue. Vous pouvez établir un contexte tortue de l'une des trois manières suivantes :

- ✔ Dans un bouton en sélectionnant "Turtles" dans le menu "Agent(s)" de la boîte de dialogue "Button" qui permet d'éditer les propriétés du bouton. Tout code placé dans la zone "Commands" de ce bouton sera alors exécuté par toutes les tortues.
- ✔ Dans le Centre de commande en sélectionnant `Turtles>` dans le menu placé à gauche de la Ligne de commandes. Toute commande que vous entrerez dans cette ligne sera exécutée par toutes les tortues.
- ✔ En utilisant `ask turtles` dans les procédures, dans la Ligne de commandes ou dans les boutons.

Il en va de même pour les patches, les liens et l'observateur, sauf que vous ne pouvez pas utiliser `ask` avec l'observateur. Tout code qui ne se trouve pas dans des crochets `ask` est par défaut du code pour l'observateur.

Voici un exemple d'utilisation de `ask` dans une procédure NetLogo :

```
to setup
  clear-all
  crt 100                       ;; créer 100 tortues
  ask turtles
    [ set color red             ;; les colorer en rouge
      rt random-float 360       ;; leur donner une orientation aléatoire
      fd 50 ]                   ;; les disperser
   ask patches
   [ if pxcor > 0               ;; colorer les patches de la moitié
     [ set pcolor green ] ]     ;; droite de la Vue en vert
end
```

Les modèles de la Bibliothèque des modèles sont pleins d'autres exemples. La section "Code Examples" est un bon endroit pour commencer.

En général, l'observateur utilise la commande `ask` pour demander à toutes les tortues, tous les patches ou tous les liens d'exécuter des commandes. Mais vous pouvez aussi utiliser `ask` pour demander à une seule tortue, à un seul patch ou à un seul lien d'exécuter des commandes. Les reporters `turtle`, `patch`, `link` et `patch-at` sont utiles pour cette technique. Par exemple :

```
to setup
  clear-all
  crt 3                           ;; créer 3 tortues
  ask turtle 0                    ;; demander à la première...
    [ fd 1 ]                      ;; ...d'avancer d'un pas
  ask turtle 1                    ;; demander à la deuxième...
    [ set color green ]           ;; ...de devenir verte
  ask turtle 2                    ;; demander à la troisième...
    [ rt 90 ]                     ;; ...de tourner à droite de 90 degrés
  ask patch 2 -2                  ;; demander au patch placé en (2,-2)
    [ set pcolor blue ]           ;; ...de devenir bleu
  ask turtle 0                    ;; demander à la première tortue
    [ ask patch-at 1 0            ;; ...de demander au patch placé à l'est
      [ set pcolor red ] ]        ;; ...de devenir rouge
  ask turtle 0                    ;; demander à la première tortue...
    [ create-link-with turtle 1 ] ;; ...de créer un lien avec la seconde
  ask link 0 1                    ;; demander au lien entre les tortues 0 et 1
    [ set color blue ]            ;; ...de se colorer en bleu
end
```

Chaque tortue créée possède un **numéro d'identification** (appelé "who number" en NetLogo). La première tortue créée a le numéro 0, la deuxième le numéro 1, et ainsi de suite. La primitive reporter `turtle` demande un numéro d'identification en entrée et retourne la tortue qui possède ce numéro. La primitive reporter `patch` demande en entrée les valeurs pour `pxcor` et `pycor` et retourne le patch qui possède ces coordonnées. La primitive reporter `link` demande deux entrées qui sont les numéros d'identification des deux tortues reliées par un lien. Et la primitive reporter `patch-at` demande des décalages (*offsets*) : les distances, dans les directions X et Y, à partir du premier agent. Dans l'exemple ci-dessus, on demande à la tortue dont le numéro d'identification est 0 de s'adresser au patch placé à l'est (et non aux patches placés au nord) d'elle-même.

Vous pouvez aussi sélectionner un sous-ensemble de tortues, un sous-ensemble de patches ou un sous-ensemble de liens puis leur demander de faire quelque chose. Mais cette manière de faire implique un nouveau concept appelé **ensembles d'agents** (*agentsets*). La section suivante expliquera ce concept en détails.

Quand vous demandez à un ensemble d'agents d'exécuter plus d'une seule commande, chaque agent doit terminer sa tâche avant que l'agent suivant puisse exécuter la sienne. Un agent exécute toutes les commandes, puis l'agent suivant les exécute toutes à son tour, et ainsi de suite. Par exemple, si vous écrivez :

```
ask turtles
[
  fd 1
  set color red
]
```

d'abord une tortue se déplace et devient rouge, puis une autre tortue se déplace et devient rouge, etc.

Mais si vous écrivez ce qui suit :

```
ask turtles [ fd 1 ]
ask turtles [ set color red ]
```

d'abord toutes les tortues se déplacent. Et une fois qu'elles se sont toutes déplacées, elles deviennent rouges.

(La commande `ask` possède une autre forme obéissant à une autre règle de priorité. Voir Ask-Concurrent ci-dessous.)

## Les ensembles d'agents (agentsets)

Un **ensemble d'agents** (*agentset*) peut contenir soit des tortues, soit des patches, soit des liens, donc jamais plus d'un type d'agents à la fois.

Les éléments d'un ensemble d'agents ne sont pas dans un ordre particulier. En réalité, ils sont toujours dans un ordre aléatoire. Et chaque fois que vous utilisez cet ensemble, ses agents sont dans un ordre aléatoire ***différent***. Cette manière de traiter les ensembles d'agents vous aide à faire en sorte que le modèle ne traite pas une tortue, un patch ou un lien particulier différemment des tous les autres (à moins que vous ne le vouliez). Puisque l'ordre est à chaque fois aléatoire, aucun agent ne passe toujours en premier.

Nous avons déjà rencontré la primitive `turtles` qui retourne l'ensemble d'agents contenant toutes les tortues, la primitive `patches` qui retourne l'ensemble d'agents contenant tous les patches et la primitive `links` qui retourne l'ensemble d'agents contenant tous les liens.

Mais ce qui est particulièrement puissant dans le concept d'ensemble d'agents est que vous pouvez construire des ensembles d'agents qui ne contiennent que *certaines* tortues, que *certains* patches ou que *certains* liens. Par exemple, toutes les tortues rouges, tous les patches dont la coordonnée `pxcor` est divisible par cinq ou toutes les tortues situées dans le premier quadrant et qui sont placées sur un patch vert ou qui ont des liens les reliant à la tortue 0. Ces ensembles d'agents peuvent ensuite être utilisés par `ask` ou par divers reporters qui acceptent des ensembles d'agents en entrée.

Une manière de créer un ensemble d'agents ne contenant que les tortues placées sur le patch courant est d'utiliser le reporter `turtles-here`, ou d'utiliser `turtles-at` pour créer un ensemble d'agents contenant les tortues placées sur le patch situé à x patches horizontalement et à y patches verticalement du patch courant. Il existe également la primitive `turtles-on` pour obtenir l'ensemble des tortues placées sur un patch particulier ou sur un ensemble de patches, ou l'ensemble des tortues placées sur le même patch qu'une tortue donnée ou qu'un ensemble de tortues.

Voici encore quelques exemples supplémentaires montrant comment créer des ensembles d'agents :

```
;; toutes les autres tortues:
other turtles
;; toutes les autres tortues sur ce patch:
other turtles-here
;; toutes les tortues rouges:
turtles with [color = red]
;; toutes les tortues rouges sur mon patch:
turtles-here with [color = red]
;; les patches de la moitié droite de la vue:
patches with [pxcor > 0]
;; toutes les tortues situées à moins de 3 patches:
turtles in-radius 3
;; les quatre patches placés à l'est, au nord, à l'ouest et au sud:
patches at-points [[1 0] [0 1] [-1 0] [0 -1]]
;; abréviation pour ces quatre patches:
neighbors4
;; les tortues du premier quadrant qui sont sur des patches verts:
turtles with [(xcor > 0) and (ycor > 0)and (pcolor = green)]
;; les tortues situées sur mes quatre patches voisins:
turtles-on neighbors4
;; tous les liens connectés à la tortue 0
[my-links] of turtle 0
```

Notez l'utilisation de `other` pour exclure de l'ensemble l'agent appelant. C'est une pratique courante.

Une fois que vous avez créé un ensemble d'agents, voici quelques tâches que vous pouvez accomplir :

- ✔ Utiliser `ask` pour demander aux agents de l'ensemble de faire quelque chose.
- ✔ Utiliser `any?` pour tester si l'ensemble d'agents est vide.

- ✔ Utiliser `all?` pour voir si tous les agents de l'ensemble satisfont à une condition.
- ✔ Utiliser `count` pour savoir exactement combien il y a d'agents dans l'ensemble.

Et voici encore quelques actions plus complexes que vous pouvez entreprendre :

- ✔ Prendre un agent au hasard dans l'ensemble avec `one-of`. Par exemple, nous pouvons faire qu'une tortue choisie aléatoirement deviennent verte :

```
ask one-of turtles [ set color green ]
```

- ✔ Ou demander à un patch choisi au hasard de donner naissance à une nouvelle tortue avec `sprout` :

```
ask one-of patches [ sprout 1 ]
```

- ✔ Utiliser les reporters `max-one-of` ou `min-one-of` pour trouver quel agent de l'ensemble est « le plus » ou « le moins » sur une certaine échelle. Par exemple, pour éliminer la tortue la plus riche, vous pourriez écrire :

```
ask max-one-of turtles [sum assets] [ die ]
```

- ✔ Dessiner un histogramme de l'ensemble d'agents en utilisant la commande `histogram` (en combinaison avec `of`).
- ✔ Utiliser `of` pour créer une liste de valeurs, une pour chaque agent de l'ensemble. Puis utiliser l'une des primitives de traitement de listes de NetLogo pour faire quelque chose avec cette liste. (Voir la section Les listes ci-dessous). Par exemple, pour trouver quelle est la fortune moyenne des tortues, vous pouvez écrire :

```
show mean [sum assets] of turtles
```

- ✔ Utiliser les reporters `turtle-set`, `patch-set` et `link-set` pour créer de nouveaux ensembles d'agents en rassemblant des agents provenant de différentes sources.
- ✔ Tester si deux ensembles d'agents sont égaux en utilisant `=` ou `!=`.
- ✔ Utiliser `member?` pour voir si un agent particulier est membre d'un ensemble d'agents.

Ces exemples ne font qu'égratigner la surface. Fouillez dans la Bibliothèque des modèles pour y trouver de nombreux autres exemples et consultez le Dictionnaire NetLogo pour davantage d'informations concernant toutes les primitives d'ensembles d'agents.

D'autres exemples d'utilisation d'ensembles d'agents sont donnés dans les diverses entrées de ces primitives dans le Dictionnaire NetLogo. Au cours de votre apprentissage de la programmation en NetLogo, il est important de commencer de penser aux combinaisons de commandes dans le sens de : comment chaque élément passe des informations à l'élément suivant. Les ensembles d'agents sont une part importante de ce schéma conceptuel et offrent au développeur NetLogo puissance et souplesse, tout en étant plus proche du langage naturel.

**Exemple de code** : " Ask Ordering Example "

Plus haut, nous avions dit que les agents des ensembles d'agents sont toujours dans un ordre aléatoire, ordre aléatoire qui change chaque fois que l'ensemble est utilisé. Si vous avez besoin que vos agents fassent quelque chose dans un ordre bien déterminé, il faut utiliser une liste d'agents à la place d'un ensemble d'agents. Voir la section Les listes ci-dessous.

## Les races (breeds)

NetLogo permet de définir différentes **races** (*breeds*) de tortues et races de liens. Une fois ces races définies, vous pouvez aller plus loin et donner à ces différentes races des comportements différents. Par exemple, vous pouvez avoir une race appelée `moutons` et une race appelée `loups` et faire que les loups essaient de manger les moutons ou avoir des races appelées `routes` et `trottoirs` puis faire en sorte que les piétons marchent sur les trottoirs et les véhicules circulent sur les routes.

Vous définissez les races de tortues au moyen du mot-clé `breed`. Les définitions des races doivent se trouver au début du code du modèle, avant la définition des procédures.

```
breed [loups loup]
breed [moutons mouton]
```

Notez que, dans les deux définitions ci-dessus, les noms des races sont d'abord écrites au pluriel puis au singulier. Cette syntaxe est à respecter et si le nom au pluriel ne diffère pas du nom au singulier, il faut marquer la différence, par exemple :

```
breed [ souris une-souris ]
```

Vous pouvez faire référence à un membre d'une race en utilisant le nom de la race au singulier, tout comme on le fait avec le reporter `turtle`. De plus, lorsque le nom des membres de la race est affiché, ce nom est aussi au singulier.

Certaines commandes et reporters ont dans leur nom le nom de la race au pluriel, comme `create-<breeds>`. D'autres ont dans leur nom le nom de la race au singulier, comme dans `<breed>`

L'ordre dans lequel les races sont déclarées est aussi l'ordre dans lequel elles sont « empilées » dans la vue. Ainsi, les races définies plus bas apparaissent au-dessus des races définies avant elles. Dans l'exemple ci-dessus, les `moutons` sont dessinés par-dessus les `loups`.

Quand vous définissez une race telle que `moutons`, un ensemble d'agents de cette race est automatiquement créé, ce qui fait que toutes les fonctionnalités liées aux ensembles d'agents sont immédiatement disponibles pour l'ensemble d'agents `moutons`.

Les primitives suivantes sont aussi automatiquement créées et disponibles une fois qu'un ensemble d'agents (ici `moutons`) a été défini : `create-moutons`, `hatch-moutons`, `sprout-moutons`, `moutons-here`, `moutons-at`, `moutons-on` et `is-a-moutons?`.

De plus, vous pouvez utiliser `moutons-own` pour définir de nouvelles variables tortue que seules les tortues appartenant à cet ensemble `moutons` peuvent avoir.

Un ensemble d'agents de la race de tortues est stocké dans la variable tortue `breed`. Vous pouvez ainsi tester cette race de tortues comme suit :

```
if breed = moutons [ ... ]
```

Notez aussi que les tortues peuvent changer de race. Un loup n'est pas obligé de rester un loup tout au long de sa vie. Transformons un loup choisi au hasard en un mouton :

```
ask one-of loups [ set breed moutons ]
```

La primitive `set-default-shape` est utile pour associer certaines formes de tortues à certaines races. Voir la section concernant les formes plus loin.

Voici un petit exemple d'utilisation des races :

```
breed [souris une-souris]
breed [grenouilles grenouille]
souris-own [fromage]
to setup
  clear-all
  create-souris 50
    [ set color white
    set fromage random 10 ]
  create-grenouilles 50
    [ set color green ]
end
```

**Exemple de code** : " Breeds and Shapes Example "

### Les races de liens (*Link Breeds*)

Les races de liens ressemblent beaucoup aux races de tortues, mais il y a quelques différences :

Quand vous déclarez une race de liens, vous devez précisez s'il s'agit d'une race de liens orientés ou non-orientés en utilisant les mots-clés `directed-link-breed` et `undirected-link-breed`.

```
directed-link-breed [ routes route]
undirected-link-breed [amis ami ]
```

Une fois que vous avez créé un lien de race, vous ne pouvez pas créer des liens sans race, et vice-versa. (Toutefois, vous pouvez avoir dans le même monde des liens orientés et des liens non-orientés, mais il ne doivent pas être de la même race.)

Contrairement à ce qui se passe avec les races de tortues, les races de liens doivent avoir un nom au singulier, car nombre de commandes et de reporters de liens demandent un nom au singulier, comme `<link-breed>-neighbor?`.

Les primitives suivantes sont aussi automatiquement disponibles une fois qu'une race de liens orientés (ici `routes`) a été définie : `create-route-from`, `create-routes-from`, `create-route-to`, `create-routes-to`, `in-route-neighbor?`, `in-route-neighbors`, `in-route-from`, `my-in-routes`, `my-out-routes`, `out-route-neighbor?`, `out-route-neighbors` et `out-route-to`.

Et les primitives suivantes sont aussi automatiquement disponibles une fois qu'une race de liens non-orientés (ici `amis`) a été définie : `create-ami-with`, `create-amis-with`, `ami-neighbor?`, `ami-neighbors`, `ami-with`, `my-amis`.

Tout comme avec les races de tortues, l'ordre dans lequel les races de liens ont été déclarées définit l'ordre dans lequel ces liens sont dessinés. Ainsi, les `amis` seront toujours au-dessus des `routes` (si pour une raison quelconque ces deux races se trouvent dans le même modèle). Vous pouvez aussi utiliser `<link-breeds>-own` pour déclarer séparément des variables pour chaque race de liens.

Vous pouvez changer la race des liens comme pour les tortues, mais vous ne pouvez pas transformer des liens de race en liens sans race, ceci dans le but d'empêcher d'avoir des liens de race et des liens sans races dans le même monde (ce qui est impossible).

```
ask one-of amis [ set breed routes ]
ask one-of amis [ set breed links ] ;; génère une erreur d'exécution
```

La primitive `set-default-shape` peut aussi être utilisée avec des races de liens pour les associer à une forme de lien particulière.

**Exemple de code** : " Link Breeds Example "

# Les boutons

Les boutons placés dans le panneau "Interface" offrent un moyen commode de contrôler le modèle. En général, un modèle aura au moins un bouton "setup" (initialiser), pour mettre en place l'état initial du monde, et un bouton "go" pour faire démarrer la simulation. Certains modèles auront d'autres boutons pour accomplir d'autres actions.

Chaque bouton contient au moins un peu de code NetLogo (sinon il ne sert à rien, si ce n'est de faire joli). Ce code est exécuté quand le bouton est pressé.

Un bouton peut être soit un bouton « une fois » (*once button*), soit un bouton « pour-toujours » (*forever button*). Vous spécifiez le type désiré en éditant le bouton et en cochant ou décochant la case "Forever". Les boutons une-fois exécutent leur code une seule fois, puis stoppent et reviennent en position déclenché. Les boutons pour-toujours exécutent leur code encore et encore, jusqu'à ce que, soit le code rencontre la commande `stop`, soit l'utilisateur presse encore une fois ce bouton pour stopper la simulation. Si vous déclenchez ce bouton pour stopper la simulation, l'exécution du code n'est pas interrompue immédiatement. Le bouton attend jusqu'à ce que le code du cycle en cours ait été exécuté, puis il revient en position déclenché.

Normalement, un bouton porte le nom du code qu'il exécute. Par exemple, un bouton qui affiche "go" contient généralement le code `go`, ce qui signifie « exécute la procédure `go` ». (Les procédures sont définies dans le panneau "Procedures", voir ci-dessous.) Mais vous pouvez aussi éditer un bouton et spécifier un nom d'affichage (dans le champ "Display name" de la fenêtre d'édition "Button"), nom qui apparaîtra sur le bouton à la place de son code. Vous pouvez utiliser cette fonctionnalité si vous pensez que le nom du code n'est pas assez explicite pour l'utilisateur (surtout si vous traduisez l'interface d'un modèle).

Quand vous écrivez du code dans un bouton, vous devez aussi spécifier quels agents devront exécuter ce code. Comme exécuteur du code, vous pouvez choisir, dans le menu "Agent(s)", l'observateur ("Observer"), toutes les tortues ("Turtles"), tous patches ("Patches") ou tous les liens ("Links"). Si vous voulez que le code ne soit exécuté que par certaines tortues ou par certains liens, vous pouvez créer un bouton observateur et faire que l'observateur utilise la commande `ask` pour demander à ces tortues ou à ces patches particuliers de faire quelque chose.

Quand vous éditez un bouton, vous avez la possibilité de lui assigner une « touche action ». Cette assignation a pour résultat qu'une pression sur cette touche du clavier produit le même effet qu'un clic sur le bouton — attention, il faut que le focus soit dans la zone de l'interface contenant la Vue, sinon les frappes au clavier vont dans la Ligne de commandes (il suffit de cliquez dans cette zone avec la souris pour y placer le focus). Si le bouton est un bouton pour-toujours, il restera enfoncé jusqu'à ce que l'on presse encore une fois sur la touche (ou que l'on clique le bouton). Les touches actions sont particulièrement utiles pour les jeux ou les modèles dans lesquels les actions sur les boutons doivent être rapides.

### Les boutons attendent leur tour

Il est possible d'avoir plus d'un bouton pressé à la fois. Si cette situation se présente, les boutons « attendent leur tour », ce qui signifie que le code d'un seul bouton est exécuté à la fois. Chaque bouton exécute son code une fois jusqu'au bout pendant que les autres attendent, puis le bouton suivant prend son tour.

Dans les exemples suivants, "setup" est un bouton une-fois et "go" est un bouton pour-toujours.

**Exemple 1** : L'utilisateur presse "setup" puis "go" tout de suite après, sans attendre que le bouton "setup" revienne à sa position déclenché. Résultat : "setup" termine son travail avant que "go" ne prenne la relève.

**Exemple 2** : Alors que "go" est encore enclenché, l'utilisateur presse "setup" pour réinitialiser la simulation. Résultat : "go" termine le cycle en cours, le bouton "setup" exécute alors son code et enfin le bouton "go" reprend son travail en démarrant une nouvelle simulation avec les paramètres spécifiés dans "setup".

**Exemple 3** : L'utilisateur a deux boutons pour-toujours enfoncés en même temps. Résultat : un bouton exécute tout son code une fois puis l'autre bouton exécute tout son code une fois, et ainsi de suite, alternativement.

Notez que si un bouton est « planté » dans une boucle infinie, aucun des autres boutons ne pourra exécuter son code.

### Boutons tortue, patch et lien « pour-toujours »

Il y a une différence subtile entre placer des commandes dans un bouton « pour-toujours » de type tortue, patch ou lien et placer les mêmes commandes dans un bouton observateur qui exécute `ask turtles`, `ask patches` ou `ask links`. Une commande `ask` ne se termine pas tant que tous ses agents n'ont pas fini d'exécuter toutes les commandes données par `ask`. Il en résulte que les agents, puisqu'ils exécutent leurs commandes indépendamment et en concurrence les uns avec les autres, peuvent ne plus être synchronisés en cours de cycle, mais ils se re-synchronisent à la fin du cycle, une fois toutes leurs tâches accomplies. Il n'en va pas de même pour les boutons « pour-toujours» de type tortue, patch ou lien, car dans ce cas, puisque `ask` n'est pas utilisé, chaque tortue, patch ou lien exécute son code encore et encore sans s'occuper des autres. Il en résulte qu'ils ne se synchronisent plus et restent désynchronisés jusqu'à la fin de la simulation.

Pour le moment, cette fonctionnalité n'est que très rarement utilisée dans les modèle de la Bibliothèque des modèles. Un des rares qui l'utilise est le modèle "Termites" de la section "Biology" des "Sample Models". Dans ce modèle, le bouton "go" est un bouton tortue pour-toujours, ce qui fait que chaque termite exécute son code indépendamment des autres termites, et l'observateur n'est pas du tout impliqué dans le déroulement de la simulation. Ce qui signifie que si, par exemple, vous vouliez ajouter une traceur au modèle, vous devriez ajouter un second bouton pour-toujours (un bouton observateur pour-toujours), et démarrer les deux boutons pour-toujours en même temps.

Dans sa version actuelle, NetLogo n'offre pas la possibilité, pour un bouton pour-toujours, de démarrer un autre bouton. Les boutons ne sont activés que lorsque vous les cliquez (ou pressez les touches qui leur sont attribuées.)

## Les listes

Dans les modèles les plus simples, chaque variable ne contient qu'un « morceau » d'information, en général un nombre ou une chaîne. La fonctionnalité **liste** permet de stocker plusieurs morceaux d'information dans une seule variable en rassemblant ces morceaux d'information dans une liste. Chaque valeur de la liste peut être de n'importe quel type : nombre, chaîne, agent, ensemble d'agents, ou même une autre liste.

En NetLogo, les listes sont un moyen efficace de rassembler de l'information. Si vos agents doivent exécuter des calculs répétitifs sur de multiples variables, il est souvent plus simple d'avoir une seule variable de type liste qu'un grand nombres de variables numériques distinctes. Plusieurs primitives simplifient le déroulement des mêmes opérations sur chaque valeur d'une liste.

Le Dictionnaire NetLogo comporte toute une section consacrée aux primitives traitant des listes.

### Listes constantes

Vous pouvez construire une liste simplement en mettant entre crochets les valeurs devant faire partie de la liste, comme ceci : `set maliste [ 2 4 6 8 ]`. Notez que les éléments de la liste sont séparés par des espaces. Vous pouvez construire des listes de nombres ou des listes de chaînes de la même manière, et même des listes de listes, comme ici : `[ [ 2 4 ] [ 3 5 ] ]`.

Une **liste vide** est construite en ne mettant rien entre les crochets, comme ceci : `[]`.

### Construire des listes « à la volée »

Si vous voulez construire une liste dont les valeurs sont déterminées par des reporters plutôt que d'être une simple collection de constantes, utilisez le reporter `list`. Ce reporter accepte deux autres reporters, les active et retourne leurs résultats sous forme de liste.

Ainsi, si je désire une liste contenant deux valeurs aléatoires, je peux utiliser le code suivant :

```
set liste-aleatoire list (random 10) (random 20)
```

Ce morceau de code met dans la variable `liste-aleatoire` une nouvelle liste constituée de deux nombres aléatoires entiers chaque fois qu'il est exécuté.

Pour construire des liste plus longues ou plus courtes, vous pouvez utiliser le reporter `list` avec plus de deux ou moins de deux entrées, mais dans les deux cas, il faut mettre tout l'appel entre parenthèses, par exemple :

```
(list random 10)
(list random 10 random 20 random 30)
```

Pour plus d'informations, voir la section Nombre d'arguments variable.

Certains types de listes sont construits plus facilement en utilisant le reporter `n-values` qui permet de construire une liste d'une longueur spécifique en appelant un reporter donné de manière répétée. Vous pouvez construire une liste contenant plusieurs fois la même valeur, tous les nombres appartenant à un certain intervalle, un ensemble de nombres aléatoires, ou encore bien d'autres choses. Voir les différentes entrées y-relatives du dictionnaire pour plus de détails et d'exemples.

La primitive `of` permet de construire une liste à partir d'un ensemble d'agents. Elle retourne une liste contenant la valeur de chaque agent pour un reporter donné. (Le reporter peut être un simple nom de variable ou une expression plus complexe – voire même un appel de procédure définie en utilisant la primitive `to-report`.) Une construction fréquente est :

```
max [...] of turtles
sum [...] of turtles
```

et ainsi de suite.

Vous pouvez combiner deux ou plusieurs listes avec le reporter `sentence` , qui concatène des listes en combinant leur contenu en une liste unique, plus grande. Tout comme `list`, `sentence` demande normalement deux arguments, mais peut accepter n'importe quel nombre d'arguments si l'appel est entouré de parenthèses.

### Modifier les membres d'une liste

Du point de vue technique, les listes ne peuvent pas être modifiées, mais vous pouvez construire de nouvelles listes à partir de listes plus anciennes. Si vous voulez que la nouvelle liste remplace l'ancienne, utilisez le mot-clé `set`. Par exemple :

```
set maliste [2 7 5 Bob [3 0 -2]]
; maliste est maintenant [2 7 5 Bob [3 0 -2]]
set maliste replace-item 2 maliste 10
; maliste est maintenant [2 7 10 Bob [3 0 -2]]
```

On peut remplacer un membre d'une liste avec le reporter `replace-item` qui demande trois arguments, le premier spécifiant quel membre de la liste doit être modifié. Le premier membre d'une liste porte le numéro 0, le deuxième le numéro 1, et ainsi de suite.

Pour ajouter un membre, disons 42, à la fin de la liste, utilisez le reporter `lput`. (`fput` ajoute un membre au début de la liste.)

```
set maliste lput 42 maliste
; maliste est maintenant [2 7 10 Bob [3 0 -2] 42]
```

Mais que faire si vous changez d'avis? Le reporter `but-last` (`bl` en abrégé) retourne tous les membres de la liste sauf le dernier.

```
set maliste but-last maliste
; maliste est maintenant [2 7 10 Bob [3 0 -2]]
```

Supposons que vous vouliez vous passer du membre 0, le 2 du début de la liste.

```
set maliste but-first maliste
; maliste est maintenant [7 10 Bob [3 0 -2]]
```

Supposons que vous vouliez remplacer par 9 le -2 qui est le troisième membre de la liste emboîtée en troisième position dans la liste « de base ». La solution est de se rendre compte que le nom qui peut être utilisé pour appeler la liste emboîtée `[ 3 0 -2 ]` est `item 3 maliste`. Ensuite, le reporter `replace-item` peut être emboîté pour changer la « liste-dans-une-liste ». Les parenthèses sont ajoutées pour plus de clarté.

```
set maliste (replace-item 3 maliste (replace-item 2 (item 3 maliste) 9))
; maliste est maintenant [7 10 Bob [3 0 9]]
```

### Itération sur des listes

Si vous voulez effectuer une opération successivement sur chaque membre d'une liste, la primitive commande `foreach` et le reporter `map` peuvent être utiles.

`foreach` est utilisée pour exécuter une commande ou un ensemble de commandes sur chaque membre d'une liste. Il demande en entrée une liste et un bloc de commandes, comme ici :

```
foreach [2 4 6]
  [ crt ?
    show (word "créé " ? " tortues") ]
=> créé 2 tortues
=> créé 4 tortues
=> créé 6 tortues
```

Dans le bloc de commandes, la variable `?` contient la valeur courante de la liste passée en entrée.

Voici encore quelques exemples d'utilisation de `foreach` :

```
foreach [1 2 3] [ ask turtles [ fd ? ] ]
;; les tortues avancent de 6 patches (1 + 2 + 3)
foreach [true false true true] [ ask turtles [ if ? [ fd 1 ] ] ]
;; les tortues avancent de 3 patches (1 + 0 + 1 + 1)
```

`map` est semblable à `foreach`, mais c'est un reporter. Il demande en entrée une liste et un autre reporter. Notez toutefois que, contrairement à `foreach`, le reporter doit être à la première place, comme ici :

```
show map [round ?] [1.2 2.2 2.7]
;; affiche [1 2 3]
```

`map` retourne une liste contenant les résultats de l'action du reporter sur chaque membre de la liste reçue en entrée. A nouveau, utilisez `?` pour faire référence au membre courant de la liste.

Voici un autre exemple de l'utilisation de `map` :

```
show map [? < 0] [1 -1 3 4 -2 -10]
;; affiche [false true false false true true]
```

`foreach` et `map` ne sont pas nécessairement utiles dans toutes les situations dans lesquelles vous voulez opérer sur tout une liste. Dans certains cas, vous devrez avoir recours d'autres techniques, telles que des boucles créées par `repeat` ou `while` ou une procédure récursive.

La primitive `sort-by` utilise une syntaxe ressemblant à celles de `map` et de `foreach`, sauf que, puisque le reporter doit comparer deux objets, il faut utiliser les deux variables spéciales `?1` et `?2` à la place de `?`.

Voici un exemple avec `sort-by` :

```
show sort-by [?1 < ?2] [4 1 3 2]
;; affiche [1 2 3 4]
```

### Nombre d'arguments variable

Certaines commandes et certains reporters utilisant des listes et des chaînes peuvent recevoir un nombre d'arguments variable. Dans ces cas, pour pouvoir leur passer un nombre d'arguments autre que celui par défaut, la primitive et ses arguments doivent être entourés de parenthèses. En voici quelques exemples :

```
show list 1 2
=> [1 2]
show (list 1 2 3 4)
=> [1 2 3 4]
show (list)
=> []
```

Notez que chacune de ces commandes spéciales est définie avec un nombre d'arguments par défaut pour lequel les parenthèses ne sont pas nécessaires. Les primitives offrant cette fonctionnalité sont: `list`, `word`, `sentence`, `map` et `foreach`.

### Les listes d'agents

Plus haut, nous avons dit que les éléments des ensembles d'agents sont toujours dans un ordre aléatoire, ordre qui change chaque fois que l'on fait appel à l'ensemble. Si vos agents doivent faire quelque chose dans un ordre déterminé, vous devez utiliser des listes d'agents à la place des ensembles d'agents.

Les deux primitives `sort` et `sort-by` vous aident dans cette tâche.

Les primitives `sort` et `sort-by` peuvent recevoir comme argument un ensemble d'agents. Le résultat est toujours une nouvelle liste contenant le même nombre d'agents que l'ensemble d'agents dont elle est issue, mais dans un ordre particulier qui ne changera pas.

Si vous utilisez `sort` sur un ensemble d'agents de type tortue, le résultat est une liste de tortues triée par numéro d'identification ("`who` number") croissant .

Si vous utilisez `sort` sur un ensemble d'agents de type patches, le résultat est une liste de patches triée de gauche à droite et de haut en bas.

Si vous utilisez `sort` sur un ensemble d'agents de type lien, le résultat est une liste de liens triés en ordre croissant, d'abord en fonction de `end1` puis en fonction de `end2`, tous les liens restants étant triés par race en fonction de l'ordre dans lequel ces races ont été déclarées dans le panneau "Procedures".

Si vous avez besoin d'un ordre décroissant, vous pouvez combiner la primitive `reverse` avec le reporter `sort`, par exemple `reverse sort turtles`.

Si les agents doivent être triés en fonction d'un autre critère que celui utilisé par défaut par la primitive `sort`, vous devez la remplacer par la primitive `sort-by`

Voici un exemple :

```
sort-by [[size] of ?1 < [size] of ?2] turtles
```

Cette commande retourne une liste de tortues triée par taille (valeur de la variable tortue `size` ) croissante.

### Commander une liste d'agents

Une fois la liste d'agents obtenue, vous allez demander à chacun d'eux de faire quelque chose. À cet effet, il

faut utiliser une combinaison des commandes `foreach` et `ask` de la manière suivante :

```
foreach sort turtles [
  ask ? [
    ...
  ]
]
```

Ce morceau de code fait appel à chaque tortue successivement en fonction de son numéro d'identification en ordre croissant. Mettez `patches` à la place de `turtles` pour faire appel aux patches dans l'ordre, de gauche à droite et de haut en bas.

Si vous utilisez la commande `foreach` de cette manière, les agents de la liste exécutent les commandes transmise par `ask` séquentiellement, et non concurremment. Chaque agent termine ses commandes avant que l'agent suivant ne commence avec les siennes.

Notez qu'on ne peut pas utiliser `ask` directement sur une liste de tortues. En effet, cette primitive ne fonctionne qu'avec des ensembles d'agents ou des agents isolés..

### La performance des listes

Si votre modèle fait un usage intensif des listes, et spécialement de longues listes, vous avez tout intérêt à connaître la vitesse d'exécution des différentes opérations sur les listes afin de pouvoir écrire un code qui ne soit pas trop lent.

Les listes de NetLogo sont des listes « unidirectionnelles » (*singly linked*). Ce terme technique, issu des sciences informatiques, signifie que quand NetLogo doit trouver un membre d'une liste, il commence sa recherche au début de la liste et la parcourt, membre après membre, jusqu'à ce qu'il trouve celui qu'il recherche. Par exemple, pour trouver le centième membre, il doit d'abord passer par les 99 précédents.

Mais cela signifie aussi que certaines opérations sont particulièrement efficientes, notamment les opérations sur les têtes de listes. Les reporters `first`, `but-first` et `fput` sont très rapides : leur durée d'exécution est la même quelle que soit la longueur de la liste. Donc, si vous construisez une liste en ajoutant un membre après l'autre, l'utilisation de `fput` (mets-premier) produit un code bien plus rapide que l'utilisation de `lput` (mets-dernier). (Si cette manière de faire produit une liste en sens inverse de celle que vous vouliez, vous pouvez utiliser `reverse` pour la « retourner » une fois qu'elle aura été construite.)

Le reporter `length` est aussi très rapide car NetLogo garde toujours une trace de la longueur de chaque liste, il n'a donc jamais besoin de les mesurer véritablement.

Parmi les reporters les plus lents sur les longues listes, on trouve `item`, `lput`, `but-last`, `last` et `one-of`.

## Les math

En NetLogo, tous les nombres sont stockés en interne sous la forme de nombres à virgule flottante en double précision, tels que définis dans le standard IEEE 754. Ce sont des nombres à 64 bits composés d'un bit de signe, d'un exposant à 11 bits et d'une mantisse à 52 bits. Voir le standard IEEE 754 plus plus de détails.

Pour NetLogo, un **entier** est simplement un nombre qui semble ne pas avoir de partie décimale. Aucune distinction n'est faite entre 3 et 3.0 : ces deux notations représentent le même nombre. (C'est de cette manière que la plupart des gens utilisent les nombres au quotidien, mais ce n'est pas celle utilisée par certains langages de programmation. En effet, certains langage de programmation traitent les nombres entiers et les nombres en virgule flottante comme des types distincts.)

NetLogo affiche toujours les nombres entiers sans le point décimal et le zéro qui suit, donc sans " `.0` " :

```
show 1.5 + 1.5
observer: 3
```

Si un nombre avec une partie décimale est fourni dans un contexte où un entier est demandé, la partie fractionnaire (décimale) est simplement ignorée. Ainsi, par exemple, la commande `crt 3.5` crée trois tortues, le `0.5` étant ignoré.

Le domaine couvert par les nombres entiers va de -9007199254740992 à +9007199254740992 (±2^53, soit environ 9 quadrillions). Les opérations qui conduisent au dépassement de ces limites ne provoquent pas d'erreur d'exécution, mais la précision diminue quand les bits de poids faible sont ignorés (le nombre est arrondi) afin de pouvoir faire entrer le nombre obtenu dans les 64 bits disponibles. Avec de très grands nombres, l'opération d'arrondi peut conduire à des résultats surprenants :

```
show 2 ^ 60 + 1 = 2 ^ 60
=> true
```

Des opérations avec des nombres plus petits peuvent aussi produire des résultats surprenants lorsqu'elles comprennent certaines fractions puisque toutes les fractions ne peuvent pas être représentées exactement en notation décimale. Il y a donc automatiquement un calcul d'arrondi. Par exemple :

```
show 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6
=> 0.9999999999999999
show 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9
=> 1.0000000000000002
```

Toute opération produisant les valeurs « infini » ou « pas un nombre » génère une erreur d'exécution.

### La notation scientifique

NetLogo utilise la « notation scientifique » pour afficher les très grands et très petits nombres en virgule flottante. Exemples :

```
show 0.000000000001
=> 1.0E-12
show 50000000000000000000
=> 5.0E19
```

On reconnaît les nombres affichés en notation scientifique par la présence de la lettre E (E pour « exposant »). Ce E signifie « fois 10 puissance », donc, par exemple, 1.0E-12 signifie « 1.0 fois 10 puissance -12 » :

```
show 1.0 * 10 ^ -12
=> 1.0E-12
```

Vous pouvez aussi utiliser la notation scientifique dans le code NetLogo que vous écrivez :

```
show 3.0E6
=> 3000000
show 8.123456789E6
=> 8123456.789
show 8.123456789E7
=> 8.123456789E7
show 3.0E16
=> 3.0E16
show 8.0E-3
=> 0.0080
show 8.0E-4
=> 8.0E-4
```

Ces exemples montrent que les nombres ayant une partie décimale sont affichés en notation scientifique si

l'exposant est inférieur à -3 ou supérieur à 6. Les nombres sortant du domaine des nombres entiers (n'appartenant pas à l'intervalle -9007199254740992 +9007199254740992 ou +/-2^53) sont toujours affichés en notation scientifique :

```
show 2 ^ 60
=> 1.15292150460684698E18
```

Quand vous écrivez un nombre, la lettre E peut être en majuscule ou en minuscule. Quand il affiche un nombre, NetLogo utilise toujours la majuscule E :

```
show 4.5e20
=> 4.5E20
```

### La précision en virgule flottante

Puisqu'en NetLogo les nombres sont sujets aux limitations dues à la manière dont les nombres en virgule flottante sont représentés en binaire, vous pouvez obtenir des réponses qui sont légèrement imprécises. Par exemple :

```
show 0.1 + 0.1 + 0.1
=> 0.30000000000000004
show cos 90
=> 6.123233995736766E-17
```

C'est une conséquence inhérente à l'arithmétique en virgule flottante. On la trouve dans tous les langages de programmation qui utilisent les nombres en virgule flottante.

Si vous avez à faire à des quantités demandant une précision absolue, par exemple des francs et des centimes, une technique courante consiste à n'utiliser que des entiers (les centimes) en interne, puis à diviser le résultat par 100 pour obtenir un affichage en francs.

Si vous devez utiliser des nombres en virgule flottante, alors, dans certaines situations, vous devrez remplacer des tests d'égalité stricte tels que `if x = 1 [ ... ]` par des tests qui tolèrent une légère imprécision, par exemple `if abs (x - 1) < 0.0001 [ ... ]`.

Enfin, la primitive `precision` est utile pour arrondir les nombres en vue d'un affichage. Les moniteurs NetLogo arrondissent aussi les nombres qu'ils affichent avec une précision (un nombre de décimales) qui est configurable.

## Les nombres aléatoires

Les nombres aléatoires utilisés par NetLogo sont ce qu'il est convenu d'appeler des nombres « pseudo-aléatoires ». (Ce qui est typique en programmation.) Cela signifie qu'ils paraissent aléatoires, alors qu'ils sont en réalité générés par un processus déterministe. « Déterministe » veut dire que vous obtenez à chaque fois les mêmes résultats si vous commencez avec la même « valeur de départ » (appelée aussi parfois « germe » ou « amorce ). Nous allons expliquer dans quelques instants ce que nous entendons par «valeur de départ».

Dans le domaine de la modélisation scientifique, les nombres pseudo-aléatoires sont absolument nécessaires. Car il est important que les résultats d'une expérience scientifique soient reproductibles – afin que chacun puisse essayer par lui-même et obtienne les mêmes résultats que vous. Puisque NetLogo utilise les nombres pseudo-aléatoires, les « expériences » que vous faites avec ses modèles sont reproductibles par d'autres.

Voici comment cela fonctionne. Le générateur de nombres aléatoires de NetLogo peut être initialisé avec

une certaine « valeur de départ », qui peut être n'importe quel nombre entier. Une fois le générateur « initialisé » avec la commande `random-seed`, il produit toujours la même séquence de nombres aléatoires à partir de la même « valeur de départ » donnée. Par exemple, si vous exécutez ces commandes :

```
random-seed 137
show random 100
show random 100
show random 100
```

Vous obtiendrez toujours les nombres 95, 7, et 54, dans cet ordre.

Notez toutefois que vous n'avez la garantie d'obtenir les mêmes suites de nombres que si vous utilisez la même version de NetLogo. Il peut arriver que, quand nous développons une nouvelle version de NetLogo, le générateur de nombres aléatoires change. (Pour le moment nous utilisons un générateur appelé « Tornade de Mersenne » (*Mersenne Twister*).)

Pour créer un nombre convenant à l'amorçage aléatoire du générateur de nombres aléatoires, autrement dit une « valeur de départ », il faut utiliser le reporter `new-seed`. Ce reporter crée et retourne une « amorce » impartialement choisie dans tout l'ensemble des amorces possibles, en se basant sur la date et l'heure courante du système. Il ne retourne jamais deux fois la même amorce.

**Exemple de code** : " Random Seed Example "

Si vous ne fournissez pas l'amorce aléatoire vous-même, NetLogo lui donne une valeur basée sur la date et l'heure courantes du système. Mais il n'y a aucun moyen de savoir quelle amorce aléatoire est choisie. C'est pourquoi, si les simulations faites avec votre modèle doivent être reproductibles, vous devez fournir vous-même cette amorce aléatoire avec `random-seed`.

Les primitives NetLogo qui ont `random` dans leurs noms (`random`, `random-float`, etc.) ne sont pas les seules à utiliser des nombres pseudo-aléatoires. Bien d'autres opérations font aussi des choix aléatoires. Par exemples, les agents des ensembles d'agents sont toujours rangés dans un ordre aléatoire, les primitives `one-of` and `n-of` choisissent les agents au hasard, la commande `sprout` crée des tortues dont les couleurs et les orientations sont choisies au hasard, et le reporter `downhill` choisit un patch au hasard quand il y a deux ou plusieurs patches avec la même valeur. Tous ces choix aléatoires sont également gouvernés par l'amorce aléatoire, c'est ce qui fait que les simulations sont reproductibles.

En plus des nombres entiers et des nombres en virgule flottante uniformément distribués générés par les primitives `random` et `random-float`, NetLogo offre encore d'autres distributions aléatoires. Voyez les entrées `random-normal`, `random-poisson`, `random-exponential`, et `random-gamma` du dictionnaire NetLogo.

### Le générateur auxiliaire

Le code exécuté par les boutons ou par le Centre de commande utilise le générateur de nombres aléatoires principal.

Le code placé dans les moniteurs utilise un générateur aléatoire auxiliaire. De cette manière, même si un moniteur exécute une opération qui utilise les nombres aléatoires, le résultat de la simulation n'en est pas affecté. Il en va de même pour le code des curseurs.

### Le « hasard local »

Il est parfois nécessaire de spécifier explicitement qu'une section de code ne doive pas influencer l'état du générateur aléatoire principal de manière à ce que le résultat de la simulation n'en soit pas affecté. La commande `with-local-randomness` est justement là pour traiter ces cas. Voir cette entrée dans le Dictionnaire NetLogo pour plus d'informations.

## Les formes de tortue

Les images des tortues NetLogo sont des images vectorielles. Elles sont construites à partir des formes géométriques de base que sont les carrés, les cercles et les courbes plutôt qu'à partir d'une grille de pixels. On peut agrandir, rétrécir et/ou faire pivoter ces formes vectorielles. NetLogo place les formes vectorielles d'échelle 1, 1.5 et 2 dans des caches bitmap de manière à accélérer la vitesse d'exécution du programme.

L'image d'une tortue, stockée dans sa variable `shape`, peut être spécifiée au moyen de la commande `set`.

À leur création, les nouvelles tortues ont une forme par défaut appelée `default`. La primitive `set-default-shape` permet de remplacer la forme de tortue par défaut par une autre, ou de spécifier une forme de tortue par défaut différente pour chaque race de tortue.

La primitive `shapes` retourne une liste des différentes formes de tortues actuellement disponibles dans le modèle. Cette liste est utile si, par exemple, vous voulez affecter à une tortue une forme choisie au hasard :

```
ask turtles [ set shape one-of shapes ]
```

Utilisez l'éditeur de formes de tortue "Turtle Shapes Editor" pour créer vos propres formes de tortue, pour ajouter à votre modèle des formes provenant de notre bibliothèque de formes ou pour transférer des formes d'un modèle à l'autre. Consultez le chapitre L'éditeur de formes de ce manuel pour plus d'informations.

L'épaisseur des lignes utilisées pour dessiner les formes vectorielles peut être contrôlée au moyen de la primitive `__set-line-thickness`.

**Exemples de code** : " Breeds and Shapes Example ", " Shape Animation Example ".

## Les formes des liens

Les formes des liens sont semblables aux formes des tortues, vous devez simplement utiliser l'éditeur de formes de liens "Link Shape Editor" pour les créer et les éditer. Les formes des liens sont constituées de 0, 1, 2 ou 3 lignes qui peuvent avoir différents motifs et un indicateur d'orientation composé des mêmes éléments que les formes des tortues. Les liens possèdent aussi une variable `shape` qui peut recevoir n'importe quelle forme de lien se trouvant dans le modèle. Par défaut, les liens ont la forme `default`, toutefois, vous pouvez en changer en utilisant la primitive `set-default-shape`. Le reporter `link-shapes` retourne toutes les formes de liens incluses dans le modèle courant.

L'épaisseur des lignes d'une forme de lien est contrôlée par la variable de type lien `thickness` .

## Le compteur de cycles (Tick Counter)

Dans de nombreux modèles NetLogo, le temps de la simulation est décomposé en pas discrets appelés **cycles** ("*ticks*"). NetLogo possède un compteur de cycles intégré, ce qui permet toujours de savoir combien de pas la simulation a déjà effectués.

La valeur actuelle du compteur de cycles est affichée dans bord supérieur de la vue. (Vous pouvez utiliser le bouton "Settings" pour cacher cet affichage ou pour remplacer son étiquette "ticks" par une autre dénomination.)

Dans le code, il faut utiliser le reporter `ticks` pour obtenir la valeur courante du compteur de cycles. La commande `tick` incrémente le compteur de cycles d'une unité. La commande `clear-all` remet le compteur de cycles à zéro. Si vous voulez remettre ce compteur à zéro sans tout effacer, utilisez la commande `reset-ticks`.

Si votre modèle est conçu pour une mise à jour basée sur les cycles, alors la commande `tick` est généralement aussi utilisée pour mettre à jour l'affichage de la vue. Voir ci-dessous Les mises à jour de la Vue pour plus d'informations.

### Quand incrémenter le compteur de cycles

Nous vous conseillons d'utiliser la commande `tick` une fois que tous les agents ont terminé leurs mouvements et leurs actions, mais avant que vous ne mettiez à jour les graphiques ou ne fassiez des statistiques. De cette manière, si le code pour l'affichage des valeurs ou pour le calcul se sert du compteur de cycles, il utilisera la nouvelle valeur de ce compteur, reflétant le fait que le cycle est maintenant terminé. Exemple :

```
to go
  ask turtles [ move ] ;; les tortues se déplacent
  ask patches [ grow ] ;; les patches font pousser quelque chose
  tick                 ;; on actualise le compteur de cycles
  dessine-graphes      ;; on affiche les résultats
end

to dessine-graphes
  plotxy ticks count turtles
end
```

En mettant `tick` avant `dessine-graphes`, le code pilotant le traceur de courbes reçoit la bonne valeur du compteur de cycles lorsqu'il fait appel au reporter `ticks`.

### Les fractions de cycle

Dans la plupart des modèles, le compteur de cycle commence à 0 et est incrémenté par pas de 1, passant d'un nombre entier à un nombre entier. Mais le compteur de cycles peut aussi prendre des valeurs intermédiaires en virgule flottante .

Pour faire avancer le compteur de cycles d'un nombre fractionnaire, il faut utiliser la commande `tick-advance` qui demande une entrée numérique spécifiant de combien faire avancer le compteur de cycles.

Une utilisation typique des cycles fractionnaires est l'approximation d'un mouvement continu ou courbe. Voyez par exemple les modèles "GasLab" de la Bibliothèques des modèles (dans la section "Chemistry & Physics"). Ces modèles calculent le moment exact où un événement particulier devra avoir lieu, puis avancent le compteur de cycle exactement sur ce moment.

## Les mises à jour de la Vue

En NetLogo, la **Vue** (*View*) permet de voir sur l'écran de l'ordinateur les agents de votre modèle. Quand les agents se déplacent ou se transforment, vous les voyez se déplacer et se transformer dans la Vue.

Bien entendu, vous ne pouvez pas réellement voir vos agents. La Vue n'est qu'une image dessinée par NetLogo, montrant à quoi ressemblent vos agents à un instant donné. Une fois cet instant passé et vos agents quelque peu déplacés et/ou transformés, cette image doit être redessinée pour refléter le nouvel état du monde. Redessiner la vue est appelé **mise à jour** (*updating*) de la vue.

Mais quand la Vue doit-elle être mise à jour? Cette section explique comment NetLogo décide quand mettre à jour la Vue, et comment vous pouvez influencer le moment de cette mise à jour.

NetLogo offre deux modes de mise à jour, les mises à jour « continues » ("continuous") et les mises à jour « par cycle » ("on ticks"). Vous pouvez passer d'un type de mise à jour à l'autre au moyen du menu placé dans le haut du panneau "Interface".

Les mises à jours continues sont le réglage par défaut quand vous démarrez NetLogo ou quand vous commencez un nouveau modèle. Cependant, presque tous les modèles de notre Bibliothèque des modèles utilisent la mise à jour par cycle.

Les mises à jour continues sont les plus simples à mettre en oeuvre, mais les mises à jours par cycles offrent plus de contrôle sur le moment et la fréquence des mises à jours.

Le moment exacte de la mise à jour est important car il détermine ce que vous voyez à l'écran. Si une mise à jour se fait à un moment inapproprié, vous observerez certainement quelque chose d'inattendu – peut-être quelque chose de confus ou de trompeur.

La fréquence des mises à jour est aussi importante, car les mises à jour prennent du temps. Plus NetLogo passe de temps à mettre la Vue à jour, plus la simulation ralentit. Avec moins de mises à jour, la simulation tourne plus rapidement.

### Les mises à jour continues

Les mises à jour continues sont très simples. Avec les mises à jour continues, NetLogo met la Vue à jour un certain nombre de fois par seconde – par défaut, 50 fois par seconde quand le curseur vitesse se trouve à sa position par défaut "normal speed", au milieu de son échelle.

Si vous déplacez le curseur sur une vitesse inférieure (vers la gauche), NetLogo fait une mise à jour de la Vue plus de 50 fois par seconde, ralentissant effectivement la déroulement de la simulation. À une vitesse supérieure à la moyenne (curseur vers la droite), NetLogo met à jour moins de 50 fois par seconde, ce qui permet à la simulation de « tourner» plus rapidement. Pour les vitesses les plus grandes, les mises à jour peuvent être séparées par plusieurs secondes.

Aux vitesses les plus lentes, NetLogo fait des mises à jour si fréquentes que vous voyez les agents se déplacer (ou changer de couleur, etc.) les uns après les autres.

Si vous avez besoin de désactiver temporairement les mises à jour continues, utilisez la commande `no-display`. La commande `display` réactive les mises à jour et du même coup force une mise à jour immédiate (à moins que l'utilisateur n'ait fortement accéléré la simulation à l'aide du curseur vitesse).

### Les mises à jour contrôlées par les cycle

Comme nous l'avons déjà signalé ci-dessus dans la section Le compteur de cycles, dans bien des modèles NetLogo, le temps qui passe est divisé en pas discrets appelés **cycles** (*ticks*). Et il y a typiquement une mise à jour de la Vue par cycle, entre deux cycles successifs. C'est le comportement par défaut pour les mises à jour basées sur les cycles.

Si vous voulez des mises à jour supplémentaires, vous devez les forcer à l'aide de la commande `display`. (La mise à jour peut être sautée si l'utilisateur accélère fortement la simulation avec le curseur vitesse.)

Il n'est pas nécessaire d'utiliser le compteur de cycles pour avoir une mise à jour basée sur les cycles. Si le compteur de cycles n'avance jamais, la vue ne sera mise à jour que lorsqu'on utilise la commande `display`.

Si vous déplacez le curseur vitesse vers la droite (vitesse plus élevée), NetLogo sautera un certain nombre de mises jour qui auraient normalement dû se faire. Le déplacement du curseur vitesse vers la gauche (vitesse plus lente) n'augmente pas le nombre de mises à jour, il oblige plutôt NetLogo à faire une pause après chaque mise à jour. Plus basse est la vitesse, plus longues sont les pauses.

Même en régime de mises à jour basées sur les cycles, la vue est aussi mise à jour chaque fois qu'un bouton de l'interface remonte en position déclenché (aussi bien les boutons une-fois que les boutons pour-toujours) et aussi chaque fois qu'une commande passée dans le Centre de commande termine son travail. C'est pourquoi il n'est pas nécessaire d'ajouter une commande `display` aux boutons une-fois qui ne font pas avancer le compteur de cycles. Par contre, de nombreux boutons pour-toujours qui ne font pas avancer le compteur de cycles ont besoin d'utiliser la commande `display`. Un exemple de la Bibliothèque des modèles qui le démontre est le modèle "Life" (dans la section "Computer Science >> Cellular Automata"). Les boutons pour-toujours qui permettent à l'utilisateur de dessiner dans la vue utilisent la commande `display` pour permettre à cet utilisateur de voir ce qui se passe, même si le compteur de cycle n'est pas incrémenté.

### Quel mode de mise à jour utiliser?

Les avantages des mises à jour basées sur les cycles sur les mises à jour continues sont :

- ✔ Un comportement de mises à jour de la Vue consistant et prévisible, qui ne varie pas d'un ordinateur à l'autre ou d'une simulation à l'autre.
- ✔ Les mises à jour continues peuvent perturber l'utilisateur de votre modèle en lui laissant voir des états de la simulation qu'il n'est pas sensé voir, ce qui peut conduire à des conclusions trompeuses.
- ✔ Une vitesse plus élevée de la simulation. La mise à jour de la Vue peut prendre du temps, donc si une mise à jour par cycle est suffisante, alors la spécification d'une mise à jour seulement par cycle peut rendre le modèle plus rapide.
- ✔ Puisque les boutons `setup` ne font pas avancer le compteur de cycles, ils ne sont pas influencés par le curseur vitesse, ce qui est normalement le comportement souhaité.

Comme indiqué ci-dessus, la plus grande partie des modèles de la Bibliothèque des modèles utilisent maintenant les mises à jour basées sur les cycles.

Les mises à jour continues sont utiles pour les modèles dont la simulation n'est pas divisée en courtes phases discrètes. "Termites" en est un exemple dans la Bibliothèques des modèles. (Mais voyez aussi le modèle "State Machine Example" qui montre comment recoder le modèle "Termites" en utilisant les cycles).

Même avec les modèles qui sont normalement conçus pour une mise à jour basée sur les cycles, il peut être utile de basculer temporairement sur une mise à jour continue au cours de la mise au point du modèle. Observer ce qui se passe dans le cycle plutôt que de d'observer le résultat du cycle complet peut aider à résoudre des problèmes de fonctionnement. Après avoir basculé en mode mises à jour continues, vous pouvez utiliser le curseur vitesse pour ralentir le modèle jusqu'à ce que vous voyez les agents se déplacer les uns après les autres. N'oubliez pas de revenir ensuite en mode de mises à jour basées sur les cycles lorsque vous avez terminé car le mode de mise à jour est enregistré avec le modèle.

## Dessiner des graphiques

Les fonctionnalités de traçage de courbes de NetLogo permettent de créer et d'afficher des graphiques qui sont sensés aider à comprendre ce qui se passe au cours de la simulation.

Avant de pouvoir dessiner des graphiques, vous devez créer un ou plusieurs **traceurs de courbes** (*plots*), ou plus simplement **traceurs**, dans le panneau "Interface". Chaque traceur doit avoir son propre nom, nom qui sera utilisé dans le code développé dans le panneau "Procedures" pour faire référence au traceur correspondant.

Voir le Guide de l'Interface pour des informations supplémentaires concernant l'utilisation et le paramétrage des traceurs dans le panneau "Interface".

### Sélectionner un traceur

Si le modèle ne comporte qu'un traceur, vous pouvez directement commencer de dessiner les courbes. Mais s'il en comporte plusieurs, vous devez spécifier quel traceur vous voulez utiliser. À cet effet, utilisez la commande `set-current-plot` avec le nom du traceur entre guillemets doubles (chaîne de caractères) comme ceci :

```
set-current-plot "Distance en fonction du temps"
```

Vous devez fournir le nom du traceur exactement comme vous l'aviez orthographié quand vous avez créé le traceur. Notez que si, par la suite, vous modifiez le nom du traceur, vous devez aussi modifier tous les appels de la primitive `set-current-plot` de votre modèle afin qu'elle reçoive le nouveau nom. (Copie-coller peut se révéler utile ici.)

### Sélectionner un crayon

Quand un nouveau traceur est créé, il ne possède qu'un seul crayon. Et vous pouvez immédiatement commencer de dessiner une courbe si le traceur actif ne possède qu'un seul crayon.

La comparaison de l'évolution de plusieurs paramètres au cours de la simulation présentant souvent un grand intérêt, les traceurs de NetLogo peuvent avoir plusieurs crayons afin de pouvoir dessiner plusieurs courbes en même temps sur le même graphique. Vous pouvez donc ajouter des crayons au crayon par défaut dans l'éditeur du traceur en utilisant les contrôles de la section "Plot Pens" située au bas de la fenêtre d'édition. Chaque crayon doit alors avoir un nom différent, nom qui sera utilisé pour faire référence à ce crayon dans le code développé dans le panneau "Procedures".

Pour un traceur à plusieurs crayons, il faut spécifier quel crayon doit être utilisé pour dessiner la courbe. Si vous ne spécifiez pas de crayon, le traçage de la courbe se fera avec le premier crayon du traceur. Pour dessiner avec un autre crayon, il faut appeler la procédure `set-current-plot-pen` avec le nom du crayon placé entre guillemets doubles, comme ici :

```
set-current-plot-pen "distance"
```

### Dessiner des points

Les deux commandes de base pour dessiner effectivement (faire apparaître quelque chose dans le traceur) sont `plot` et `plotxy`.

Avec `plot`, vous ne devez spécifier que la valeur y (valeur du paramètre fourni par la simulation) que vous voulez représenter. La valeur x (qui représente généralement le cours du temps) est automatiquement 0 pour le premier point, 1 pour le deuxième, et ainsi de suite. (C'est le cas si l'« intervalle » du crayon vaut 1, sa valeur par défaut. Mais cet intervalle est modifiable.)

La commande `plot` est particulièrement utile quand vous voulez que le modèle dessine un nouveau point à chaque cycle. Exemple :

```
to setup
  ...
  plot count turtles
end

to go
  ...
  plot count turtles
end
```

Notez que dans cet exemple, si nous faisons dessiner le graphique par les deux procédures `setup` et `go`, c'est parce que nous voulons que notre graphe comprenne aussi l'état initial de la simulation. Nous dessinons à la fin de la procédure `go` et non au début, car nous voulons que le graphique soit à jour après que le code associé au bouton go ait terminé son travail.

Si vous avez besoin de spécifier les deux valeurs x et y pour le point que vous voulez dessiner, vous devez utiliser la procédure `plotxy`.

**Exemple de code** : " Plotting Example "

### D'autres types de graphes

Par défaut, les crayons des traceurs NetLogo dessinent en mode ligne de manière à ce que les points du graphique soient reliés par des lignes.

Si vous voulez déplacer le crayon sans laisser une trace, vous pouvez utiliser la commande `plot-pen-up`. Après exécution de cette commande, les commandes `plot` et `plotxy` déplacent le crayon mais ne dessinent rien. Une fois le crayon à l'endroit voulu, utilisez `plot-pen-down` pour l'abaisser, prêt à dessiner.

Si vous voulez ne dessiner que des points à la place des lignes, ou si vous voulez dessiner des barres à la place des lignes ou des points, vous devez changer le mode du crayon du traceur. Trois modes sont disponibles : "`line`" (ligne), "`bar`" (barre) et "`point`", "`line`" étant le mode par défaut.

Normalement, le mode d'un crayon est spécifié ou modifié dans l'éditeur du traceur. Cette action change son mode par défaut. Mais il est aussi possible de changer le mode du crayon temporairement en utilisant la commande `set-plot-pen-mode`. Cette commande demande un nombre en entrée: `0` pour le mode ligne, `1` pour le mode barre et `2` pour le mode point.

### Les histogrammes

Un histogramme est un type de graphique spécial qui montre combien de fois certaines valeurs, ou des valeurs dans certains intervalles, apparaissent dans une ensemble de nombres générés par votre modèle.

Supposons par exemple que les tortues de votre modèle ont une variable `age`. Vous pouvez créer un histogramme montrant la distribution des âges des tortues avec la commande `histogram`, comme ceci :

```
histogram [age] of turtles
```

Les nombres que vous voulez représenter au moyen de l'histogramme ne doivent pas obligatoirement provenir d'un ensemble d'agents; ils peuvent provenir de n'importe quelle liste de nombres.

Notez que l'utilisation de la commande `histogram` ne met pas automatiquement le crayon courant du traceur en mode barre. Si vous voulez des barres, vous devez mettre ce crayon en mode barre vous-mêmes. (Comme cela a déjà été signalé, vous pouvez changer le mode par défaut du crayon du traceur en éditant le traceur dans le panneau "Interface".)

La largeur des barres d'un histogramme est contrôlée par l'intervalle du crayon du traceur. L'intervalle par défaut peut être spécifié en éditant le traceur dans le panneau "Interface", mais vous pouvez aussi le modifier temporairement avec les commandes `set-plot-pen-interval` ou `set-histogram-num-bars`. Si vous utilisez cette dernière commande, NetLogo ajuste la valeur de l'intervalle de manière à ce que le nombre de barres spécifié couvre l'ensemble courant des valeurs de x.

**Exemple de code** : " Histogram Example "

### Effacer et réinitialiser

Vous pouvez effacer le contenu du traceur courant avec la commande `clear-plot` ou effacer le contenu de tous les traceurs avec la commande `clear-all-plots`. La commande `clear-all` efface aussi tous les traceurs en plus d'effacer tout ce qu'il a d'autre dans le modèle.

Si vous ne voulez effacer que le tracé fait par le crayon courant, utilisez la commande `plot-pen-reset`.

L'effacement du contenu d'un traceur ou la réinitialisation d'un crayon ne se limite pas à la suppression des données affichées. Ces actions réinitialisent aussi le traceur ou le crayon à leurs réglages par défaut, tels qu'ils ont été spécifiés dans le panneau "Interface" lors de la création du traceur ou lors de sa dernière édition. C'est pourquoi les effets des commandes telles que `set-plot-x-range` et `set-plot-pen-color` ne sont que temporaires.

### Le dessin automatique

Par défaut, tous les traceurs de NetLogo ont la fonctionnalité de dessin automatique "*autoplotting*" activée à leur création. Cela signifie que si le modèle essaie de dessiner un point qui sort du domaine d'affichage courant, l'échelle du graphique diminue le long d'un ou des deux axes afin que le nouveau point soit visible.

Dans l'idée que les échelles ne doivent pas changer chaque fois qu'un point est ajouté, elles laissent un espace supplémentaire chaque fois qu'elle croissent. Cet espace s'accroît de 25% horizontalement et de 10% verticalement si nécessaire.

Si vous voulez désactiver cette fonctionnalité, éditez le traceur et désélectionnez le bouton "Autoplot". Pour le moment, il n'est pas possible de désactiver cette fonctionnalité pour un seul axe; elle s'applique toujours aux deux axes.

### Les crayons de traceur temporaires

La plupart des graphiques peuvent être dessinés avec un nombre constant de crayons. Mais certains ont des besoins plus complexes et il serait intéressant que le nombre de crayons disponibles pour les dessiner puisse changer en fonction de certaines conditions. C'est pour des situations de ce genre que NetLogo offre la possibilité de créer des crayons « temporaires » à partir du code, puis de les utiliser pour dessiner. Ces crayons sont appelés temporaires parce qu'ils disparaissent quand le contenu de traceur est effacé (avec les commandes `clear-plot`, `clear-all-plots` ou `clear-all`).

Pour créer un crayon temporaire, utilisez la commande `create-temporary-plot-pen`. Une fois le crayon créé, vous pouvez l'utiliser comme n'importe quel crayon ordinaire. Par défaut, le nouveau crayon est abaissé, sa couleur est noire, son intervalle est de 1 et son mode de tracé est ligne. Tous ces paramètres sont modifiables par des commandes (voir la section Les Traceurs du Dictionnaire NetLogo).

### Afficher la légende

Vous pouvez afficher la légende des courbes d'un traceur en cochant le bouton "Show legend" dans la fenêtre d'édition du traceur. Si vous ne voulez pas voir apparaître un certain crayon dans la légende, il faut désactiver le bouton "Show in Legend" de ce crayon.

### Conclusion

Tous les aspects du système des traceurs de courbes de NetLogo n'ont pas été abordés ici. Consultez la section Les Traceurs du Dictionnaire NetLogo pour en savoir plus sur d'autres commandes et reporters s'occupant des graphiques et des traceurs.

De nombreux exemples de modèles de la Bibliothèque des modèles montrent diverses techniques avancées de traçage de graphiques. Consultez aussi les exemples de code suivants :

**Exemples de code** : " Plot Axis Example ", " Plot Smoothing Example " et " Rolling Plot Example "

## Les chaînes de caractères (strings)

Pour écrire une chaîne de caractères en NetLogo, il faut l'entourer de guillemets doubles anglais, comme ceci : `"Bonjour le monde"`.

La chaîne vide est obtenue en ne mettant rien (pas même un espace) entre ces guillemets, comme ceci : `""`.

La plupart des primitives dédiées aux listes fonctionnent aussi avec les chaînes :

```
but-first "string" => "tring"
but-last "string" => "strin"
empty? "" => true
empty? "string" => false
first "string" => "s"
item 2 "string" => "r"
last "string" => "g"
length "string" => 6
member? "s" "string" => true
member? "rin" "string" => true
member? "ron" "string" => false
position "s" "string" => 0
position "rin" "string" => 2
```

```
position "ron" "string" => false
remove "r" "string" => "sting"
remove "s" "strings" => "tring"
replace-item 3 "string" "o" => "strong"
reverse "string" => "gnirts"
```

Quelques primitives sont spécifiques aux chaînes, telles que : `is-string?`, `substring` et `word` :

```
is-string? "string" => true
is-string? 37 => false
substring "string" 2 5 => "rin"
word "tur" "tle" => "turtle"
```

Les chaînes peuvent être comparées au moyen des opérateurs `=` , `!=` , `<` , `>` , `<=` et `>=` .

Si vous avez besoin de mettre un caractère spécial dans une chaîne, utilisez les séquences d'échappement suivantes :

- `\n` = *newline*
- `\t` = *tab*
- `\"` = *double quote*
- `\\` = *backslash*

## Les sorties à l'écran (affichage)

Cette section est consacrée l'affichage des données sur l'écran. Ce qui est affiché à l'écran peut aussi être enregistré par la suite dans un fichier au moyen de la commande `export-output`. Si vous avez besoin d'une méthode plus souple pour écrire des données dans un fichier externe, voyez la section suivante, E/S de fichiers.

Les commandes NetLogo de base pour générer une sortie à l'écran sont `print`, `show`, `type` et `write`. Ces commandes permettent d'écrire dans la zone d'affichage du Centre de commande.

Pour une description détaillée de ces commandes, consultez leurs entrées dans le Dictionnaire NetLogo. En voici un résumé :

- ✔ `print` est utile dans la plupart des situations.
- ✔ `show` affiche aussi le nom de l'agent qui est à l'origine de l'affichage.
- ✔ `type` permet d'écrire plusieurs sorties sur la même ligne.
- ✔ `write` permet de formater les données affichées dans un format qui peut ensuite être relu avec `file-read`.

Un modèle NetLogo peut aussi avoir en option une « zone de sortie » "Output" dans son panneau "Interface", zone distincte et indépendante du Centre de Commande. Pour y envoyer une sortie (plutôt que dans le Centre de commande), utilisez les commandes `output-print`, `output-show`, `output-type` et `output-write` .

Le contenu de la zone de sortie peut être effacé avec la commande `clear-output` et enregistré dans un fichier avec la commande `export-output`. Le contenu de la zone de sortie est aussi sauvegardé par la commande `export-world`. La commande `import-world` efface la zone de sortie et remplace son contenu par les données du fichier importé. Il faut toutefois noter que de grandes quantités de données envoyées et affichées dans la zone de sortie peuvent accroître considérablement la taille des mondes exportés.

Si vous utilisez les commandes `output-print`, `output-show`, `output-type`, `output-write`, `clear-output` ou `export-output` dans un modèle qui n'a pas de zone de sortie séparée, ces primitives agissent alors sur la zone d'affichage du Centre de commande.

## E/S de fichiers

NetLogo possède un ensemble de primitives qui donnent la possibilité d'interagir avec des fichiers externes. Elles commencent toutes par le préfixe `file-`.

Lorsqu'on travaille avec des fichiers , les deux principaux modes sont la **lecture** (*reading*) et l'**écriture** (*writing*). La différence réside dans la direction du flux de données. Quand vous lisez une information dans un fichier, les données qui sont stockées dans le fichier « coulent » dans votre modèle. D'un autre côté, l'écriture permet aux données de « couler » de votre modèle vers et dans un fichier.

Quand un modèle NetLogo tourne en tant qu'applet dans un navigateur internet, il ne peut que lire les données des fichiers qui se trouvent sur le serveur et dans le même répertoire que le fichier du modèle. Les applets ne peuvent pas écrire dans un fichier.

Quand on travaille avec des fichiers, il faut toujours commencer par utiliser la primitive `file-open`. Cette dernière spécifie le fichier avec lequel vous voulez travailler. Aucune des autres primitives dédiées aux fichiers ne fonctionne si le fichier n'a pas d'abord été ouvert.

Immédiatement après cette primitive, vous devez utiliser l'une des primitives `file-` suivantes qui permettent de spécifier dans quel mode, **lecture** ou **écriture**, se trouvera le fichier jusqu'à ce qu'il soit refermé. Pour passer d'un mode à l'autre, il faut fermer puis ré-ouvrir le fichier.

Les primitives de lecture comprennent `file-read`, `file-read-line`, `file-read-characters` et `file-at-end?`. Notez qu'il faut que le fichier existe déjà avant que vous puissiez l'ouvrir en mode lecture.

**Exemple de code** : " File Input Example "

Les primitives pour écrire dans un fichier ressemblent à celles utilisées pour écrire dans le Centre de commande, sauf qu'ici, la sortie est enregistrée dans un fichier. Elles comprennent `file-print`, `file-show`, `file-type` et `file-write`. Notez que vous ne pouvez jamais « écraser » (*overwrite*) des données. En d'autres mots, si vous voulez écrire des données dans un fichier qui en contient déjà, ces nouvelles données seront ajoutées à la fin du fichier. (Si vous voulez écraser le contenu d'un fichier, utilisez d'abord `file-delete` pour l'effacer puis ouvrez-le pour y écrire.)

**Exemple de code** : "File Output Example"

Quand vous avez fini de travailler avec un fichier, vous devez utiliser la commande `file-close` pour terminer votre session avec ce fichier et le refermer. Si par la suite vous voulez supprimer ce fichier, utilisez la primitive `file-delete` pour l'effacer. Si plusieurs fichiers sont ouverts en même temps et qu'il faut en fermer quelques-uns, il faut d'abord en sélectionner un avec `file-open` avant de le fermer avec `file-close`, manoeuvre à répéter pour tous les fichiers à fermer, comme dans l'exemple ci-dessous :

```
;; Ouverture de 3 fichiers
file-open "myfile1.txt"
file-open "myfile2.txt"
file-open "myfile3.txt"

;; et maintenant fermeture ciblée des 3 fichiers
file-close  ;; ferme le fichier myfile3.txt qui est le fichier courant
file-open "myfile2.txt"
file-close
file-open "myfile1.txt"
file-close
```

Ou, si vous savez que vous voulez fermer tous les fichiers ouverts, il suffit d'utiliser `file-close-all`.

Deux primitives valent encore la peine d'être signalées, ce sont `file-write` et `file-read`. Ces primitives ont été tout spécialement développées pour faciliter l'enregistrement et la lecture des constantes NetLogo telles

que les nombres, les listes, les booléens et les chaînes. `file-write` écrit toujours les variables de manière telle que `file-read` puisse les lire et les interpréter correctement.

```
file-open "myfile.txt"  ;; Ouvrir le fichier en écriture
ask turtles
 [ file-write xcor file-write ycor ]
file-close

file-open "myfile.txt"  ;; Ouvrir le fichier en lecture
ask turtles
 [ setxy file-read file-read ]
file-close
```

**Code Examples** : " File Input Example " et " File Output Example "

### À l'utilisateur de choisir

Les primitives `user-directory`, `user-file` et `user-new-file` sont utiles quand vous voulez que l'utilisateur puisse choisir un répertoire ou un fichier dans lequel ou avec lequel travailler.

## Les animations

Cette section décrit comment enregistrer une animation représentant une simulation NetLogo sous la forme d'une séquence vidéo QuickTime.

Il faut commencer par l'utilisation de la commande `movie-start` pour créer une nouvelle vidéo. Le nom de fichier que vous fournissez doit se terminer par `.mov` , extension servant à identifier les séquences vidéos QuickTime.

Pour ajouter une image à la séquence vidéo, utilisez `movie-grab-view` si vous voulez enregistrer uniquement le contenu de la Vue courante (l'aspect du monde des tortues) ou `movie-grab-interface` si vous voulez enregistrer la totalité du panneau "Interface". À l'intérieur d'une séquence vidéo, vous ne pouvez utiliser qu'une seule de ces deux primitives : vous ne pouvez pas mélanger les deux types de prises de vues.

Quand vous avez ajouté toutes les images voulues, refermez la séquence vidéo avec `movie-close`.

```
;; exporter une vidéo de 30 images de la vue
setup
movie-start "out.mov"
movie-grab-view ;; montrer l'état initial
repeat 30
  [ go movie-grab-view ]
movie-close
```

Par défaut, une vidéo est projetée à 15 images par seconde. Pour faire une vidéo avec une fréquence d'images différente, appelez la primitive `movie-set-frame-rate` avec un autre nombre d'images par seconde. Il faut spécifier le nombre d'images par seconde après l'appel de `movie-start` mais avant d'enregistrer la première image.

Pour connaître la fréquence d'images de la séquence vidéo ou le nombre d'images déjà enregistrées, appelez la primitive `movie-status` qui retourne une chaîne décrivant le statut de la vidéo courante.

Pour supprimer une séquence vidéo et effacer le fichier vidéo, appelez la primitive `movie-cancel`.

**Exemple de code** : " Movie Example "

Les séquences vidéos faites avec NetLogo sont des fichiers QuickTime non compressés. Pour projeter une vidéo QuickTime, vous pouvez utiliser le QuickTime Player téléchargeable gratuitement sur le site d'Apple.

N'étant pas compressées, les séquences vidéos peuvent occuper beaucoup de place sur le disque. Vous voudrez certainement les compresser avec des programmes adéquats. Ces logiciels offrent généralement un grand choix de modes de compression. Certains modes de compression travaillent sans pertes de données alors que d'autres compriment avec pertes. Comprimer avec pertes signifie que, dans le but de réduire la taille des fichiers, certains détails des images de la vidéo sont ignorés, donc perdus. En fonction de la nature de la simulation enregistrée, vous aurez certainement tout intérêt à éviter toute perte de données (donc à utiliser un mode de compression sans pertes), par exemple dans le cas où la vue contient des détails au niveau des pixels.

QuickTime Pro est un logiciel capable de compresser des vidéos QuickTime aussi bien sur les plates-formes Mac que Windows. Sur Mac, iMovie peut aussi très bien remplir ce rôle.

## La perspective

Les Vues 2D et 3D montrent le monde du point de vue de l'observateur. Par défaut, l'observateur regarde le monde d'en haut, situé quelque part le long de la partie positive de l'axe Z passant par l'origine (le centre du patch 0 0). Vous pouvez modifier le point de vue de l'observateur en utilisant les commandes observateur `follow`, `ride` et `watch` et les commandes tortue `follow-me`, `ride-me` et `watch-me`. Quand il est en mode poursuite ("`follow`") ou chevauchée ("`ride`"), l'observateur se déplace avec l'agent sujet autour du monde. La différence entre ces deux modes n'est perceptible que dans la Vue 3D. Dans la Vue 3D, l'utilisateur peut modifier sa distance par rapport à l'agent au moyen de la souris. Quand l'observateur suit l'agent à une distance nulle, c'est comme s'il le chevauchait. Quand l'observateur est en mode observation ("`watch`"), il suit les mouvements d'une tortue des yeux sans se déplacer lui-même. Dans les deux types de Vue (2D et 3D), un spot illumine le sujet (la tortue observée) et, dans la vue 3D, l'observateur fait toujours face au sujet. Pour savoir quel agent se trouve dans la « ligne de mire » de l'observateur, vous pouvez utiliser le reporter `subject`.

**Exemple de code** : " Perspective Example "

## Le dessin (drawing)

Dans la Vue 2D, le monde de NetLogo est constitué de trois couches superposées. À la base se trouve la couche des patches, au milieu la couche de dessin et au sommet la couche des tortues. Ces deux dernières couches sont transparentes pour permettre de voir ce qu'il y a en dessous.

Le dessin est la couche (le calque) sur laquelle les tortues peuvent laisser des marques visibles. Cette couche est initialement vide et totalement transparente.

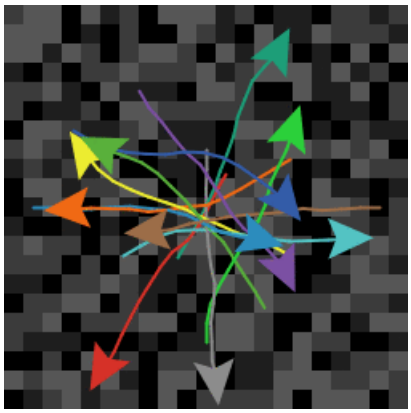
Seul l'utilisateur peut voir ce qui se trouve sur la couche de dessin, les tortues (et les patches) ne le peuvent pas. Elles ne peuvent ni « sentir » ce qui se trouve sur cette couche, ni y réagir. Le dessin n'est là que pour les utilisateurs qui le regardent.

Les tortues peuvent y dessiner et y effacer des lignes au moyen des commandes `pen-down` et `pen-erase`. Quand le crayon d'une tortue est abaissé (`pen-down`) ou quand ce crayon est transformé en gomme (`erase`), la tortue dessine (ou efface) une ligne derrière elle chaque fois qu'elle se déplace. Les lignes sont de la même couleur que la tortue. Pour ne plus dessiner (ou effacer), il suffit de lever le crayon avec `pen-up`.

Les lignes dessinées par la tortue ont normalement une épaisseur de 1 pixel. Si vous voulez des lignes d'une autre épaisseur, donnez à la variable `pen-size` une autre valeur avant de dessiner (ou d'effacer). Cette variable a toujours la valeur 1 pour les nouvelles tortues.

Pour les lignes dessinées par une tortue se déplaçant dans un mode qui ne précise pas son orientation, par exemple avec des commandes telles que `setxy` ou `move-to`, c'est le chemin le plus court obéissant à la topologie qui est dessiné.

Ci-dessous, le dessin fait par quelques tortues au-dessus d'un pavage de patches colorés de manière aléatoire. Notez comment les tortues passent au-dessus des lignes et comment les lignes couvrent les patches. Les variables `pen-size` des tortues ont reçu la valeur 2.



La commande `stamp` demande à la tortue de laisser son image sur le dessin, image qui apparaîtra lorsqu'elle se déplacera. `stamp-erase` efface tous les pixels du dessin placés au-dessous de la forme de la tortue.

Pour effacer tout le dessin, utilisez la commande observateur `clear-drawing`. (Vous pouvez aussi utiliser `clear-all`, qui efface en même temps aussi tout le reste.)

### Importer une image

La commande observateur `import-drawing` permet d'importer un fichier image d'une mémoire de masse pour mettre l'image sur le dessin.

`import-drawing` n'est utile que pour avoir une image d'arrière-plan pour les gens qui regardent la Vue. Si vous voulez que les tortues et les patches puissent réagir par rapport à l'image, vous devez utiliser à la place `import-pcolors` ou `import-pcolors-rgb`.

### Comparaison à d'autres Logo

En NetLogo, le dessin fonctionne quelque peu différemment que dans les autres Logo.

Les principales différences sont :

- ✔ Les crayons des nouvelles tortues sont levés, non abaissés.
- ✔ Plutôt que d'utiliser une commande `fence` pour confiner les tortues à l'intérieur des frontières, vous devez, en NetLogo, éditer le monde et désactiver l'« enroulement » des tortues.
- ✔ Il n'y a pas de `screen-color`, `bgcolor` ou `setbg`. Vous pouvez construire une arrière-plan solide en colorant les patches, par exemple avec `ask patches [ set pcolor blue ]`

Les fonctionnalités de dessins non supportées par NetLogo sont :

- Il n'y a pas de commande `window`. Cette commande est utilisée dans certains autres Logo pour permettre à la tortue de se déplacer sur un plan infini.
- Il n'y a pas de commandes `flood` ou `fill` pour remplir une zone fermée avec une certaine couleur.

# La topologie

La topologie du monde NetLogo peut avoir quatre formes possibles : un **tore** (*torus*), une **boîte** (*box*), un **cylindre vertical** (*vertical cylinder*) et un **cylindre horizontal** (*horizontal cylinder*). La topologie est contrôlée en activant ou en désactivant l'« enroulement » du monde dans les directions X et Y. Le monde par défaut est un tore, comme le furent tous les mondes NetLogo avant la version 3.1.

Un tore enroule le plan du monde dans les deux directions X et Y, autrement dit, les bords supérieur et inférieur du monde se rejoignent tout comme le font les bords gauche et droit. Donc, si une tortue se déplace au-delà du bord droit du monde, elle réapparaît au bord gauche, à la même hauteur. Il en va de même pour les bords supérieur et inférieur.

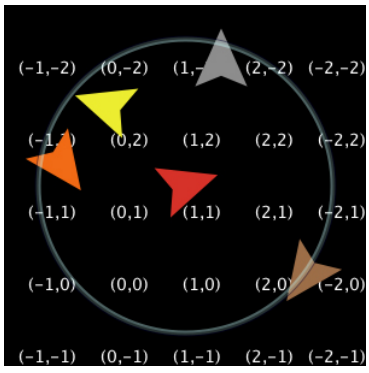
Une boîte n'a aucun enroulement. Le monde est entouré d'une bordure infranchissable et les tortues qui essaient de se déplacer au-delà des bords de ce monde ne le peuvent pas. Notez que les patches placés le long de ses bords ont moins de huit voisins, ceux des coins en ont trois, les autres cinq.

Les cylindres horizontaux et verticaux n'enroulent le plan du monde que dans une direction. Un cylindre horizontal (son axe est horizontal) s'enroule verticalement, ce qui fait que le haut du monde est connecté à la base, les bords gauche et droit sont barrés. Un cylindre vertical est l'opposé, il s'enroule horizontalement (son axe est vertical) de manière à ce que ses bords gauche et droit soient connectés alors que ses bords supérieur et inférieur sont barrés.

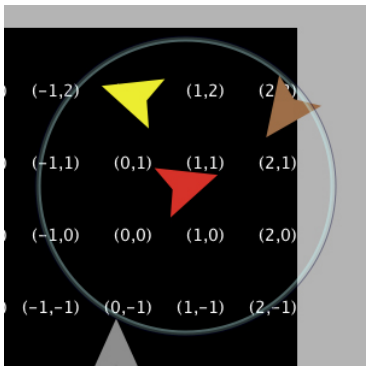
**Exemple de code** : " Neighbors Example "

Dès sa version 3.0, NetLogo offre des réglages permettant d'activer la visualisation de l'enroulement de manière à ce que, si la forme d'une tortue dépasse un bord, la partie de la forme qui dépasse le bord apparaisse sur le bord opposé. (Les tortues elles-mêmes ne sont que des points qui n'occupent aucun espace, elles ne peuvent donc pas se trouver sur deux bords du monde en même temps. Dans la vue, elles paraissent occuper un certain espace puisqu'elles y sont représentées graphiquement par leurs formes.)

L'enroulement affecte aussi l'apparence de la vue quand vous suivez une tortue. Sur un tore, où que la tortue aille, vous verrez toujours le monde entier autour d'elle.




Avec une boîte ou un cylindre, le monde a des barrières et les zones hors frontières sont dessinées en gris :



**Exemples de code** : " Termites Perspective Demo " (tore), " Ants Perspective Demo " (boîte)

Alors que les réglages offerts par la version 3.0 ne contrôlaient que l'*apparence* de l'enroulement dans la vue, NetLogo 3.1 possède des réglages qui spécifient si le monde s'enroule « pour de vrai », c'est-à-dire si les bords opposés sont réellement connectés. Ces nouveaux réglages d'enroulement déterminent la topologie du monde, autrement dit si le monde est un tore, une boîte ou un cylindre. Ils influencent le comportement et non plus seulement l'apparence visuelle du monde du modèle.

Par le passé, les auteurs de modèles devaient écrire du code spécial pour simuler un monde en boîte à l'aide de primitives spéciales « non-enroulantes ». Des versions non-enroulantes étaient disponibles pour les primitives `distance(xy)`, `in-radius`, `in-cone`, `face(xy)` et `towards(xy)`. À partir de NetLogo 3.1, ces versions ne sont plus nécessaires. C'est maintenant la topologie qui contrôle si les primitives « enroulent » ou non. Ces primitives utilisent toujours le chemin le plus court autorisé par la topologie. Par exemple, la distance du centre du patch occupant le coin inférieur gauche (`min-pxcor`, `min-pycor`) et le centre du patch occupant le coin supérieur droit (`max-pxcor`, `max-pycor`) sera la suivante, en fonction de la topologie et étant donné que les pxcor et pycor min et max sont +/-2 :

- **Tore** – `sqrt(2)`, soit ~ 1.414 (cette valeur sera toujours la même quelles que soient les dimensions du monde, puisque ces patches sont directement voisins en diagonale dans un tore.)
- **Boîte** – `sqrt(world-width^2 + world-height^2)`, soit ~ 7.07.
- **Cylindre vertical** – `sqrt(world-height^2 + 1)`, soit ~ 5.099.
- **Cylindre horizontal** – `sqrt(world-height^2 + 1)`, soit ~ 5.099.

Toutes les autres primitives agissent de même pour la distance. Si vous aviez utilisé auparavant des primitives non-enroulantes, nous vous recommandons de les enlever et de modifier à la place la topologie du monde.

Plusieurs raisons devraient vous inciter à modifier vos anciens modèles pour qu'ils utilisent les différentes topologies plutôt que des primitives non-enroulantes.

Premièrement, nous pensons que puisque vous utilisez des primitives non-enroulantes, vous modélisez en fait un monde qui n'est pas un tore. Si vous utilisiez une topologie correspondant au monde que vous modélisez, NetLogo effectuerait automatiquement pour vous les tests de franchissement des frontières, ce qui devrait vous faciliter la tâche, rendre le code plus simple à comprendre et ajouter des effets visuels permettant à l'utilisateur du modèle de mieux comprendre ce que vous avez modélisé. Notez qu'avec les primitives non-enroulantes, il était même très difficile de modéliser des mondes cylindriques puisque ces primitives retournent la distance ou l'orientation alors que l'enroulement n'est pas autorisé dans certaines directions.

Vos modèles peuvent contenir des erreurs. Si vous utilisez une combinaison des deux types de primitives (non-enroulantes et enroulantes), il y aura toujours un bogue dans votre modèle, que ce soit pour une raison ou une autre (nous avons trouvés quelques bogues dans nos modèles). Ainsi, le modèle "Conductor" comparait une distance non-enroulante à une distance pour tester si la position suivante était « enroulée » autour du monde et, dans ce cas, l'électron sortait du système. C'est une manière astucieuse de résoudre le problème, mais elle présente malheureusement un défaut. Les électrons qui s'enroulent dans la direction Y quittaient aussi le système, ce qui n'était pas correct dans notre cas. La seule manière correcte de quitter le système étant d'atteindre la cathode à l'extrémité gauche du fil de fer.

Si vous supprimez les commandes non-enroulantes, la topologie n'est plus codée dans le modèle. Il est donc plus facile de tester différentes formes du monde sans un long travail de recodage (vous pourriez avoir besoin d'ajouter quelques tests pour passer du tore à la boîte, ce qui est expliqué plus en détails dans la section "Comment convertir votre modèle").

Notez que, bien que nous ayons retiré les entrées des primitives non-enroulantes du Dictionnaire, elles sont toujours disponibles dans le langage NetLogo lui-même. Ceci dans le but de vous permettre de faire tourner les anciens modèles sans avoir besoin de les modifier.

### Comment convertir votre modèle

Quand vous ouvrez pour la première fois un ancien modèle avec une version 3.1 ou supérieure, NetLogo remplace automatiquement toutes les occurrences de ( `screen-edge-x` ) par `min-pxcor` et toutes les occurrences de ( `screen-edge-x` ) par `max-pxcor` (il fait la même chose pour les primitives en Y). Bien ce que ne soit pas directement en relation avec les modifications de la topologie, vos devriez aussi réfléchir si le déplacement de l'origine du monde loin du centre de la Vue garde encore tout son sens pour votre modèle. Avant NetLogo 3.1, le monde devait être symétrique par rapport à l'origine, c'est-à-dire que le monde devait avoir une largeur et une hauteur semblables. Ce n'est maintenant plus le cas puisque vous pouvez utiliser n'importe quelle combinaison de minima et de maxima, pourvu que le point (0,0) existe et soit toujours dans le monde. Si vous ne modélisez logiquement que dans un ou deux quadrants, ou si l'utilisation exclusive des nombres positifs peut simplifier votre code, vous pouvez envisager de modifier votre modèle. Si vous aviez modélisé quelque chose qui nécessitait une grille uniforme, vous serez certainement content de vous débarrasser des astuces de programmation qui la rendaient possible dans le passé.

**Exemples de code** : " Lattice Gas Automaton ", " Binomial Rabbits " et " Rugby "

Nous avons ajouté de nouvelles primitives à la version 3.1 de NetLogo, primitives qui sont essentielles si vous modifiez la topologie et qui restent intéressantes même si vous ne le faites pas. `random-pxcor`, `random-pycor`, `random-xcor` et `random-ycor` retournent des valeurs aléatoires entrant dans les limites minimales et maximales (x et y). Dans les anciennes versions de NetLogo, nous faisions souvent appel à l'enroulement pour placer aléatoirement les tortues à travers le monde au moyen du code `setxy random-float screen-size-x random-float screen-size-y`. Toutefois, si l'enroulement n'est pas autorisé dans une direction ou une autre, ce morceau de code ne marche pas (vous obtenez une erreur d'exécution si vous essayez de placer des tortues hors du monde). Quelle que soit la topologie, il est plus simple et plus direct d'utiliser `setxy random-xcor random-ycor` à la place.

Pour convertir un ancien modèle dans le but d'utiliser une topologie, vous devez d'abord décider quels réglages décrivent le mieux le monde modélisé. Si la réponse ne vous paraît pas immédiatement évidente en vous basant sur le mode réel (une chambre est une boîte, un fil de fer est un cylindre), voici quelques indices qui peuvent vous aider. Si quelque part dans le code vous testez les limites du monde ou si certains patches ne sont pas considérés comme des voisins des patches de l'autre bord de la Vue, il est à peu près sûr que vous n'utilisez pas un tore. Si vous testez les limites dans les deux directions X et Y, c'est une boîte, si le test est en X seulement, c'est un cylindre horizontal, s'il est en Y seulement, c'est un cylindre vertical.

Si vous aviez utilisé des primitives non-enroulantes, vous ne modélisiez certainement pas un tore. Soyez toutefois prudent avec ce critère si vous aviez utilisé un mélange de primitives enroulantes et non-enroulantes. Il se pourrait en effet que vous ayez utilisé une primitive non-enroulante pour un élément visuel alors que le reste du monde NetLogo est bien un tore.

Après avoir déterminé la topologie utilisée et l'avoir modifiée en éditant la Vue dans le panneau "Interface", vous aurez peut-être à apporter quelques petites modifications au code. Si vous avez décidé que le monde est un tore, il n'y aura probablement aucun changement à faire. Si votre modèle n'utilise que les voisins des patches et la diffusion, vous n'aurez probablement pas à faire beaucoup de modifications.

Si votre modèle contient des tortues qui se déplacent, votre prochaine étape consistera à déterminer ce qui se passe lorsqu'elles atteignent le bord du monde. Il y a quelques options usuelles : soit la tortue rebondit (systématiquement ou aléatoirement) dans le monde, soit elle quitte le système (elle meurt), soit elle est cachée. Il n'est plus nécessaire de tester le franchissement les limites du monde en utilisant les coordonnées des tortues, il suffit simplement demander à NetLogo si la tortue est sur le bord du monde. Il y a deux moyens d'y parvenir. Le plus simple est d'utiliser la primitive `can-move?` :

```
if not can-move? distance [ rt 180 ]
```

`can-move?` retourne simplement `true` (vrai) si la position située à `distance` de la tortue est encore dans le monde NetLogo, sinon elle retourne `false` (faux). Dans ce cas, si la tortue est sur le bord du monde, elle revient simplement par le chemin qu'elle a pris pour y aller. Vous pouvez aussi utiliser `patch-ahead 1 !=`

nobody à la place de can-move?. Si vous devez faire quelque chose de plus subtil que de simplement revenir en arrière, il peut être intéressant d'utiliser les primitives patch-at avec dx et dy.

```
if patch-at dx 0 = nobody [
  set heading (- heading)
]
if patch-at 0 dy = nobody [
  set heading (180 – heading)
]
```

Ce morceau de code teste si la tortue heurte un mur horizontal ou vertical et la fait rebondir contre ce mur.

Dans certains modèles, la tortue meurt si elle ne peut plus avancer (elle quitte le système, comme dans "Conductor"ou "Mousetraps").

```
if not can-move? distance[ die ]
```

Si vous déplacez la tortue en utilisant setxy plutôt que forward, vous devez faire un test pour vous assurer que le patch sur lequel vous voulez mettre la tortue existe bien car setxy génère une erreur d'exécution si les coordonnées reçues sortent du monde. C'est une situation courante quand le modèle simule un plan infini et que les tortues hors de la vue doivent être simplement cachées.

```
let new-x new-value-of-xcor
let new-y new-value-of-ycor

ifelse patch-at (new-x - xcor) (new-y - ycor) = nobody
  [ hide-turtle ]
  [ setxy new-x new-y
    show-turtle ]
```

Plusieurs modèles de la Bibliothèque des modèles utilisent cette technique, par exemple "Gravitation", "N-Bodies" et "Electrostatics" en sont de bons exemples.

En utilisant une topologie différente, vous obtenez la diffusion « pour rien » (ce qui était très difficile par le passé). Chaque patch diffuse et répartit également entre ses différents voisins la quantité spécifiée dans la variable diffuse. Si le patch possède moins de huit voisins, (ou 4 si vous utilisez diffuse4), ce qui n'a pas pu être diffusé reste dans le patch diffuseur. Cela signifie que la somme totale des valeur de cette variable patch dans le monde reste constante. Si vous avez du code spécial chargé de gérer la diffusion, vous pouvez le supprimer. Toutefois, si vous voulez que la matière diffusée déborde des limites du monde comme ce serait le cas sur un plan infini, vous devez toujours nettoyer (vider) les bords à chaque pas comme dans l'exemple "Diffuse Off Edges".

## Les liens

Un **lien** (*link*) est un agent qui relie deux tortues. Le deux tortues placées aux extrémités du lien sont appelées des **noeuds** (*nodes*). Le lien est toujours dessiné sous la forme d'une ligne entre les deux tortues. Les liens n'ont pas de position (des coordonnées) comme les tortues. Ils ne sont pas considérés comme se trouvant sur un patch particulier et vous ne pouvez pas déterminer la distance d'un lien à un autre point.

Il y a deux types de liens, les liens **non-orientés** (*undirected*) et les liens **orientés** (*directed*). Un lien orienté « sort » d'un noeud pour « entrer » dans un autre noeud ou « vient » d'un noeud pour aller « vers » un autre noeud. La relation parent-enfant peut être modélisée à l'aide d'un lien orienté. Un lien non-orienté est vu de la même manière par les deux noeuds, chaque noeud a un lien « avec » un autre noeud. La relation entre conjoints ou entre frères (ou soeurs) peut être modélisée par un lien non-orienté.

Dans un modèle NetLogo, il y a un ensemble d'agents global réunissant tous les liens, comme il en existe un pour toutes les tortues et un autre pour tous les patches. Vous pouvez créer des liens non-orientés avec les commandes `create-link-with` et `create-links-with`, et des liens orientés en utilisant les commandes `create-link-to`, `create-links-to`, `create-link-from` et `create-links-from`. Une fois le premier lien (orienté ou non-orienté) créé, tous les liens sans race doivent être de ce même type (les liens pouvant aussi appartenir à des races différentes, comme les tortues). Il n'est en effet pas possible d'avoir deux liens sans race dont l'un est orienté et l'autre est non-orienté. Une erreur d'exécution est générée si vous essayez de le faire. (Si tous les liens sans race meurent, alors vous pouvez créer des liens sans race d'un type autre que celui des liens précédents.)

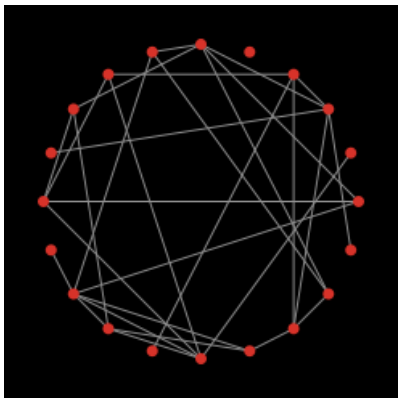
En général, les primitives qui fonctionnent avec des liens orientés ont les suffixes `-in`, `-out`, `-to` et `-from` dans leur nom. Les primitives des liens non-orientés les omettent ou utilisent «`-with`».

Les races de liens, comme les races de tortues, permettent de définir différents types de liens dans votre modèle. Les races de liens doivent être soit orientées, soit non-orientées, contrairement aux liens sans race, car ils sont définis à la compilation plutôt qu'à l'exécution. Les races de liens sont déclarées avec les mots-clés `undirected-link-breed` et `directed-link-breed`. Les liens d'une race peuvent être créés avec les commandes `create-<breed>-with` et `create-<breeds>-with` pour les races non-orientées et les commandes `create-<breed>-to`, `create-<breeds>-to`, `create-<breed>-from` et `create-<breeds>-from` pour les races de liens orientés.

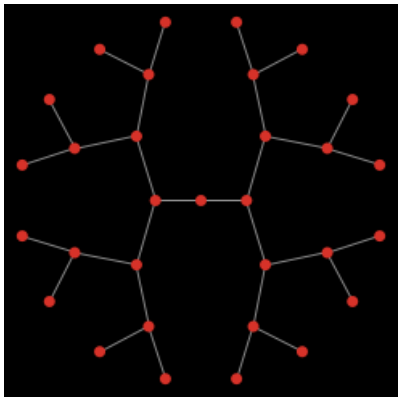
Il ne peut pas y avoir plus d'un lien non-orienté de la même race (ou plus de deux liens orientés) entre deux agents, ni plus d'un lien de la même race orienté dans la même direction entre une paire d'agents. Vous pouvez avoir deux liens orientés de la même race (ou deux liens sans race) entre une paire d'agents si ces liens sont de sens (d'orientation) opposés.

### Les réseaux de liens

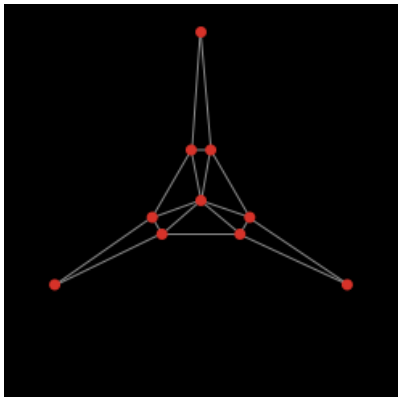
En tant que support des réseaux, nous avons aussi ajouté plusieurs primitives qui vous aiderons à visualiser les réseaux. La plus simple est `layout-circle` qui distribue les agents régulièrement autour du centre du monde sur un cercle de rayon donné.



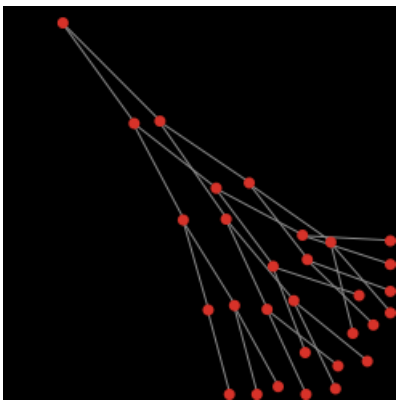
La distribution `layout-radial` est une bonne disposition si vous avez quelque chose qui ressemble à une structure en arbre, et elle fonctionnera même s'il y a quelques cycles dans l'arbre; toutefois plus il y aura de cycles, plus l'aspect en sera dégradé. `layout-radial` demande un agent « racine » (*root*) qui doit être le noeud central placé en (0,0), puis elle arrange les noeuds qui lui sont connectés (à l'agent racine) en un motif concentrique. Les noeuds situés à un niveau du noeud racine sont arrangés selon un motif circulaire, le niveau suivant autour de ces noeuds est également arrangé selon un motif circulaire, et ainsi de suite. La primitive `layout-radial` tentera de tenir compte des graphes asymétriques et donnera plus de place aux branches qui sont les plus grandes. `layout-radial` peut aussi recevoir une race en entrée, ce qui permet d'utiliser une race de liens pour afficher un réseau plutôt qu'un autre.

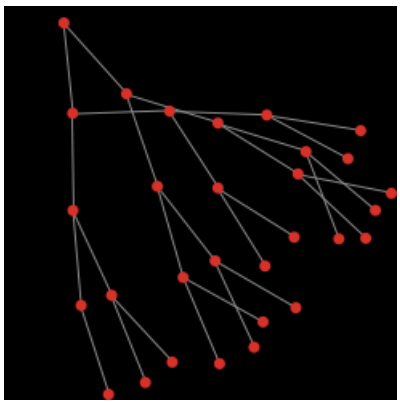


Quand on lui donne un ensemble de noeuds d'ancrage, `layout-tutte` place tous les autres noeuds au centre de masse (centre de gravité) des noeuds qui leur sont liés. L'ensemble des ancrages est automatiquement disposé en cercle de rayon défini par l'utilisateur et les autres noeuds convergeront à leur place (cela signifie bien entendu que vous devrez exécuter plusieurs fois cette primitive avant que l'arrangement des noeuds obtenu soit stable.



`layout-spring` et `__layout-magspring` sont presque semblables et sont très utiles pour de nombreux types de réseaux. L'inconvénient est qu'il sont relativement lents puisqu'ils demandent un grand nombre d'itérations pour converger. Dans les deux réseaux, les liens jouent le rôle de ressorts qui tirent les noeuds qu'ils relient les uns vers les autres alors que les noeuds se repoussent les uns les autres. Dans le ressort magnétique, il y a aussi un champ magnétique qui « tire » les noeuds dans une direction magnétique que vous choisissez. Les intensités des toutes ces forces sont contrôlées par les paramètres passés en entrée à ces primitives. Ces entrées auront toujours une valeur comprise entre 0 et 1. Gardez à l'esprit que même de très petites modifications peuvent déjà affecter l'apparence du réseau. Les ressorts ont aussi une longueur (en unités patch), toutefois, à cause de toutes les forces mises en jeu, les noeuds ne finissent pas exactement à cette distance les uns des autres. Le réseau de type ressort magnétique demande aussi une entrée booléenne, `bidirectional?`, qui indique si le ressort doit pousser dans les deux directions parallèlement au champ magnétique. Si `bidirectional?` est vrai (`true`) les éléments du réseau seront espacés plus uniformément.





**Exemples de code** : " Network Example ", " Network Import Example ", " Giant Component ", " Small Worlds ", " Referential Attachment "

## Ask-Concurrent

Dans les versions précédentes de NetLogo, la primitive `ask` activait les agents en parallèle (en concurrence) par défaut, c'est-à-dire que les différents agents obéissant aux ordres transmis par `ask` tentaient d'exécuter leurs tâches « en même temps ». Depuis NetLogo 4.0, `ask` est sériel, c'est-à-dire que les agents exécutent les commandes transmises par `ask` les uns après les autres.

Les informations suivantes décrivent le comportement de la commande `ask-concurrent` qui agit de la même manière que le faisait l'ancien `ask`.

`ask-concurrent` produit une concurrence simultanée par un mécanisme de « *turn-taking* » que l'on pourrait traduire par « chacun son tour ». Le premier agent prend un tour, puis l'agent suivant prend un tour, et ainsi de suite jusqu'à ce que tous les agents de l'ensemble d'agents concernés par la commande aient effectué leur tour. Puis l'on revient au premier agent. Ce mécanisme se poursuit jusqu'à ce que tous les agents aient exécuté toutes les commandes reçues.

Un « tour » d'agent se termine lorsque l'agent a effectué une action qui modifie l'état du monde, tel qu'un déplacement, la création d'une tortue ou encore la modification de la valeur d'une variable globale, de tortue, de patch ou de lien. (L'action sur une variable locale ne compte pas.)

Les commandes `forward` (`fd`) et `back` ( `bk`) sont traitées différemment. Quand elles sont utilisées dans une commande `ask-concurrent`, elles peuvent nécessiter plusieurs tours pour faire leur travail. Pendant son tour, une tortue ne peut se déplacer que d'un pas. Ainsi, avec la commande `fd 20` (avancer 20 pas), qui est équivalente à `repeat 20 [ fd 1 ]`, le tour de la tortue se termine après l'exécution de `fd 1` (elle a donc besoin de 20 tours pour faire tout son travail). Si la distance spécifiée n'est pas un nombre entier, la partie fractionnaire compte comme un pas et prend un tour complet. Par exemple, `fd 20.3` est équivalent à `repeat 20 [ fd 1 ] fd 0.3`.

La commande `jump` se fait toujours en un seul tour, quelle que soit la distance.

Pour comprendre la différence entre `ask` et `ask-concurrent`, considérez les deux commandes suivantes :

```
ask turtles [ fd 5 ]
ask-concurrent turtles [ fd 5 ]
```

Avec `ask`, la première tortue avance de dix pas, puis la deuxième tortue avance de dix pas, et ainsi de suite.

Avec `ask-concurrent`, toutes les tortues avancent d'un pas. Puis toutes les tortues avancent d'un deuxième pas, et ainsi de suite. Ce qui fait que cette dernière commande est équivalente à :

```
repeat 5 [ ask turtles [ fd 1 ] ]
```

**Exemple de code** : " Ask-Concurrent Example " montre la différence entre `ask` et `ask-concurrent`.

Le comportement de `ask-concurrent` ne peut pas toujours être imité aussi simplement en utilisant `ask` comme dans l'exemple ci-dessus. Considérez la commande :

```
ask-concurrent turtles [ fd random 10 ]
```

Afin d'obtenir le même comportement en utilisant `ask`, il faudrait écrire :

```
turtles-own [ pas ]
ask turtles [ set pas random 10 ]
while [any? turtles with [pas > 0]
[
  ask turtles with [pas > 0]
  [
    fd 1
    set pas pas – 1
  ]
]
```

Pour prolonger le « tour » d'un agent, utilisez la commande `without-interruption`. (Les blocs de commandes dans certaines commandes, telles que `create-turtles` et `hatch` sont « encadrées » par une commande `without-interruption` implicite.)

Notez que le comportement de `ask-concurrent` est complètement déterministe. Étant donnés le même code et les mêmes conditions initiales, il se passera toujours la même chose (si vous utilisez la même version de NetLogo et que vous lancez l'exécution du modèle avec la même valeur d'amorçage du générateur de nombres aléatoires. (Voir la section Les nombres aléatoires.)

De manière générale, nous vous recommandons d'écrire le code de votre modèle de manière à ce qu'il ne dépende pas des détails de fonctionnement de `ask-concurrent`. Nous ne donnons aucune garantie que la sémantique de cette commande restera la même dans les futures versions de NetLogo.

## Les attaches

L'**attache** (*tie*) connecte deux tortues de manière à ce que le mouvement de l'une des tortues influence l'emplacement et l'orientation de l'autre. L'attache étant une propriété des liens, il doit y avoir un lien entre deux tortues pour pouvoir créer une relation de type attache.

Quand la propriété `tie-mode` d'un lien a reçu la valeur `fixed` ou `free`, `end1` et `end2` sont attachés l'un à l'autre. Si le lien est orienté, `end1` est l'agent « racine » et `end2` est l'agent « feuille ». Cela signifie que quand `end1` se déplace (en utilisant `fd`, `jump`, `setxy`, etc.), `end2` se déplace aussi de la même distance et dans la même direction. Mais quand `end2` se déplace, `end1` n'en est pas affecté.

Si le lien est non-orienté, la relation d'attache est réciproque, ce qui signifie si l'une des deux tortues se déplace, l'autre tortue suit. Ainsi, chaque tortue peut être considérée comme la racine ou la feuille en fonction de quelle tortue se déplace. La tortue racine est toujours la tortue qui a commencé le mouvement.

Quand la tortue racine pivote à droite ou à gauche, le tortue feuille tourne autour de la tortue racine du même angle, comme si une barre rigide reliait les deux tortues. Quand `tie-mode` est réglé à `fixed`, l'orientation de la tortue feuille change du même angle. Si `tie-mode` est réglé à `free`, l'orientation de la tortue feuille n'est pas modifiée.

Le mode d'attache `tie-mode` d'un lien peut être réglé à `fixed` au moyen de la commande `tie` et désactivé (c'est-à-dire que les tortues ne sont plus attachées) avec la commande `untie`. Pour mettre le mode d'attache à `free`, vous devez utiliser `set tie-mode "free"`.

**Exemple de code** : " Tie System Example "

## Les fichiers source multiples

Le mot-clé `__includes` permet d'utiliser plusieurs fichiers source pour un seul modèle NetLogo.

Ce mot-clé commence par deux caractères soulignement pour indiquer que cette fonctionnalité est expérimentale et peut encore être modifiée dans de futures versions de NetLogo.

Quand vous ouvrez un modèle qui utilise le mot-clé `__includes`, ou si vous ajoutez ce mot-clé au début du code du modèle et pressez le bouton "Check", le menu "Includes" apparaît dans la barre d'outils. Ce menu vous permet alors de sélectionner les fichiers inclus dans le modèle.

Quand vous ouvrez des fichiers inclus, leur code apparaît dans des panneaux "Procedures" supplémentaires. Consultez le Guide de l'interface pour plus de détails.

Les fichiers source externes (`.nls`) peuvent contenir tout ce que vous pouvez mettre normalement dans un panneau "Procedures", c'est-à-dire des définitions de variables globales (`globals`), des variables de races (`breed`), des variables tortue particulières (`turtles-own`), des variables patch particulières ( `patches-own`) ou race particulières (`breeds-own`), des définitions de procédures, etc. Notez toutefois que ces déclarations partagent toutes le même espace de noms. Ce qui signifie que si vous déclarez une variable globale `ma-globale` dans le panneau "Procedures", vous ne pouvez pas aussi déclarer une variable (ou quoi que ce soit d'autre) portant le nom de `ma-globale` dans l'un des fichiers qui sont inclus dans le modèle. La variable `ma-globale` sera accessible par tous les fichiers inclus. Il en irait de même si `ma-globale` était déclarée dans l'un quelconque des fichiers inclus.

## La syntaxe

Cette section contient une terminologie technique qui pourra paraître « ésotérique » à bien des lecteurs.

### Les mots-clés

Les seuls et véritables mots-clés du langage sont `globals`, `breed`, `turtles-own`, `patches-own`, `to`, `to-report` et `end`, plus `extensions` et le mot-clé expérimental `__includes` (Les noms des primitives prédéfinies, qui ne peuvent pas être remplacés ou redéfinis, sont aussi en définitive des sortes de mots-clés.)

### Les identificateurs

Toutes les primitives, les noms des variables globales et des variables d'agents et les noms des procédures se partagent un espace de noms global où les minuscules ne sont pas distinctes des majuscules. Les noms locaux (ceux des variables spécifiées par `let` et les noms des arguments des procédures) ne doivent pas « faire de l'ombre » (être les mêmes) que ceux des noms globaux ou que ceux d'autres variables locales. Les identificateurs peuvent contenir des lettres, des chiffres et les caractères ASCII suivants :

```
[ . ? = * ! < > : # + / % $ _ ^ ' & - ]
```

Les caractères non-ASCII (surtout les lettres accentuées) ne sont pour le moment pas autorisés dans les identificateurs. (Nous sommes conscients qu'il s'agit d'une gêne pour les utilisateurs non-anglophones et avons planifié la résolution de ce problème pour une mise à jour future.)

Certains noms de primitives commencent par deux caractères soulignement pour indiquer que ces primitives sont expérimentales et donc fortement sujettes à modifications, voire susceptibles de disparaître dans une des versions futures de NetLogo.

Les identificateurs qui commencent par un point d'interrogation sont réservés.

## Les domaines de validité ou portée (scope)

Les domaines de validité (appelés aussi portées) dans NetLogo sont de type lexical. Les variables locales (y compris les arguments des procédures) sont accessibles dans le bloc de commandes dans lequel elles ont été déclarées. Elles ne sont par contre pas accessibles par les procédures appelées par ces commandes.

## Les commentaires

Le caractère point-virgule marque le début d'un commentaire, commentaire qui se termine automatiquement à la fin de la ligne. Il n'y a pas de syntaxe pour un commentaire multi-lignes.

## La structure d'un programme

Un programme est formé de déclarations optionnelles (`globals`, `breed`, `turtles-own`, `patches-own`, `<BREED>-own`) placées dans n'importe quel ordre, suivies de zéro à plusieurs définitions de procédures. Les diverses races peuvent être déclarées dans des déclarations `breed` séparées, les autres déclarations ne pouvant par contre n'apparaître qu'une seule fois.

Chaque définition de procédure doit commencer par `to` ou par `to-report` suivi du nom de la procédure et, en option, de la liste des arguments placée entre crochets. Chaque définition de procédure doit se terminer par `end` placé seul sur la dernière ligne de la procédure. Entre les deux (`to` et `end`) prennent place zéro ou plusieurs commandes.

## Les commandes et les reporters

Les commandes reçoivent zéro ou plusieurs entrées par l'intermédiaire de leurs arguments; les arguments sont des reporters qui à leur tour peuvent recevoir zéro ou plusieurs entrées. Aucune ponctuation ne sépare ni ne termine les commandes; aucune ponctuation ne sépare les arguments. Les identificateurs (tous les noms utilisés par le langage) doivent être séparés par des espaces, par des parenthèses ou encore par des crochets. (Ainsi, par exemple, `a+b` est un identificateur unique, alors que l'expression `a(b[c]d)e` contient cinq identificateurs.)

Toutes les commandes sont préfixées. Tous les reporters définis par l'utilisateur sont préfixés. La plupart des primitives reporters sont aussi préfixées, mais certaines (les opérateurs arithmétiques, les opérateurs booléens et quelques opérateurs d'ensembles d'agents (tels que `with` et `in-points`) sont infixés.

Toutes les commandes et les reporters, aussi bien primitives que définis par l'utilisateur, demandent par défaut un nombre d'entrées (arguments) prédéfini. (C'est pour cette raison que le langage peut être analysé syntaxiquement même s'il n'y a pas de ponctuation pour séparer ou terminer les commandes et/ou les arguments.) Certaines primitives sont « variadic », c'est-à-dire qu'elles peuvent optionnellement recevoir un nombre d'arguments différent de celui par défaut. Dans ce cas, il faut utiliser des parenthèses pour marquer cette particularité, comme dans `(list 1 2 3)` par exemple (puisque la primitive `list` ne demande par défaut que deux arguments). Les parenthèses sont aussi utilisées pour modifier l'ordre de priorité des opérateurs, par exemple `(1 + 2) * 3`, comme c'est le cas dans les autres langages de programmation.

Parfois, l'argument d'une primitive est un bloc de commandes (zéro ou plusieurs commandes placées entre crochets) ou un bloc de reporters (une seule expression reporter entre crochets). Les procédures définies par l'utilisateur ne peuvent pas recevoir de bloc de commandes ou de reporter en entrée.

Les priorités des opérateurs sont les suivantes, des plus hautes au plus basses :

- `with`, `at-points`, `in-radius`, `in-cone`
- (toutes les autres primitives et les procédures définies par l'utilisateur)
- `^`
- `*` , `/` , `mod`
- `+` , `-`
- `<` , `>` , `<=` , `>=`
- `=` , `!=`
- `and` , `or` , `xor`

## Comparaison avec d'autres Logos

Il n'existe pas de définition standard du langage Logo. Ce terme recouvre plutôt une famille de langages assez disparate ayant une origine commune, le langage Lisp. Nous croyons toutefois que NetLogo possède suffisamment de caractéristiques communes avec les autres dialectes Logo pour mériter aussi le nom de Logo. Il n'en reste pas moins que NetLogo diffère par certains aspects de la plupart des autres Logo. Les différences les plus importantes sont énumérées ci-dessous.

### Les différences de surface (visibles)

- La priorité des opérateurs mathématiques est différente. Les opérateurs mathématiques infixés (tels que `+` , `*` , etc.) ont une priorité inférieure à celle des reporters portant un nom. Par exemple, dans la plupart des Logo, si vous écrivez `sin x + 1`, cette expression est interprétée comme `sin (x + 1)`. Mais NetLogo l'interprète de la même manière que le font la plupart des autres langages, qui est aussi la manière dont la même expression est interprétée dans la notation mathématique standard, c'est-à-dire comme `(sin x) + 1`.
- Les reporters `and` et `or` sont des formes spéciales et non des fonctions ordinaires : elles « courcircuitent », c'est-à-dire qu'elles n'évaluent leur deuxième argument que si nécessaire (donc en fonction du résultat de l'évaluation du premier argument).
- Les procédures ne peuvent être définies que dans le panneau "Procedures", et non interactivement dans le Centre de commande.
- Les procédures reporters, c'est-à-dire les procédures qui retournent une valeur (appelées fonctions dans les autres langages de programmation), doivent être définie par `to-report` plutôt que par `to`. La commande servant à exporter (retourner) la valeur calculée par la procédure reporter est `report` et pas `output`.
- Lors de la définition de la procédure, les arguments (décrivant les valeurs passées en entrées) de la procédure doivent être entourés de crochets, par exemple `to square [ x ]`.
- Les noms des variables sont toujours utilisés sans ponctuation : il faut toujours écrire `foo`, jamais `:foo` ou `"foo`. (Afin que cela fonctionne, nous avons créé une forme spéciale `set` qui n'évalue pas sa première entrée, plutôt qu'une commande `make` qui demande un argument avec ponctuation.) Il en résulte que les procédures et les variables partagent le même espace de noms.

Les trois dernières différences sont illustrées dans les définitions des procédures suivantes :

| La plupart des Logo | NetLogo |
|---|---|
| `to square :x`<br>`output :x * :x`<br>`end` | `to-report square [x]`<br>`report x * x`<br>`end` |

### Des différences plus « profondes »

- ✔ En NetLogo, les portées (domaines de validité) sont définies lexicalement, et pas dynamiquement.
- ✔ NetLogo n'a pas de type de données « `word` » (ce que Lisp appelle des « `symbols` »). Nous pourrions éventuellement l'ajouter un jour, mais comme il n'est que rarement demandé, il est fort probable que sa présence ne soit pas souvent nécessaire pour la modélisation à base d'agents. Nous avons les chaînes. Dans la plupart des situations où les Logo traditionnels utilisent le type `word`, nous utilisons simplement des chaînes à la place. Par exemple, en Logo, vous pourriez écrire `[see spot run]` (une liste de mots), mais en NetLogo, vous devez écrire `"see spot run"` (une chaîne) ou `["see" "spot" "run"]` (une liste de chaînes).
- ✔ La commande NetLogo `run` fonctionne avec des chaînes, pas avec des listes (puisque nous n'avons pas le type de données `word`), et ne permet pas la définition ou la redéfinition des procédures.
- ✔ Les structures de contrôle telles que `if` et `while` sont des formes spéciales, pas des fonctions ordinaires. Vous ne pouvez pas définir vos propres formes spéciales, ce qui fait que vous ne pouvez pas non plus définir vos propres structures de contrôles. (La commande NetLogo `run` ne vous est ici d'aucune utilité.)
- ✔ Comme avec la plupart des Logo, les fonctions par valeurs ne sont pas supportées. La plupart des Logo offrent des fonctionnalités générales similaires, voire moins performantes, en permettant le passage et la manipulation de fragments de code sous forme de liste. Les capacités de NetLogo dans ce domaine sont actuellement limitées. Quelques-unes de nos formes spéciales utilisent le style "`templates`" de l'UCBLogo pour accomplir des choses semblables, par exemple `sort-by [length ?1 < length ?2] string-list`. Dans certaines circonstances, l'utilisation de `run` et de `runresult` est possible, mais contrairement à la plupart des Logo, ils opèrent sur des chaînes, pas sur des listes.

Bien entendu, le langage NetLogo offre encore beaucoup d'autres fonctionnalités que l'on ne trouve pas dans les autres Logo, les plus importantes étant les agents et les ensembles d'agents.

# Chapitre 13

# Guide de transition

De nombreux modèles créés avec des versions précédentes de NetLogo tournent encore avec NetLogo 4.0. Toutefois, certains modèles doivent subir quelques modifications pour pouvoir fonctionner avec cette nouvelle version. Si d'anciens modèles ne marchent plus avec NetLogo 4.0, ce chapitre peut vous apporter une aide précieuse pour faire les mises à jour nécessaires.

Ce à quoi vous devez faire attention dépend de l'âge des modèles à adapter. Plus la version de NetLogo avec laquelle le modèle a été développé et mis au point est ancienne, plus les choses auxquelles vous devrez faire attention sont nombreuses.

Ce chapitre ne dresse pas une liste exhaustive de toutes les modifications qui ont été faites pour aboutir à NetLogo 4.0. Il ne présente que les changements qui sont d'une certaine importance pour les utilisateurs. Consultez la section Quoi de neuf? pour une liste complète des modifications apportées aux versions successives de NetLogo.



## Depuis NetLogo 3.1

### Numérotation des agents

Avant NetLogo 4.0, le numéro d'identification (*who number*) d'une tortue morte (enregistré dans le variable tortue `who` de chaque tortue) pouvait être réaffecté à une tortue nouveau-né par la suite. Avec NetLogo 4.0, les numéros d'identification ne sont jamais réutilisés avant que la numérotation n'ait été réinitialisée à 0 par la commande `clear-all` ou par la commande `clear-turtles`. Cette modification du comportement de la numérotation peut empêcher le bon fonctionnement de quelques anciens modèles.

### Création de tortues : aléatoire ou « ordonnée »

NetLogo 4.0 offre deux commandes observateur différentes pour créer de nouvelles tortues, `create-turtles` (en abrégé `crt`) et `create-ordered-turtles` ( `cro` en abrégé).

`crt` donne aux nouvelles tortues des couleurs et des orientations (caps) aléatoires (nombres entiers). `cro` distribue les couleurs séquentiellement et donne aux tortues des caps répartis séquentiellement et équitablement distribués sur les 360 degrés du cercle, la première tortue pointant vers le haut (donc vers le Nord) avec un cap de 0 degré.

Avant NetLogo 4.0, la commande `crt` se comportait de la même manière que le fait maintenant la commande `cro`. Si le bon fonctionnement de votre ancien modèle dépend d'une distribution ordonnées des tortues à leur création, vous devez modifier le code de votre modèle et remplacer la (les) commande(s) `crt` par une (des) commande(s) `cro`.

Il n'est pas rare que les anciens modèles qui utilisent `crt` contiennent aussi des commandes supplémentaires pour distribuer aléatoirement les caps des nouvelles tortues, par exemple `rt random 360` ou `set heading random 360`. Ces commandes ne sont plus nécessaires quand elles sont utilisées avec des commandes `crt`.

## Additionner des chaînes et des listes

Avant NetLogo 4.0, l'opérateur `+` (addition) pouvait être utilisé aussi bien pour concaténer des chaînes de caractères que pour joindre des listes. Avec le NetLogo actuel, l'opérateur + ne fonctionne plus qu'avec des nombres. Pour concaténer des chaînes, il faut utiliser la primitive `word`, et pour joindre des listes, il faut se servir de la primitive `sentence`. Ces modifications ont été faites pour accélérer la vitesse d'exécution du code qui utilise l'opérateur +.

Ancien code :

```
print "Il y a " + count turtles + " tortue."
```

Nouveau code :

```
print (word "Il y a " count turtles " tortues.")
```

De même, si vous devez joindre des listes, utilisez `sentence`.

Cette modification ne se fait pas automatiquement lors de la conversion des anciens modèles. Les utilisateurs doivent changer leurs codes à la main.

Nous sommes bien conscients que cette modification est gênante pour les utilisateurs habitués à l'ancienne syntaxe. Nous avons procédé à ce changement pour des raisons d'efficacité et de consistance du code. Nous pouvons en effet implémenter un opérateur d'addition qui n'additionne que les nombres de manière bien plus efficace qu'un opérateur qui doit manipuler plusieurs types de données très différents. Et comme l'addition est une opération très courante, c'est la vitesse générale de NetLogo qui en bénéficie.

## Les primitives `-at`

L'observateur ne peut plus utiliser les primitives `patch-at`, `turtles-at` et `breeds-at`. Il faut les remplacer par les primitives `patch`, `turtles-on patch` et `BREEDS-on patch`. Notez que `patch` arrondit maintenant ses entrées (avant il n'acceptait que des nombres entiers en entrée).

## Les liens

NetLogo 3.1 avait la possibilité d'utiliser des liens pour connecter les tortues dans le but de construire des réseaux, des graphes et des figures géométriques. Ces liens étaient eux-mêmes des tortues.

Dans NetLogo 4.0, plutôt que d'être des tortues, les liens forment maintenant un quatrième type d'agents indépendant, à côté de l'observateur, des tortues et des patches. Les primitives qui utilisent des liens ne sont dorénavant plus considérées comme expérimentales, elles font maintenant pleinement parties du langage.

Les modèles qui utilisent les anciennes primitives de liens expérimentales basées sur les tortues devront êtres mis à jour pour utiliser à la place les agents de types lien. Les différences ne sont pas énormes, mais une mise à jour « à la main » est nécessaire.

Les liens sont discutés en détails dans la section Liens du Guide de la programmation et dans les entrées du Dictionnaire NetLogo relatives aux primitives dédiées aux liens. Consultez la section "Networks" de la Bibliothèque des modèles ("Models Library") pour des exemples de modèles qui utilisent les liens. Il y a aussi quelques exemples dans la section "Code Examples".

D'abord, vous devez éliminer toutes les races appelées `links` si vous n'utilisez qu'un seul type de liens car dans ce cas, vous n'avez pas du tout besoin d'utiliser des races. Si vous utilisez plusieurs types de liens, intéressez-vous aux commandes `undirected-link-breeds` et `directed-link-breeds`. Les commandes et les reporters qui contiennent le mot `links` dans leur nom (tels que `__create-links-with`, etc.) seront automatiquement convertis dans leurs nouvelles versions dépourvues des caractères de soulignement (`create-links-with`). Toutefois, les primitives qui utilisent un nom de race différents (tel que `edges`) ne sont pas converties. Vous devez enlever ces caractères de soulignement à la main, et à moins que vous ne déclariez une race de liens avec ce nom, vous devez aussi remplacer la désignation de la race par `links`.

Les primitives `remove-link(s)-with/from/to` n'existent plus. Il faut tuer le lien avec `die`. Par exemple :

```
ask turtle 0 [ __remove-links-with link-neighbors ]
```

devient

```
ask turtle 0 [ ask my-links [ die ] ]
```

Plusieurs commandes d'agencement (*layout*) des tortues ont des entrées légèrement différentes : les deux premières entrées sont en général un ensemble d'agents de type tortue et un ensemble d'agents de type lien qui seront utilisés pour constituer l'agencement. Pour les détails, consultez dans le dictionnaire les entrées des primitives `layout-spring`, `__layout-magspring`, `layout-radial` et `layout-tutte`

Vous devrez peut-être aussi réarranger la déclaration des variables `turtles-own` puisque les liens ne sont maintenant plus de véritables tortues. Toutes les variables qui concernent les liens doivent être déplacées dans un bloc `links-own`.

Puisque les liens ne sont plus des tortues, ils ne possèdent plus de variables tortues préprogrammées (bien que certaines variables lien soient les mêmes, telles que `color` et `label`). Si vous avez utilisés la position d'anciens liens (de type tortue), vous devez maintenant calculer le point placé au milieu du lien. Ce qui est relativement simple dans un monde non « enroulé ».

```
to-report link-xcor
  report mean [xcor] of both-ends
end

to-report link-ycor
  report mean [ycor] of both-ends
end
```

Bien qu'un peu plus compliquée dans un monde « enroulé », la conversion reste compréhensible :

```
to-report link-xcor
  let other-guy end2
  let x 0
  ask end1
  [
    hatch 1
    [
      face other-guy
      fd [distance other-guy] of myself / 2
      set x xcor
      die
    ]
  ]
  report x
end
```

Il en va de même avec `ycor`.

Si vous avez utilisé soit la taille, soit le cap de liens-tortues, vous pouvez utiliser à la place les primitives `link-length` et `link-heading`.

## Nouvelle syntaxe de "of"

Nous avons remplacé les trois constructions de langage `-of` (avec trait-d'union), `value-from` et `values-from` par la seule construction `of` (sans trait-d'union).

| ancienne | nouvelle |
|---|---|
| `color-of turtle 0` | `[color] of turtle 0` |
| `value-from turtle 0 [size * size]` | `[size * size] of turtle 0` |
| `mean values-from turtles [size]` | `mean [size] of turtles` |

Quand `of` est utilisé avec un seul agent, il retourne une seule valeur. Quand il est utilisé avec un ensemble d'agents, il retourne une liste de valeurs (rangées dans un ordre aléatoire puisque les agents d'un ensemble d'agents sont toujours rangés dans un ordre aléatoire).

Notez que quand vous ouvrez un ancien modèle avec la nouvelle version de NetLogo, les anciennes primitives `-of`, `value-from` et `values-from` sont automatiquement convertie pour utiliser `of`, mais certaines utilisations imbriquées de ces constructions sont trop compliquées pour le convertisseur automatique et doivent être adaptées à la main.

## Fonctionnement sériel de "ask"

La commande `ask` active maintenant ses agents de manière sérielle plutôt qu'en concurrence. En d'autres mots, les agents appelés sont activés un à la fois, les uns après les autres. Ce n'est qu'une fois qu'un agent a terminé toutes les tâches définies dans la commande `ask` que l'agent suivant peut de « mettre au travail ».

Notez que même l'ancienne commande `ask` n'a jamais vraiment été concurrente; nous simulions l'exécution concurrente des tâches en entrelaçant le travail des agents au moyen d'un mécanisme « chacun son tour » tel que décrit dans les NetLogo FAQ.

Nous avons procédé à cette modification de comportement car, selon notre expérience, les utilisateurs écrivaient souvent du code qui conduisaient leurs modèles à se comporter de manière inattendue en raison du comportement concurrentiel simulé, mais n'écrivaient que rarement des modèles qui profitaient pleinement de ce type de comportement. Les modèles qui présentent un comportement inattendu, voire bizarre, devraient normalement pouvoir être « ramenés à la raison » en ajoutant des commandes `without-interruption` aux bons endroits, mais il était difficile pour les utilisateurs de savoir si cette commande était vraiment indispensable et si oui, où la mettre.

Avec NetLogo 4.0, `without-interruption` n'est plus nécessaire à moins que votre modèle n'utilise la primitive `ask-concurrent` (ou qu'un bouton "forever" de tortue ou de patch ne contienne du code qui dépende de la concurrence simulée). Dans la plupart des modèles, toutes les utilisations de `without-interruption` peuvent être supprimées.

La concurrence simulée autrefois utilisée par `ask` est toujours réalisable de trois manières :

- ✔ Vous pouvez utiliser la primitive `ask-concurrent` à la place de `ask` pour obtenir l'ancienne concurrence simulée.
- ✔ Les commandes données directement aux tortues, aux patches et aux liens dans le Centre de commande ont une commande `ask-concurrent` implicite.
- ✔ Les boutons "forever" de tortues, patches et liens ont aussi une commande `ask-concurrent` implicite.

Notez toutefois que la commande `ask` elle-même est toujours sérielle, quel que soit le contexte dans lequel elle est utilisée.

Dans notre Bibliothèques des modèles, les modèles qui font usage de cette concurrence sont rares. Le modèle "Termites" en est un exemple remarquable car il utilise la concurrence d'un bouton tortue "forever".

## Compteur de cycles (*ticks*)

NetLogo a maintenant un compteur de cycles pour représenter l'écoulement du temps de la simulation.

Vous faites avancer le compteur de cycles par pas d'une unité de temps avec la commande `tick`. Si vous devez lire sa valeur, utilisez le reporter `ticks`. La commande `clear-all` remet le compteur de cycles à zéro, tout comme le fait aussi la commande `reset-ticks`.

Dans la plupart des modèles, la valeur du compteur cycle (le temps écoulé depuis sa mise en route ou sa dernière réinitialisation) est un nombre entier, mais si vous devez utiliser des incréments de temps plus petits, vous pouvez vous servir de la commande `tick-advance` pour faire avancer ce compteur de cycles de n'importe quelle valeur, y compris d'une valeur fractionnaire. Le modèle "Vector Fields" et les modèles du groupe "GasLab" de la Bibliothèque des modèles sont des exemples de modèles qui utilisent `tick-advance`.

La valeur du compteur de cycles est affichée dans la partie droite de la barre de titre de la Vue 2D (ou de la Vue 3D). (Vous pouvez utiliser la boîte de dialogue "Model Settings" appelée par le bouton "Settings… " de la barre d'outils du panneau "Interface" pour cacher ou afficher ce compteur ou pour remplacer son intitulé par défaut ("ticks:") par quelque chose d'autre.)

## Modes de mises à jour de la Vue

Par le passé, NetLogo essayait toujours de mettre la Vue à jour environ 20 fois par seconde. Nous appelons maintenant ce genre de mises à jour « continuous » (continu). Le plus gros problème qu'elle pose est que vous voulez en général que les mises à jour de la Vue se fassent entre les cycles de la simulation et non au milieu d'un cycle. C'est pourquoi nous avons ajouté un bouton-menu dans la barre d'outils de l'interface qui permet de choisir entre une mise à jour continue ("continuous") de style ancien et une mise à jour à la fin de chaque cycle ("on ticks"). Et vous pouvez supprimer toutes les mises à jour au moyen de la case à cocher "view updates". Vous pouvez aussi empêcher les mises à jour avec la commande `no-display` et les remettre en fonction avec la commande `display`.

NetLogo permet toujours les mises à jour continues. C'est d'ailleurs le réglage par défaut au démarrage du programme. Mais la plupart des modèles de la Bibliothèques des modèles utilisent maintenant les mises à jour commandées par les cycles. Avec les mises à jour basées sur les cycles, celles-ci n'ont lieu qu'au moment où le compteur de cycles avance. (La commande `display` peut être utilisée pour forcer des mises à jour supplémentaire; voir ci-dessous).

Les avantages des mises à jour basées sur les cycles telles que nous venons de les voir sont les suivants :

- ✔ Comportement des mises à jour cohérent et prévisible, qui ne varie pas d'un ordinateur à l'autre ou d'une simulation à l'autre.
- ✔ Les mises à jour intermédiaires peuvent « embrouiller » l'utilisateur de votre modèle en lui montrant des choses qu'il n'est pas supposé voir, ce qui peut le conduire à des conclusions erronées.
- ✔ Augmentation de la vitesse de la simulation. La mise à jour de la Vue prend du temps, donc si une mise à jour par cycle est suffisante, obliger le modèle à ne faire qu'une mise à jour par cycle accélérera la simulation du modèle.
- ✔ Plutôt que d'avoir une case à cocher "force view update" dans chaque bouton (comme dans NetLogo 3.1), il n'y a qu'un réglage à faire pour tout le modèle, et ce réglage peut aussi être fait par l'utilisateur du modèle.

- ✔ L'utilisation du curseur de vitesse "speed" pour ralentir un modèle ne fait qu'insérer des pauses entre les cycles. Ce qui fait qu'avec les mises à jour basées sur les cycles, les boutons `setup` ne sont plus influencés par le curseur vitesse. Ce qui était vraiment ennuyeux avec l'ancien curseur vitesse. (Mais cet inconvénient est toujours là pour les modèles qui utilisent les mises à jour continues).

Comme mentionné ci-dessus, la plupart des modèles de notre Bibliothèque des modèles utilisent maintenant les mises à jour basées sur les cycles.

Même pour les modèles dont les mises à jour sont normalement basées sur les cycles, il peut être utile de basculer en mode mises à jour continues de manière temporaire lors de la mise au point du modèle. Observer en détail ce qui se passe au cours d'un cycle plutôt que de ne voir que le résultat final du cycle peut être d'une aide non négligeable pour trouver la (ou les) cause(s) d'un dysfonctionnement du modèle.

Si vous modifiez votre modèle pour qu'il utilise les mises à jour basées sur les cycles, vous devez aussi ajouter la commande `tick` à votre code, sinon le Vue ne sera jamais mise à jour. (Notez toutefois que la Vue est encore toujours mise à jour quand un bouton se relève ou quand une commande passée dans le Centre de commande a fini son travail. Donc il n'y a pas de raison que la Vue reste figée indéfiniment).

### Comment faire pour qu'un modèle utilise les cycles et les mises à jour basées sur les cycles

Voici les étapes à suivre pour convertir votre modèle de manière à ce qu'il utilise les cycles et les mises à jour basées sur les cycles offerts par NetLogo 4.0 :

- ✔ Dans la partie droite de la barre d'outils du panneau "Interface", remplacez la commande "continuous" par la commande "on ticks" du bouton-menu qui se trouve sous de la case à cocher "view updates".
- ✔ Ajouter la commande `tick` à la fin ou vers la fin de votre procédure `go`. Dans les modèles de la Bibliothèque des modèles, nous avons toujours mis la commande `tick` après avoir fait bouger les agents mais avant toute commande de dessin des graphiques dans les traceurs de courbes. Ceci parce que les commandes de dessin de graphiques peuvent contenir quelque chose du genre `plotxy ticks ...` et que nous voulons que ce soit la nouvelle valeur du compteur de cycles qui soit utilisée, pas l'ancienne. Bien que la plupart des modèles ne fassent pas référence au compteur de cycles dans leurs commandes de dessin des graphiques, nous vous suggérons, pour des raisons de consistance et pour éviter des erreurs, de toujours mettre la commande `tick` avant les commandes destinées aux traceurs de courbes.

Certains modèles nécessiteront des modifications supplémentaires, à savoir :

- ✔ Si votre modèle possède déjà une variable globale `ticks`, `clock` ou `time`, supprimez-la. Utilisez à la place la commande `tick` et le reporter `ticks`. Si votre modèle utilise des incréments de temps fractionnaires, remplacez la commande `tick` par la commande `tick-advance`. Si vous avez un moniteur pour afficher la valeur de cette variable, supprimez-le car il y a maintenant un affichage du temps de la simulation dans la barre de titre de la Vue 2D (et aussi dans celle de la Vue 3D).
- ✔ La commande `clear-all` remet le compteur de cycles à zéro. Si vous n'utilisez pas cette commande dans la procédure `setup`, vous devrez faire appel à la primitive `reset-ticks` pour remettre le compteur de cycles à zéro.
- ✔ Si vous aviez utilisé les commandes `no-display` et `display` pour empêcher qu'une mise à jour de la vue ne se fasse au milieu de `go`, vous pouvez les supprimer.
- ✔ Si votre modèle à besoin de rafraîchir sa Vue sans faire avancer le compteur de cycles (exemples dans les modèles "Party" et "Dice Stalagmite", les modèles de réseau avec animation de l'agencement ou les modèles interagissant aux actions de la souris sur les boutons), utilisez la commande `display` pour forcer la mise à jour de la Vue de manière à ce que l'utilisateur voie tout de suite ce qui se passe.

## Le curseur vitesse

Les versions précédentes de NetLogo possédaient un curseur vitesse qui pouvait être utilisé pour ralentir la simulation des modèles de manière à ce que l'utilisateur puisse bien voir ce qui se passe.

Dans NetLogo 4.0, ce curseur peut aussi être utilisé pour accélérer la simulation des modèles. Il le fait en diminuant la fréquence de rafraîchissement de la Vue. La mise à jour de la Vue prend du temps, donc moins il y a de mises à jour, plus le modèle tourne rapidement.

La position par défaut du curseur se trouve au milieu de l'échelle, à la position notée "normal speed". Plus vous éloignez le curseur de sa position centrale, plus vous ralentissez ou accélérer la simulation.

À de très hautes vitesses, les mises à jours de la Vue deviennent peu fréquentes et il peut se passer plusieurs secondes entre deux mises à jours. On pourrait alors croire que la simulation tourne plus lentement puisque les mises à jour se font plus rares. Mais si vous observez le compteur de cycles ou d'autres indicateurs tels que les traceurs, vous verrez bien qu'en réalité la simulation se déroule beaucoup plus rapidement. Si ces mises à jour peu fréquentes vous déconcertent, ne déplacez pas le curseur si loin vers la droite.

Si la mise jour basée sur les cycles est activée, le ralentissement de la simulation ne provoquera pas une augmentation des mises à jour. NetLogo fait plutôt une pause après chaque cycle.

Si la mise à jour continue est activée, le ralentissement de la simulation signifie que les mises à jour se font de manière plus rapprochée. Si vous déplacez le curseur vitesse plus loin que la moitié de son chemin vers la gauche, la simulation se fait si lentement que vous pouvez voir les tortues se déplacer les unes après les autres ou décomposer leurs déplacements complexes en plusieurs segments de droite! Ce comportement a été introduit avec NetLogo 4.0. Dans les versions précédentes, aussi loin vers la gauche que vous déplaciiez le curseur, vous ne pouviez jamais voir les agents obéissant à une commande `ask` se déplacer les uns après les autres; tous les agents soumis cette commande paraissaient toujours se déplacer tous ensemble.

## Les nombre

NetLogo ne fait maintenant plus, en interne, de différence entre les nombres entiers et les nombres en virgule flottante. Avant, on avait par exemple :

```
observer> print 3
3
observer> print 3.0
3.0
observer> print 1 + 2
3
observer> print 1.5 + 1.5
3.0
observer> print 3 = 3.0
true
```

La dernière ligne montre que même si la distinction entre le nombre entier 3 et le nombre en virgule flottante 3.0 est maintenue, les deux nombres sont quand même considérés comme égaux.

Et maintenant :

```
observer> print 3
3
observer> print 3.0
3
observer> print 1 + 2
3
observer> print 1.5 + 1.5
3
observer> print 3 = 3.0
true
```

Nous pensons que bien peu de modèles seront pénalisés par cette modification.

Un avantage de ce changement est que, maintenant, NetLogo supporte un bien plus grand ensemble de nombres entiers. L'ancien ensemble allait de `-2'147'483'648` à `2'014'704'830'647` (environ +/- 2 milliards) alors que le nouvel ensemble va de `-9'007'199'254'740'992` à `+9'007'199'254'740'992` (environ +/- 10 millions de milliards).

## Construction des ensembles d'agents

NetLogo 3.1 (et quelques versions plus anciennes) contenaient des primitives appelées `turtles-from` and `patches-from` qui étaient occasionnellement utilisées pour construire des ensembles d'agents. Dans NetLogo 4.0, ces primitives ont été remplacées par de nouvelles primitives appelées `turtle-set` et `patch-set` qui sont bien plus souples et plus puissantes. (`link-set` existe aussi.) Consultez les entrées de ces primitives dans le Dictionnaire NetLogo. Les modèles qui utilisent les anciennes primitives `turtles-from` et `patches-from` doivent être modifiées à la main pour utiliser les nouvelles primitives.

## Couleurs RGB

En NetLogo 3.1, les couleurs RGB et HSB pouvaient être transformées plus ou moins fidèlement en couleurs NetLogo au moyen des primitives `rgb` et `hsb`. Ces dernières ont été renommées `approximate-rgb` et `approximate-hsb` et acceptent maintenant des entrées allant de 0-255 et plus seulement de 0-1.

Maintenant, NetLogo peut travailler avec la totalité du spectre des couleurs RGB, c'est pourquoi il n'est plus nécessaire d'utiliser ces procédures. Vous pouvez transmettre à n'importe quelle variable de couleur une liste formée des trois nombres désignant une couleur au format RGB, avec des valeurs allant de 0-255 pour chaque couleur de base et obtenir ainsi la couleur exacte. Voir la section Les couleurs du Guide de la programmation pour plus de détails.

## L'attache

Dans les versions précédentes, la primitive `__tie` (attache) n'était encore qu'une fonctionnalité expérimentale. Avec NetLogo 4.0, les liens ont reçu une variable `tie-mode` qui peut prendre (et donc recevoir) les valeurs `"none"`, `"free"` ou `"fixed"`. Dans sa version 4.0, `tie` est devenue une primitive exclusivement réservée aux liens. Ce qui veut dire que pour attacher la tortue 1 à la tortue 0, vous devez écrire :

```
ask turtle 0 [ create-link-to turtle 1 [ tie ] ]
```

Consultez la section L'attache du Guide de la programmation pour de plus amples informations.

## Les clients HubNet

L'interface de l'activité client de HubNet n'est plus stockée dans un fichier de modèle séparé. Pour importer un client d'un ancien modèle, aller dans le menu "File", sous-menu "Import", commande "Import HubNet Client Interface...". Puis, quand c'est demandé, importez depuis le panneau "Interface". Vous n'aurez plus besoin du modèle de client externe et vous n'aurez plus à le pointer quand vous installerez l'interface client. Donc,

```
hubnet-set-client-interface "COMPUTER" [ "my-client.nlogo" ]
```

devient :

```
hubnet-set-client-interface "COMPUTER" []
```

### Performance avec les listes

L'implémentation interne des listes a été changée, ce qui modifie les performances des certaines opérations effectuées sur ces listes. Voir la section Les performances des listes du Guide de la Programmation pour des informations détaillées concernant l'implémentation actuelle des listes.

Notez déjà que la primitive `fput` est plus rapide que la primitive `lput`, c'est pourquoi vous pouvez augmenter les performances de votre modèle simplement en remplaçant `lput` par `fput` puis en inversant éventuellement la liste obtenue. Si le problème des performances persiste, pensez à utiliser les primitives de l'extension Array & Table.

## Depuis NetLogo 3.0

### Les ensembles d'agents

Si votre modèle se comporte de manière étrange ou incorrecte, c'est peut-être à cause du fait que, depuis NetLogo 3.1, les agents d'un ensemble d'agents sont toujours rangés dans un ordre aléatoire, ordre qui change chaque fois qu'une opération est effectuée sur l'ensemble. Dans les versions antérieures de NetLogo, les agents des ensembles étaient toujours rangés dans le même ordre. Si votre code dépend d'agents rangés en ordre déterminé, il ne peut plus fonctionner correctement. La manière de résoudre ce problème afin que votre modèle puisse fonctionner avec des ensembles d'agents rangés aléatoirement dépend des détails de ce que fait votre code. Dans certaines situations, il peut s'avérer bénéfique d'utiliser les primitives `sort` ou `sort-by` pour convertir un ensemble d'agents (rangés aléatoirement) en une liste d'agents (rangés en ordre déterminé). Consultez la partie Les listes d'agents de la section Les listes du Guide de la Programmation.

### L'enroulement

Si vous voyez des bouts de formes de tortues apparaître sur le bord opposé du monde, c'est parce que NetLogo 3.0 permettait de désactiver un tel « enroulement » dans la Vue sans influencer le comportement du modèle. Depuis NetLogo 3.1, si vous ne voulez pas que la Vue soit « enroulante », vous devez faire en sorte que le monde lui-même ne s'enroule pas en utilisant les fonctionnalités nouvelles apportées par la topologie. Toutefois, procéder à ces modifications peut vous amener à apporter encore d'autres changements à votre modèle. Consultez la section La topologie du Guide de la programmation pour une présentation détaillée des méthodes à utiliser pour convertir le modèle afin qu'il puisse bénéficier de cette nouvelle fonctionnalité qu'est la topologie.

### Coordonnées de tortue aléatoires

De nombreux modèle créés avec NetLogo 3.0 (ou antérieur) utilisent le code `setxy random world-width random world-height` pour disperser aléatoirement les tortues, parfois `random` étant remplacé par `random-float`. Mais ceci ne fonctionne que si l'enroulement autour du monde est activé.

(Mais pourquoi cela? Parce que, quand l'enroulement est activé, vous pouvez donner aux coordonnées des tortues des valeurs qui vont au-delà des limites du monde et NetLogo « enroule » alors la tortue et la fait « revenir » par le bord opposé du monde. Mais dans les mondes qui ne sont pas « enroulants », donner à une tortue des coordonnées x ou y qui la placerait en dehors des limites de ce monde conduit à une erreur d'exécution. La possibilité d'avoir un monde « enroulant » a été donnée à NetLogo 3.1. Consultez la section La topologie du Guide de la programmation pour des informations supplémentaires.

Pour corriger votre modèle de manière à ce qu'il se comporte correctement quelle que soit la topologie du monde, utilisez plutôt l'une de ces deux commandes :

```
setxy random-xcor random-ycor
setxy random-pxcor random-pycor
```

Ces deux commandes sont un peu différentes. La première place la tortue sur un point du monde choisi au hasard. La seconde place la tortue au centre d'un patch choisi au hasard. Une manière encore plus concise de placer la tortue au centre d'un patch choisi aléatoirement est :

```
move-to one-of patches
```

# Chapitre 14

# Les applets

Les modèles qui, normalement, fonctionnent avec le programme NetLogo peuvent aussi être exécutés sous forme d'applets Java dans un navigateur web.

## Construire et afficher les applets

Vous pouvez créer des applets en sélectionnant l'option "Save As Applet" dans le menu "File". Si le modèle n'a pas encore été sauvegardé dans son état actuel, il vous sera demandé de le faire et il vous sera aussi demandé si vous voulez un fichier `HTML`.

Pour que les applets puissent fonctionner, il faut que le fichier `html`, le fichier de votre modèle et le fichier `NetLogoLite.jar` se trouvent tous dans le même répertoire. (Vous pouvez faire une copie du fichier `NetLogoLite.jar` se trouvant dans le répertoire où vous avez installé NetLogo.)

Les applets peuvent lire et écrire des fichiers sur le serveur web. Si votre applet à besoin de fichiers supplémentaires, fichiers dans lesquels il doit lire des données, des fichiers-images qu'il doit importer, etc., vous devez aussi transférer (*upload*) ces fichiers sur le serveur. Ces fichiers devront se trouver dans la même configuration de dossiers relative au modèle que sur votre ordinateur. Les applets ne peuvent pas lire des fichiers sur l'ordinateur de l'utilisateur, mais seulement sur le serveur web. Les applets ne peuvent pas non plus « naviguer » dans le serveur web ni dans l'ordinateur de l'utilisateur, ce qui signifie que les primitives `user-file` et `user-new-file` ne font rien (et donc ne servent à rien) dans un applet. Tous les fichiers nécessaires au fonctionnement de votre modèle, y compris le fichier du modèle lui-même et le fichier `NetLogoLite.jar` doivent pouvoir est lisible par l'utilisateur sur le même serveur web.

Certains systèmes d'exploitation permettent de tester les applets localement sur l'ordinateur avant de les télécharger sur un serveur web. Toutefois, cette manière de faire ne fonctionne pas avec tous les systèmes d'exploitation, c'est pourquoi, si ces tests sont impossibles à partir de votre disque dur, essayez de transférer le tout sur un serveur web.

Il n'est pas nécessaire d'inclure dans votre page web tout ce qui se trouve dans le fichier `html` créé par NetLogo. Si vous voulez, il vous suffit juste de prendre le code HTML commençant par `<applet>` et se terminant par `</applet>`, puis de le coller dans le fichier `HTML` de votre choix. Il est même possible de placer plusieurs couples de balises `<applet> </applet>` dans un même fichier `HTML`.

Si le fichier `NetLogoLite.jar` et votre modèle se trouvent dans des dossiers différents, vous devez modifier les lignes `archive=` et `value=` du code HTML pour pointer vers ces différents emplacements. (Par exemple, si vous avez plusieurs applets situés dans des répertoires différents sur le même serveur web, vous pourriez vouloir ne placer qu'une seule copie du fichier `NetLogoLite.jar` dans un endroit central et modifier les lignes `archive=` de tous vos fichiers `HTML` pour pointer vers cette copie centralisée. Cette manière de faire vous fera gagner de l'espace disque et fera gagner du temps de téléchargement à vos utilisateurs.)

# Les besoins en Java

## Obtenir la bonne version

La version actuelle de NetLogo demande que votre navigateur web supporte au moins la version Java 1.4.1. Voici comment obtenir la bonne version de Java :

- Si vous êtes sur Windows 98 ou plus récent, vous devez télécharger le plugin Java à l'adresse http://www.java.com/en/download/windows_manual.jsp.
- Si vous êtes sur Mac OS X, vous avez besoin de OS X 10.2.6 ou plus. Si vous êtes sur OS X 10.2, vous avez aussi besoin de Java 1.4.1 Update 1, que l'on peut obtenir par l'intermédiaire de la "Mise à jour de logiciels..." du menu "Pomme" d'OS X. Les version OS X 10.3 et suivantes ont déjà la bonne version de Java. Vous devez aussi utiliser un navigateur qui supporte Java 1.4. Internet Explorer pour Mac ne fonctionne pas, Safari oui.
- Si vous êtes sur Windows 95, MacOS 8 ou MacOS 9, l'exécution des modèles via le web n'est plus supportée. Mais vous pouvez encore toujours télécharger l'application NetLogo 1.3.1 et faire tourner les modèles compatibles (donc anciens) de cette manière.
- Si vous êtes sur Linux ou un autre Unix, vous aurez besoin de la version 1.4.1 ou supérieure du « Java Runtime Environment » de Sun. Il est disponible en téléchargement à l'adresse http://www.java.com/. Consultez la page d'accueil de votre navigateur pour voir comment installer le plugin Java.

Si vous êtes sûr d'avoir le bon navigateur et le bon plugin Java, mais que l'applet ne fonctionne toujours pas, regardez dans les préférences de votre navigateur et assurez-vous que Java est bien activé.

Le site web suivant peut se révéler utile pour voir quel Java vous avez et obtenir le cas échéant la bonne version : http://www.javatester.org/.

## Augmenter la mémoire disponible

Certains applets NetLogo peuvent demander plus de mémoire que ne peut normalement en fournir le navigateur. Cette situation peut survenir si vous avez un grand nombre d'agents. Sur Windows, vous pouvez accroître l'espace mémoire disponible ("*heap*") dans le panneau "Java Control Panel" des réglages de l'environnement Java.

Si vous utilisez un Mac, notez que Mac OS X 10.2 et 10.3 ont une limite relativement basse pour la mémoire réservée aux applets Java, plus précisément de 64 mégaoctets. À l'origine, Mac OS X 10.4 avait la même limite, mais une mise à jour récente de Java l'a élevée à 96 MO. Vous pouvez obtenir cette mise à jour de Java sur le site Apple, Software Update. Mac OS X 10.5 devrait normalement aussi avoir cette limite supérieure.

Si votre navigateur utilise le plugin livré avec le **Sun JDK** ou le **JRE**, alors les instructions pour démarrer le **Java Plug-In Control Panel** sont disponibles à cette adresse. Dans le panneau "Advanced" du "Control Panel" ajoutez `-Xmx1024M` dans le champ "Java Runtime Parameters.

# Les extensions

Plusieurs extensions peuvent être utilisées dans les applets, il suffit simplement de transférer le dossier contenant les extensions `.jar` dans le même répertoire que le modèle.

Les extensions qui ont besoin de `jar` externes supplémentaires ne fonctionnent pas avec les applets. Ce sont entre autres les extensions `sound` et `gogo`.

## Problèmes connus

- ✔ Les extensions qui ont besoin de `jars` externes supplémentaires ne fonctionnent pas avec les applets.
- ✔ La Vue 3D ne fonctionne pas avec les applets.
- ✔ Le générateur de code n'est pas utilisé dans les applets (ce qui signifie que les simulations tourneront un peu plus lentement).
- ✔ Les serveurs web qui retournent des messages d'erreur personnalisés peuvent générer des exceptions java. Voir les FAQ pour de plus amples informations.

# Chapitre 15

# Guide de l'Éditeur de formes

Les éditeurs de formes de tortues "Turtle Shapes Editor" et de formes de liens "Link Shape Editor" permettent de créer et de sauvegarder des formes de tortues et de liens. NetLogo utilise pour ces formes des dessins vectoriels complètement re-dimensionnables et visibles sous tous les angles, ce qui veut dire que vous pouvez créer de telles formes en combinant des formes géométriques simples, formes qui peuvent être affichées à l'écran dans n'importe quelle taille et avec n'importe quelle orientation.

## Pour commencer

Pour créer des formes, sélectionnez "Turtle Shapes Editor" (pour les formes de tortues) ou "Link Shapes Editor" (pour les formes de liens) dans le menu "Tools". La fenêtre qui s'ouvre contient la liste de toutes les formes actuellement chargées dans le modèle, liste commençant par les formes par défaut (voir figure ci-dessous) qui sont incluses automatiquement dans tout modèle. Les éditeurs de formes permettent d'éditer les formes existantes, d'en créer de nouvelles ou d'aller en chercher dans d'autres modèles. Vous pouvez aussi importer des formes de tortues pré-existantes qui se trouvent dans une bibliothèque.

### Importer des formes

Chaque nouveau modèle NetLogo possède, dès le départ, un petit ensemble de formes de base fréquemment utilisées. Bien d'autres formes de tortues sont à disposition dans une bibliothèque que l'on ouvre en cliquant le bouton "Import from library...". Ce bouton ouvre une boîte de dialogue dans laquelle vous pouvez sélectionner une ou plusieurs formes pour les importer dans votre modèle. À cet effet, sélectionnez la ou les formes désirées et pressez le bouton "Import".

De la même manière, vous pouvez utiliser le bouton "Import from model..." pour aller chercher des formes dans un autre modèle.

### Les formes par défaut

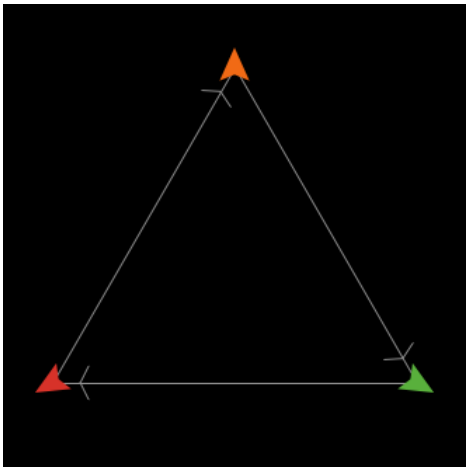
Voici les formes de tortue qui sont incluses par défaut dans chaque nouveau modèle NetLogo :

| | | | | | | |
|---|---|---|---|---|---|---|
| **Première ligne** | default | airplane | arrow | box | bug | butterfly |
| **Deuxième ligne** | car | circle | circle 2 | cow | cylinder | dot |
| **Troisième ligne** | face happy | face neutral | face sad | fish | flag | flower |
| **Quatrième ligne** | house | leaf | line | line half | pentagon | person |
| **Cinquième ligne** | plant | square | square 2 | star | target | tree |
| **Sixième ligne** | triangle | triangle 2 | truck | turtle | wheel | x |

### La bibliothèque des formes

Et voici les formes de la bibliothèque des formes (qui comprennent aussi toutes les formes par défaut) :

Par défaut, il n'y a, dans un modèle, qu'une seule forme de lien appelée `default`. Cette forme se réduit à un segment de droite (avec une simple flèche si le lien est un lien orienté).



## Créer et éditer des formes de tortues

Pressez le bouton "New" pour créer une nouvelle forme. Mais vous pouvez aussi sélectionner une forme existante et presser le bouton "Edit" pour la modifier.

### Les outils

Dans le coin supérieur gauche de la fenêtre d'édition se trouve un groupe d'outils de dessin. La flèche est l'outil de sélection qui permet de sélectionner tout élément déjà dessiné.

Pour dessiner un nouvel élément, utilisez l'un des sept autres outils, à savoir :

- L'outil **ligne** (line) pour dessiner des segments de droite.
- Les outils **cercle** (circle), **carré** (square) et **polygone** présents en deux versions : figures pleines (à gauche) et contours (à droite).

Quand vous utilisez l'outil **polygone**, cliquez avec la souris pour placer le premier sommet puis cliquez plus loin pour ajouter un nouveau sommet donc un nouveau segment (et ainsi de suite jusqu'à l'obtention de la forme désirée). Double-cliquez pour placer le dernier sommet et terminer la figure. Si vous n'avez pas double-cliqué sur le dernier point, la figure se referme automatiquement.

L'élément qui vient d'être dessiné est automatiquement sélectionné, vous pouvez donc le déplacer, le supprimer, changer sa forme, etc., si vous le désirez :

- Pour le déplacer, tirez-le avec la souris.
- Pour le supprimer, pressez le bouton "Delete".
- Pour modifier sa forme, tirez avec la souris les petites « poignées » qui apparaissent sur ses sommets (seulement quand l'élément est sélectionné).
- Pour changer sa couleur, cliquez sur une autre couleur dans la palette des couleurs.

## Les prévisualisations

Pendant que vous dessinez la forme, vous la voyez aussi en cinq exemplaires réduits dans les cinq zones de prévisualisation placées en dessous de la zone d'édition. Ces zones montrent la forme telle qu'elle pourrait apparaître dans un modèle, y compris à quoi elle ressemble quand elle pivote sur elle-même. Le nombre situé sous chaque zone de prévisualisation correspond à la taille de cette zone en pixels. Quand vous éditez la vue (bouton "Settings..." du panneau "Interface"), la taille des patches est aussi mesurée en pixels. Ainsi, par exemple, la zone de prévisualisation 20 montre à quoi ressemblera une tortue (de taille 1, autrement dit avec sa taille par défaut) dans une vue dont les patches ont des dimensions de 20 x 20 pixels.

Le pivotement peut être désactivée (case à cocher "Rotatable") si vous voulez que votre forme ait toujours la même orientation dans la vue, et ceci quel que soit le cap de la tortue.

## La superposition des formes

Les nouveaux éléments dessinés sont ajoutés au-dessus des éléments existants. Vous pouvez changer l'ordre d'empilement en sélectionnant un élément puis en utilisant les boutons "Bring to front" (Mettre à l'avant-plan) et "Send to back" (Mettre à l'arrière-plan).

## Annuler

À n'importe quel moment, vous pouvez utiliser le bouton "Undo" (Défaire) pour annuler l'action d'édition que vous venez de faire.

## Les couleurs

Les éléments dont la couleur correspond à la couleur affichée dans le bouton-menu "Color that changes" (couleur qui change) verront cette couleur remplacée par la couleur spécifiée dans la variable `color` de chaque tortue du modèle (la couleur qui change par défaut est le gris). Les éléments ayant une autre couleur ne changent pas de couleur. Vous pouvez par exemple créer des voitures qui ont toutes et toujours des phares jaunes et des roues noires mais des carrosseries de différentes couleurs.

## Les autres boutons

Les boutons "Rotate Left" (pivoter à gauche) et "Rotate Right" (pivoter à droite) font pivoter les éléments de 90 degrés. Les boutons "Flip Horizontal" (basculer horizontalement) et "Flip Vertical" (basculer verticalement) font tourner l'élément (par symétrie axiale) autour de son axe vertical ou de son axe horizontal.

Ces quatre boutons font pivoter ou basculer la forme entière, à moins qu'un élément soit sélectionné; dans ce cas, seul l'élément sélectionné est affecté.

Combinée avec "Duplicate" (dupliquer), l'utilisation de ces boutons est particulièrement intéressante si vous voulez créer des formes symétriques. Par exemple, pour dessiner un papillon, vous pouvez dessiner son aile gauche avec l'outil polygone, dupliquer cette aile gauche avec le bouton "Duplicate" puis faire tourner le duplicata autour de l'axe vertical avec le bouton "Flip Horizontal" pour mettre l'aile droite en place.

## La conception des formes

Il est tentant de dessiner des formes intéressantes et compliquées, mais souvenez-vous que, dans la plupart des modèles, la taille des patches est si petite que vous serez incapables d'apercevoir la plupart des détails. Ce sont les formes simples, grossière, de type icône, qui sont généralement les meilleures.

### Sauvegarder une forme

Quand une forme est terminée, donnez-lui un nom (qui n'a pas encore été utilisé) et pressez le bouton "OK" placé dans le coin inférieur droit de la fenêtre d'édition. La forme et son nom font maintenant partie de la liste des formes qui comprend aussi la forme « par défaut ».

## Créer et éditer des formes de liens

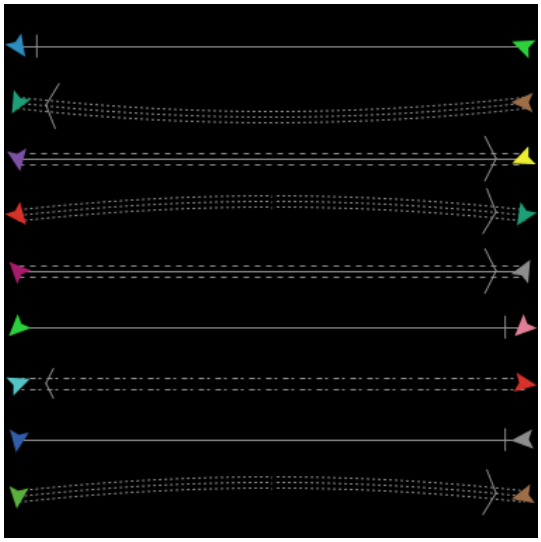
La gestion des formes de liens est très semblable à la gestion des formes de tortues. Ainsi, vous pouvez créer une nouvelle forme en pressant le bouton "New" ou éditer une forme existante en pressant le bouton "Edit". Quand l'édition de la forme est terminée, pressez le bouton "OK" si vous voulez la conserver.

### Modifier les propriétés d'une forme de lien

Vous avez la possibilité de modifier plusieurs propriétés pour chaque forme de lien. Ce sont :

- Le **nom**, "name:" — les formes de liens peuvent avoir les mêmes noms que les formes de tortues, mais le nom doit être unique parmi les forme de liens.
- L'**indicateur de direction**, "direction indicator:" — l'indicateur de direction (la petite flèche sur les liens orientés) est une forme de même type que les formes de tortues que vous pouvez éditer dans le même éditeur en pressant le bouton "Edit".
- La **courbure**, "curviness:" — est la grandeur de la « déviation par rapport à la droite » d'un lien exprimée en patches (c'est particulièrement utile si vous avez des liens orientés dans les deux directions, car cette courbure permet de les distinguer l'un de l'autre).
- Le **nombre de lignes** — Chaque forme de lien peut avoir 1, 2 ou 3 lignes, nombre que vous spécifiez en sélectionnant le motif de la ligne au moyen des boutons-menus "left line" (ligne de gauche), "middle line" (ligne médiane) et "right line" (ligne de droite).
- Les **motifs des lignes** — Les boutons-menus offrent plusieurs types de lignes pointillées, elles n'ont donc pas l'obligation d'être toutes des lignes continues.

Voici quelques formes de liens ayant des propriétés différentes :



## Utiliser les formes dans un modèle

Dans le code d'un modèle ou dans le Centre de commandes, vous pouvez utiliser n'importe quelle forme se trouvant dans le modèle (étant entendu que seules les tortues peuvent avoir des formes de tortues et que seuls les liens peuvent avoir des formes de liens). Supposons par exemple que vous vouliez créer 50 tortues ayant la forme de papillons (*butterfly*). Partant du fait qu'il y a une forme de tortue appelée `butterfly` dans le modèle (ce qui est le cas puisqu'il s'agit d'une des formes de base toujours présentes), donnez la commande suivante à l'observateur `observer>` dans le Centre de commandes :

```
observer> crt 50
```

Donnez ensuite ces commandes aux tortues (`turtles>`) pour les disperser puis changer leur forme :

```
turtles> fd random 15
turtles> set shape "butterfly"
```

Voila! Nous avons des papillons! Notez l'utilisation des guillemets doubles autour du nom la forme car les noms des formes sont des chaînes de caractères.

De manière identique, vous pouvez spécifier une autre forme pour les liens. Supposons qu'une forme de lien `"route"` se trouve dans le modèle, écrivez les commandes suivantes :

```
observer> crt 5 [ create-links-with other turtles ]
turtles> fd 5
links> set shape "route"
```

La commande `set-default-shape` est aussi utile pour assigner des formes aux tortues et aux liens.

# Chapitre 16

# Guide de BehaviorSpace

Ce chapitre comprend trois parties :

- **Qu'est-ce que BehaviorSpace?** : une description générale de l'outil, y compris les idées et les principes sous-jacents.
- **Comment cela fonctionne** : vous montre comment fonctionne l'outil et met en évidence ses fonctionnalités les plus utilisées.
- **Utilisation avancée** : comment utiliser BehaviorSpace à partir de la Ligne de commandes, ou à partir de votre propre code Java.

## Qu'est-ce que BehaviorSpace?

BehaviorSpace est un outil logiciel intégré à NetLogo qui doit vous permettre d'expérimenter avec les modèles. Il fait tourner un modèle plusieurs fois de suite en faisant varier systématiquement les réglages du modèle et en enregistrant les résultats de chaque simulation. Ce procédé est parfois appelé « balayages des paramètres » («*parameter sweeping*»). Il vous permet d'explorer l'« espace » des comportements possibles du modèle et de déterminer quelles sont les combinaisons de réglages qui aboutissent à des comportements intéressants.

### Pourquoi BehaviorSpace?

La nécessité de ce type d'expérimentations est mise en évidence par les observations suivantes. Les modèles ont souvent de nombreux paramètres réglables, chacun pouvant prendre toute une série de valeurs. Ensemble, ils forment ce qui, en mathématiques, est appelé un espace de paramètres pour le modèle, espace dont les dimensions correspondent au nombre de paramètres réglables et dans lequel chaque point est une combinaison de valeurs. Faire tourner un modèle avec différents réglages (et parfois même avec les mêmes réglages) peut conduire à des comportements radicalement différents du système qui a été modélisé. Ainsi, comment allez-vous savoir quelle configuration de valeurs particulière, ou quels types de configurations, produit le type de comportement qui vous intéresse? Cela revient à la question de savoir avec quel espace de paramètres multi-dimensionnel, parmi tous ceux possibles, votre modèle se comporte le mieux.

Par exemple, supposons que vous vouliez une synchronisation rapide des lucioles dans le modèle "Fireflies". Ce modèle possède quatre curseurs, "numbers", "cycle-length", "flash-length" et "flashes-to-reset" qui ont respectivement 2000, 100, 10 et 3 valeurs possibles. Ce qui signifie qu'il y a 2000 * 100 * 10 * 3 = 600'000 combinaisons de valeurs possibles pour ces curseurs! Essayer toutes ces combinaisons les unes après les autres n'est pas la manière la plus efficace de déterminer laquelle aboutit à la synchronisation la plus rapide.

BehaviorSpace offre un bien meilleur moyen de résoudre ce problème. Si vous spécifiez un sous-ensemble de valeurs possibles pour chaque curseur, il fera tourner le modèle avec toutes les combinaisons possibles de ces valeurs et, durant chaque simulation, enregistrera les résultats. De cette manière, il échantillonne l'espace des paramètres du modèle, non pas de manière exhaustive, mais suffisamment pour que vous puissiez voir les relations qu'il y a entre les différents curseurs et le comportement du système. Une fois toutes les simulations terminées, il génère et enregistre un ensemble de données que vous pouvez ouvrir dans un autre programme tel qu'un tableur, une base de données ou un programme de visualisation scientifique pour l'explorer à votre guise.

En permettant l'exploration de la totalité de l'espace des comportements qu'un modèle peut avoir, BehaviorSpace peut aussi être un puissant assistant au développeur de modèles.

Nous appellerons :

- **simulation** : *une seule* exécution (*run*) du modèle avec des paramètres initiaux définis.
- **expérimentation** : une succession de simulations (donc d'exécutions du modèle) avec, en général, des paramètres initiaux différents pour chaque simulation.
- **description d'expérimentation** : ensemble des réglages permettant de faire varier les paramètres initiaux au cours de l'expérimentation. Ce sont ces descriptions d'expérimentations que l'interface graphique de BehaviorSpace permet de réaliser facilement.

### Note historique

Les anciennes version de NetLogo (avant la 2.0) comprenaient une ancienne version de l'outil BehaviorSpace, version qui était passablement différente de l'actuelle. Elle n'était pas aussi souple dans les types d'expérimentations qu'elle permettait de mener. Mais elle offrait des possibilités d'afficher et d'analyser les résultats des expérimentations, possibilités qui font défaut dans la version actuelle. Avec la version actuelle, il est admis que vous utiliserez d'autres logiciels pour analyser les résultats. Nous espérons réimplanter les capacités d'affichage et d'analyses dans une future version de BehaviorSpace.

## Comment cela fonctionne

Avant d'utiliser BehaviorSpace, ouvrez votre modèle puis sélectionnez "BehaviorSpace" dans le menu "Tools" de NetLogo.

### Gérer les réglages de l'expérimentation

La boîte de dialogue que la commande précédente vient d'ouvrir permet de créer, d'éditer, de dupliquer, de supprimer et de lancer les descriptions d'expérimentations. Les expérimentations sont listées d'après leurs noms et leurs particularités.

Les descriptions des expérimentations sont considérées comme faisant partie du modèle et sont sauvegardées en tant que partie du modèle.

Pour créer une nouvelle expérimentation, pressez le bouton "New".

## Créer une nouvelle expérimentation

La boîte de dialogue "Experiment" qui s'affiche permet de spécifier les valeurs des paramètres décrits ci-après. Notez que vous ne devez pas toujours tout spécifier, certains champs pouvant être laissés vides ou avec leurs valeurs par défaut, en fonction de vos besoins.

Le champ **Experiment name:** permet de nommer l'expérimentation. Si le modèle fait l'objet de plusieurs expérimentations, leur donner des nom différents vous permet d'y voir plus clair.

Le champ à défilement **Vary variables as follows:** (faire varier les variables de la manière suivante) est l'endroit où vous spécifiez quels sont les paramètres à faire varier et quelles valeurs ces paramètres peuvent prendre. Ces variables peuvent être des curseurs (sliders), des commutateurs (switches), des sélecteurs (choosers), ainsi que toute variable globale définie dans le modèle.

Les variables de ce champ peuvent aussi comprendre les paramètres du modèle que sont `max-pxcor`, `min-pxcor`, `max-pycor` et `min-pycor`, `world-width`, `world-height` et `random-seed`. Ce ne sont, à proprement parler, pas des variables, mais BehaviorSpace vous permet de faire varier leurs valeurs comme si elles en étaient. Modifier les dimensions du monde permet d'explorer les effets de la taille du monde NetLogo sur votre modèle. Toutefois, la spécification de valeurs pour `world-width` et `world-height` ne définit pas nécessairement les limites du monde, car la manière dont ces limites sont déplacées dépend de l'emplacement de l'origine du monde. Si l'origine des coordonnées est au centre du monde, BehaviorSpace la laisse au centre du monde de manière à ce que les valeurs `world-width` ou `world-height` restent impaires. Si l'une des frontières a la coordonnée zéro, cette frontière restera à zéro et c'est la frontière opposée qui sera déplacée. Par exemple, si vous commencez avec un monde dont `min-pxcor = 0` et `max-pxcor = 10` et variez `world-width` de la manière suivante :

```
["world-width" [11 1 14]]
```

`min-pxcor` restera à zéro et `max-pxcor` prendra les valeurs 11, 12 et 13 pour chacune des simulations. Si aucune de ces conditions n'est vraie, autrement dit si l'origine n'est pas centrée ou si elle n'est pas sur un des bords du monde, vous ne pouvez pas faire varier `world-height` ou `world-width` directement, vous devez faire varier `max-pxcor`, `max-pycor`, `min-pxcor` et `min-pycor` à la place.

Faire varier la valeur transmise à `random-seed` permet de répéter les simulations en utilisant une amorce connue pour le générateur de nombres aléatoires de NetLogo. Notez que vous avez aussi la possibilité d'utiliser la commande `random-seed` dans les commandes que vous spécifiez pour l'expérimentation. Pour plus d'informations sur l'amorçage du générateur de nombres aléatoires, voyez la section Nombres aléatoires du Guide de la Programmation.

Pour une variable donnée, vous pouvez spécifier les valeurs à prendre soit en les listant, soit en spécifiant que vous voulez tester toutes les valeurs d'un intervalle donné correspondant à un certain incrément. Par exemple, pour donner à un curseur nommé `number` toutes les valeurs multiples de 50 entre 100 et 1000, vous devez écrire :

```
["number" [100 50 1000]]
```

Ou, pour lui donner seulement les valeurs 100, 200, 400, and 800, vous devez écrire :

```
["number" 100 200 400 800]
```

Ici, faites attention avec les crochets. Notez qu'il y a moins de crochets dans le second exemple. Ajouter ou non cette paire de crochets supplémentaires est votre manière de dire à BehaviorSpace si vous lui donnez une liste de valeurs individuelles ou si vous lui spécifiez un intervalle avec un incrément.

Notez également que les doubles guillemets anglais sont obligatoires autour des noms des variables.

Vous pouvez faire varier autant de paramètres que vous voulez, y compris un seul ou même aucun. Tout paramètre que vous ne faites pas varier garde sa valeur actuelle. Ne faire varier aucun paramètre est utile si vous voulez simplement faire plusieurs simulations de suite avec les réglages actuels.

L'ordre dans lequel les variables apparaissent dans le champ déroulant "Vary variables as follows :" détermine l'ordre dans lequel les différentes valeurs seront utilisées au cours de l'expérimentation. Toutes les valeurs d'une variable située plus bas dans la liste seront utilisées avant de passer à la valeur suivante d'une variable située au-dessus. Ainsi, par exemple, si vous faites varier x et y de 1 à 3, l'ordre dans lequel le modèle les utilisera dans les simulations successives sera : x=1 y=1, x=1 y=2, x=1 y=3, x=2 y=1, et ainsi de suite.

Le champ **Repetitions:** Parfois, le comportement du modèle peut passablement varier d'une simulation à l'autre même si les réglages des paramètres ne changent pas. Si vous voulez faire plusieurs simulations pour chaque combinaison de réglages de paramètres, entrez ici un nombre supérieur à 1 (le nombre de répétitions de la simulation).

Le champ **Measure runs using these reporters:** est l'endroit où vous spécifiez quelles données vous voulez récolter à chaque simulation. Par exemple, si vous voulez enregistrer comment la population de tortues augmente ou diminue au cours de chaque simulation, vous devez écrire :

```
count turtles
```

Vous pouvez spécifier un reporter, ou plusieurs, ou même aucun. Si vous en spécifiez plusieurs, chacun doit se trouver seul sur sa ligne, par exemple :

```
count frogs
count mice
count birds
```

Si vous ne spécifiez aucun reporter, la simulation se fait quand même. Cette manière de faire est utile si vous voulez enregistrer vous-même les résultats, par exemple avec la commande `export-world`.

Le champ **Measure runs at every step:** (effectuer les mesures à chaque pas). Normalement, NetLogo mesure les paramètres des simulations à chaque pas en fonction des reporters que vous avez spécifiés dans le champ précédent. Si vous faites de très longues simulations, il est possibles que vous ne vouliez pas toutes les données récoltées. Désactiver cette case pour ne récolter que les données obtenues à la fin de chaque simulation.

Le champs **Setup commands:** contient les commandes qui seront exécutées au début de chaque simulation. Vous y entrerez typiquement le nom d'une procédure qui initialise le modèle, en général la procédure nommée `setup`. Mais il est aussi possible d'y mettre d'autres commandes.

Le champ **Go commands:** contient les commandes qui sont exécutées en boucle (encore et encore) pour faire avancer la simulation au pas suivant. Ce sera typiquement le nom d'une procédure telle que `go`, mais vous pouvez y mettre toutes les commandes que vous voulez.

Le champ **Stop condition:** (condition d'arrêt) permet de faire varier la durée des simulations, chaque simulation se terminant quand une condition devient vraie. Par exemple, supposons que vous voulez que chaque simulation tourne jusqu'à ce qu'il n'y ait plus de tortues. Vous devez alors écrire :

```
not any? turtles
```

Si vous voulez que la durée de chaque simulation soit toujours la même, laissez simplement ce champ vide.

La simulation peut aussi s'arrêter parce que les commandes du champ "Go commands:" utilisent la commande `stop` de la même manière qu'elle peut être utilisée pour arrêter l'action d'un bouton "forever" (« pour-toujours »). La commande `stop` peut être directement utilisée dans les commandes du champs "Go commands" ou placée dans une procédure appelée directement par l'une des commandes de ce champ. (L'idée est que la même procédure `go` puisse fonctionner aussi bien dans un bouton que dans une expérimentation BehaviorSpace.) Notez que le pas de simulation dans lequel la commande `stop` est appelée et utilisée est considéré comme n'ayant pas abouti. Il s'ensuit qu'aucun résultat issu de ce pas ne sera enregistré. C'est pourquoi le test d'arrêt devrait toujours être fait au début de la commande go ou de la procédure et non à sa fin.

Le champ **Final commands:** permet de donner des commandes qui ne devront être exécutées qu'une seule fois, à la fin de la simulation. En général, ce champ est laissé vide, mais vous pourriez l'utiliser pour appeler la commande `export-world` ou pour enregistrer les résultats de la simulation d'une autre manière.

Le champ **Time limit:** permet de fixer une durée maximale à ne pas dépasser pour chaque simulation. (L'unité de temps est le pas (*tick*) de simulation). Si vous ne voulez pas fixer ce temps limite, mais désirez que la durée des simulations soit plutôt contrôlée par une condition d'arrêt, entrez 0 dans ce champ.

## Mener une expérimentation

Une fois les réglages de votre expérimentation terminés, pressez le bouton "OK" puis le bouton "Run". Une boîte de dialogues vous demandera alors de sélectionner les formats dans lesquels les données de l'expérimentation seront sauvegardées. Les données sont récoltées pour chaque intervalle, c'est-à-dire pour chaque simulation ou pas de simulation en fonction des réglages de l'option "Measure runs at every step".

Le format "Table" place les données de chaque série de mesures dans une ligne, avec chaque mesure dans sa propre colonne. Les données de la table sont écrites dans le fichier de sortie à la fin de chaque simulation. Il en résulte que les données de chaque simulation sont disposées verticalement les unes à la suite des autres. Ce format est surtout destiné a un traitement automatique des données suite à l'importation du fichier obtenu dans une base de données ou un logiciel de statistique.

Le format "Spreadsheet" (feuille de calcul) calcule et enregistre les valeurs minimale, moyenne, maximale et finale de chaque paramètre mesuré (*reporter*), chacun d'eux occupant une ligne. En-dessous, les valeurs mesurées à chaque pas de la simulation sont disposées en ligne (un pas par ligne), chaque valeur occupant la colonne de son reporter. Les données de chaque simulation sont disposées les unes à côté des autres. Les données dans le format "Spreadsheet" sont plus facilement lisibles et interprétables par le cerveau humain que les données au format "Table", surtout si elles ont été importée dans un tableur.

Notez toutefois que les données au format "Spreadsheet" ne sont pas sauvegardées dans le fichier des résultats avant la fin de l'expérimentation (contrairement à celles au format "Table"). Du fait que ces données sont conservées en mémoire jusqu'à la fin du travail, de très longues expérimentations peuvent conduire à un débordement de mémoire. Et tout ce qui peut interrompre l'expérimentation en cours, comme une erreur d'exécution, un débordement de mémoire, un crash du système ou une coupure de l'alimentation conduira a une perte irrémédiable de toutes les données recueillies. Pour les longues expérimentations, vous aurez tout intérêt a utiliser conjointement les formats "Spreadsheet" et "Table" de manière à ce que si quelque chose de fâcheux survient, il vous restera au moins une table des résultats partiels.

Après sélection du format des données en sortie, BehaviorSpace vous demandera le nom du fichier dans lequel sauvegarder les résultats. Le nom par défaut se termine en `.csv`. Vous pouvez changer ce nom si vous le désirez, mais vous devez conserver l'extension `.csv` qui indique que le fichier est au format CSV pour **C**omma **S**eparated **V**alues (valeurs séparées par des virgules). C'est un format de données de type texte lisible aussi bien par n'importe quel éditeur de texte que par la majorité des tableurs et des gestionnaires de bases de données.

S'affiche alors la boîte de dialogues intitulée "Running Experiment". Cette fenêtre vous tient au courant du déroulement de l'expérimentation en indiquant combien de simulations ont déjà été faites et combien de temps s'est déjà écoulé depuis de début de l'expérimentation. Si vous aviez spécifié un ou des reporters pour mesurer des données fournies par les simulations et si vous aviez laissé la coche de sélection dans la case "Measure runs at every step", vous verrez un graphique affichant l'évolution de ces valeurs au cours de chaque simulation.

Vous pouvez également observer le déroulement de la simulation dans la fenêtre NetLogo principale. (Si la fenêtre "Running Experiment" cache une partie de la fenêtre principale, il suffit de la mettre ailleurs sur l'écran.) La Vue et les traceurs de courbes sont mis à jours tout au long de chaque simulation. Si vous n'avez pas besoin de voir ces mises à jours, utilisez alors les cases à cocher placées au bas de la fenêtre "Running Experiment" pour les désactiver. Cette inactivité d'affichage accélère fortement le calcul et donc le déroulement de la simulation.

Si vous voulez interrompre l'expérimentation en cours de route, pressez le bouton "Abort" (Abandonner) placé en bas à droite. Mais notez quand même que vous perdez alors tous les résultats qui ont été collectés dans le format "Spreadsheet" jusqu'à ce moment-là. Les résultats collectés dans le format "Table" étant enregistrés à la fin de chaque simulation, vous ne perdez que ceux de la simulation en cours.

Une fois toutes les simulations terminées, le programme sauvegarde tout ce qu'il doit, ferme les fichiers et se termine automatiquement : l'expérimentation est terminée. Il n'y a « plus qu'a » interpréter les résultats.

# Utilisation avancée

## Exécution à partir de la ligne de commandes

Il est possible de faire tourner des expérimentations BehaviorSpace « sans-tête » (*headless*), c'est-à-dire à partir de la ligne de commande et sans aucune interface utilisateur graphique (GUI). Ce qui est utile pour automatiser les simulations sur une seule machine ou sur un ensemble de machines.

Aucune connaissance en programmation Java n'est nécessaire. La description de l'expérimentation peut être créée dans l'interface graphique et exécutée plus tard à partir de la ligne de commandes, ou, si vous préférez, vous pouvez créer ou éditer directement cette description utilisant le langage XML.

Il est plus facile de créer d'abord vos descriptions d'expérimentations dans l'interface graphique car elles seront sauvegardées en tant que partie du modèle. Voici un exemple d'utilisation de la ligne de commande pour lancer et exécuter une expérimentation qui a été sauvegardée dans un modèle :

```
java -server -Xmx1024M -cp NetLogo.jar \
 org.nlogo.headless.HeadlessWorkspace \
 --model Fire.nlogo \
 --experiment experiment1
```

(Pour que cette méthode fonctionne, le fichier `NetLogo.jar` doit être présent avec le sous-répertoire `lib` contenant les bibliothèques nécessaires. Aussi bien le fichier `NetLogo.jar` que le sous-répertoire `lib` font parties de NetLogo.)

Une fois l'expérimentation nommée terminée, les résultats sont envoyés à la sortie standard dans un format de feuille de calculs, tel que CSV. (Pour modifier ce réglage par défaut, voir ci-dessous.)

Quand la classe `HeadlessWorkspace` fonctionne comme une application, la propriété système `java.awt.headless` est forcée à `true`. Ceci indique à Java de fonctionner en mode headless, ce qui permet à NetLogo de tourner sur des machines ne disposant pas d'un affichage graphique (donc de type terminal).

Notez l'utilisation du drapeau `-server` pour dire à Java d'optimiser les performances pour des applications de type « serveur ». Nous recommandons l'utilisation de ce drapeau pour de meilleures performances dans la plupart des situations.

Notez aussi l'utilisation de `-Xmx` pour spécifier une taille mémoire (*heap*) maximale de un gigabyte. Si vous ne spécifiez par une mémoire maximale, vous n'obtenez que la taille par défaut de la machine java virtuelle, qui peut être inhabituellement petite. (Un gigabyte est une taille arbitraire qui devrait être bien assez grande pour la plupart des modèles, mais vous pouvez spécifier une limite maximale différente si vous le voulez.)

L'argument `--model` est utilisé pour spécifier le fichier du modèle que vous voulez ouvrir.

L'argument `--experiment` est utilisé pour spécifier le nom de l'expérimentation que vous voulez mener. (Ce nom lui a été donné au moment où vous aviez créé la description de cette expérimentation dans l'interface utilisateur.)

Voici un autre exemple montrant l'utilisation d'autres arguments, cette fois en option :

```
java -server -Xmx1024M -cp NetLogo.jar \
 org.nlogo.headless.HeadlessWorkspace \
 --model Fire.nlogo \
 --experiment experiment2 \
 --max-pxcor 100 \
 --min-pxcor -100 \
 --max-pycor 100 \
 --min-pycor -100 \
 --no-results
```

Noter l'utilisation des arguments optionnels `--max-pxcor`, `--max-pycor`, etc., pour spécifier une taille de monde différente de celle qui avait été sauvegardée avec le modèle. (La description de l'expérimentation peut aussi spécifier des valeurs pour les dimensions du monde; si ces dimensions sont déjà spécifiées dans l'expérimentation, il n'est pas nécessaire de les re-spécifier dans la ligne de commande.)

Noter également l'utilisation de l'argument optionnel `--no-results` pour indiquer qu'il n'y a pas à générer de sortie de données. Cette précision est utile si les descriptions de l'expérimentation génèrent toutes les sorties de données dont vous avez besoin d'une autre manière, par exemple par exportation de fichiers world ou par écriture dans un fichier texte.

Voici encore un autre exemple :

```
java -server -Xmx1024M -cp NetLogo.jar \
 org.nlogo.headless.HeadlessWorkspace \
 --model Fire.nlogo \
 --experiment experiment2 \
 --table table-output.csv \
 --spreadsheet spreadsheet-output.csv
```

L'argument optionnel `--table <filename>` spécifie que la sortie doit être générée dans le format "Table" et écrite dans le fichier donné au format CSV. Si `-` est mis à la place du nom du fichier, les données en sortie seront envoyées dans le flux de sortie standard du système. Les données de type "Table" sont écrites comme elles sont générées, à la fin de chaque simulation.

L'argument optionnel `--spreadsheet <filename>` spécifie que la sortie doit être générée dans le format "Spreadsheet" et écrite dans le fichier donné au format CSV. Si `-` est mis à la place du nom du fichier, les données en sortie seront envoyées dans le flux de sortie standard du système. Les données de type "Spreadsheet" ne sont écrites (donc sauvegardées) que lorsque toutes les simulations sont terminées, autrement dit à la fin de l'expérimentation.

Notez qu'il est parfaitement possible de spécifier à la fois `--table` et `--spreadsheet`, et si vous le faites, les deux types de sorties seront générées.

Le comportement de sortie par défaut, quand aucun format de sortie n'est spécifié, est d'envoyer les données sous forme de table dans le flux de sortie standard du système.

Voici enfin un dernier exemple qui montre comment faire exécuter une expérimentation qui a été enregistrée dans un fichier XML séparé plutôt que dans le fichier du modèle :

```
java -server -Xmx1024M -cp NetLogo.jar \
 org.nlogo.headless.HeadlessWorkspace \
 --model Fire.nlogo \
 --setup-file fire-setups.xml \
 --experiment experiment3
```

Si le fichier XML contient plus d'une description d'expérimentation, il est nécessaire d'utiliser l'argument `--experiment` pour spécifier le nom de la description à utiliser.

La section suivante montre comment créer des fichiers de descriptions d'expérimentations en utilisant le langage XML.

## Décrire des expérimentations en XML

Nous n'avons pour le moment pas encore de documentation détaillée sur la programmation de descriptions d'expérimentations en XML, mais si vous êtes déjà quelque peu familiarisés avec ce langage, les quelques indications suivantes devraient être suffisantes pour vous permettre de commencer.

La structure des descriptions d'expérimentations de BehaviorSpace en XML est déterminée par un fichier "Document Type Definition" (DTD). Ce DTD est stocké dans le fichier `NetLogo.jar` en tant que fichier `system/behaviorspace.dtd`. (Les fichiers JAR étant aussi des fichiers compressés zip, vous pouvez extraire le fichier DTD du fichier JAR avec l'utilitaire Java `jar` ou avec tout programme capable de décompresser les fichiers zip.)

La méthode la plus facile pour voir à quoi ressemble une description d'expérimentation en XML est d'un créer quelques-unes dans l'interface graphique de BehaviorSpace, de sauvegarder le modèle puis d'examiner le fichier `.nlogo` résultant dans un éditeur de texte. Les descriptions d'expérimentations se trouvent vers la fin du fichier `.nlogo`, dans une section qui commence et se termine par une balise `experiments`. Par exemple :

```
<experiments>
 <experiment name="experiment" repetitions="10" runMetricsEveryStep="true">
 <setup>setup</setup>
 <go>go</go>
 <exitCondition>not any? fires</exitCondition>
 <metric>burned-trees</metric>
 <enumeratedValueSet variable="density">
 <value value="40"/>
 <value value="0.1"/>
 <value value="70"/>
 </enumeratedValueSet>
 </experiment>
</experiments>
```

Cet exemple ne comporte qu'une seule description d'expérimentation, mais vos pouvez en mettre autant que vous le voulez entre les deux balises `experiments`.

En combinant l'observation du fichier DTD et celle des exemples que vous avez créés dans l'interface graphique, vous devriez pouvoir comprendre comment utiliser les différentes balises pour spécifier différents types d'expérimentations. Le ficher DTD spécifie quelles balises sont obligatoires et quelles balises sont optionnelles, lesquelles peuvent être répétées et lesquelles ne doivent apparaître qu'une seule fois, et ainsi de suite.

Quand les descriptions d'expérimentations en XML sont incluses dans le fichier du modèle, cette section ne doit pas commencer avec les en-têtes XML habituelles car le fichier `.nlogo` n'est pas un fichier XML dans sa totalité. Par contre, si vous gardez les descriptions d'expérimentations dans leur propre fichier, séparé du fichier du modèle, l'extension de nom de ce fichier devra être `.xml` et non pas `.nlogo`, et vous devrez le commencer par ses propres en-têtes XML, comme celles qui suivent :

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE experiments SYSTEM "behaviorspace.dtd">
```

La deuxième ligne doit être écrite exactement comme montrée ci-dessus. Dans la première ligne, vous pouvez spécifier un autre encodage de caractères que `us-ascii`, tel que `UTF-8` par exemple, mais souvenez-vous que NetLogo ne supporte pas les caractères non-ASCII (entres autres les caractères accentués) dans la plupart des situations. Il s'ensuit que la spécification d'un encodage différent peut se révéler peu judicieux.

### Controlling API

Si BehaviorSpace ne peut satisfaire vos attentes, une alternative possible consiste à utiliser notre Controlling API qui vous permet d'écrire du code Java pour contrôler NetLogo. L'API vous permet de faire tourner des expérimentations BehaviorSpace à partir de code Java, et vous pouvez même écrire du code personnalisé qui contrôle encore plus directement NetLogo pour lui faire faire des choses qui ressemblent à ce que lui fait faire BehaviorSpace. Consultez la section Controlling du Manuel de l'utilisateur pour en savoir plus sur ces deux possibilités.

## Conclusion

BehaviorSpace est encore en cours de développement. Nous aimerions avoir votre avis concernant des fonctionnalités supplémentaires qui pourraient être utiles pour vos recherches et travaux. Écrivez-nous à l'adresse feedback@ccl.northwestern.edu.

# Chapitre 17

# Guide du Système dynamique

Ce chapitre est divisé en trois parties :

- **Qu'est-ce que le Modélisateur de Systèmes Dynamiques?** : Une description générale de l'outil, incluant les idées et les principes que le sous-tendent.
- **Comment cela fonctionne** : Description de l'interface et de son utilisation.
- Le tutoriel **Prédation Loups-Moutons (agrégat)** vous conduit à travers la création d'un modèle avec le Modélisateur de Systèmes Dynamiques.

## Qu'est-ce que le Modélisateur de Systèmes dynamiques de NetLogo?

Les **systèmes dynamiques** sont un type de modélisation où l'on essaie de comprendre comment les choses sont en relations les unes avec les autres. Cette approche est un peu différente de celle basée sur les agents qui est normalement utilisée par les modèles NetLogo.

Avec l'approche basée sur les agents, vous programmez le comportement individuel des agents et observez ce qui « émerge » de leurs interactions. Dans un modèle de prédation loups-moutons par exemple, vous fournissez des règles décrivant comment les loups, les moutons et l'herbe interagissent les uns avec les autres. Quand vous exécutez la simulation, vous observez le **comportement émergeant** au niveau des « agrégats » : par exemple, comment les populations de loups et de moutons évoluent au cours du temps.

Avec le modélisateur de systèmes dynamiques, vous ne programmez pas le comportement individuel des agents. Vous programmez à la place comment les populations d'agents se comportent de manière globale. Par exemple, en utilisant les systèmes dynamiques pour modéliser la prédation loups-moutons, vous spécifiez comment le nombre total de moutons sera modifié quand le nombre total de loups augmentera ou diminuera, et vice-versa. Vous lancez alors la simulation pour voir comment les deux populations évoluent au cours du temps.

La programmation du modélisateur de systèmes dynamiques se fait en dessinant un diagramme définissant d'une part les populations en causes ou "stocks" et d'autre part comment ces populations interagissent les unes avec les autres. Le modélisateur lit alors votre diagramme et génère le code NetLogo approprié – les variables globales, les procédures et les reporters – permettant de faire « tourner » votre système dynamique dans NetLogo.

### Les concepts de base

Un diagramme de système dynamique est formé de quatre types d'éléments : les stocks, les variables, les flux et les liens.

- Un **stock** est une collection de choses, un agrégat. Par exemple, un stock peut représenter une population de moutons, l'eau d'un lac ou le nombre de machines dans une usine.
- Un **flux** (*flow*) transporte ces choses, remplissant ou vidant un stock. Les flux ressemblent à des tuyaux munis d'un robinet, le robinet permettant de contrôler la quantité de choses qui s'écoulent dans le tuyau.
- Une **variable** est une valeur utilisée dans le diagramme. Elle peut être soit une équation qui dépend d'autres variables, soit une constante.
- Un **lien** (*link*) permet qu'une valeur présente dans une partie du diagramme soit aussi disponible dans une autre partie. Un lien transmet un nombre d'une variable ou d'un stock à un autre stock ou à un flux.

Le Modélisateur de systèmes dynamiques calcule comment les valeurs des stocks changent au cours du temps en les évaluant encore et encore. L'estimation n'est pas toujours parfaite, mais vous pouvez influencer sa précision en changeant la valeur de **`dt`**. Quand `dt` diminue, les valeurs du modèle sont évaluées plus fréquemment, ce qui fait qu'il devient plus précis. Toutefois, la diminution de la valeur de `dt` ralentit aussi la vitesse de la simulation.

### Les exemples de modèles dynamiques

La bibliothèque des modèles NetLogo "NetLogo Models Library" contient dans sa section "Sample Models" quatre exemples qui utilisent le Modélisateur de systèmes dynamiques. Ces quatre modèles explorent la croissance de populations (et, dans les modèles avec prédation, aussi les déclins de populations). Ce sont :

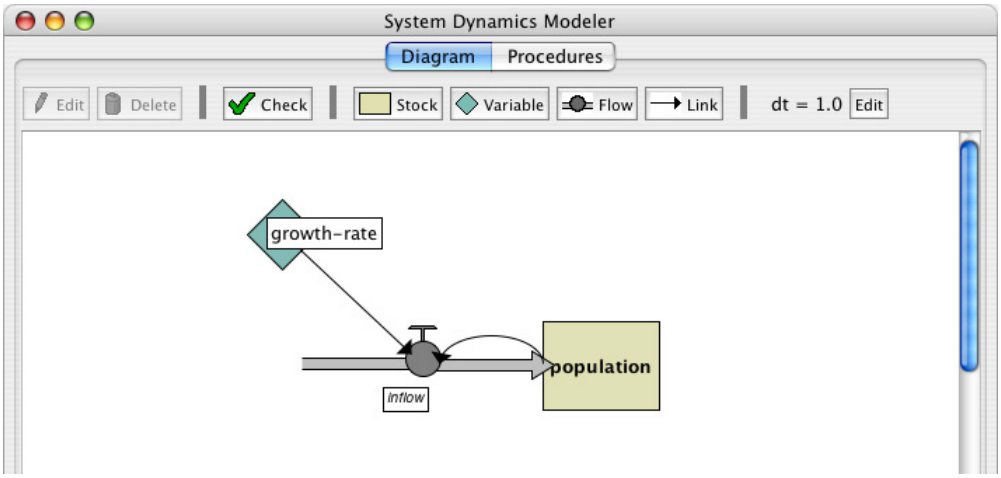
- "**Exponential Growth**" (Croissance exponentielle) et "**Logistic Growth**" (Croissance logistique) sont de simples exemples de la croissance d'un stock.
- "**Wolf Sheep Predation (aggregate)**" (Prédation loups-moutons (agrégat)) est un exemple de système avec de multiples stocks qui s'influencent les uns les autres. Il modélise un écosystème de prédateurs-proies en utilisant le Modélisateur de systèmes dynamiques.
- **"Wolf Sheep Predation (docked)**" est un exemple de modèle qui fait tourner côte à côte un modèle de système dynamique et un modèle à base d'agents. En fait, il fait tourner l'implémentation en système dynamique de la Prédation loups-moutons à côté de la Prédation loups-moutons à base d'agents qui de trouve dans la section "Biology" des "Sample Models".

## Comment cela fonctionne

Pour ouvrir le **Modélisateur de systèmes dynamiques** ("System Dynamics Modeler"), sélectionnez la commande "System Dynamics Modeler" dans le menu "Tools". La fenêtre "System Dynamics Modeler" s'ouvre alors.

### Le Panneau "Diagram"

Le panneau "Diagram" (diagramme) est l'endroit où dessiner le diagramme de votre système dynamique. Sa barre d'outils contient les boutons pour éditer, supprimer ou créer des éléments dans le diagramme.




### Créer les éléments du diagramme


Le diagramme d'un système dynamique est formé de quatre types de composants : les stocks, les variables, les flux (flows) et les liens (links).

- Les **stock** — Pour créer un stock, pressez le bouton "Stock" de la barre d'outils et cliquez dans la zone de dessin située en-dessous. Un nouveau stock apparaît. Chaque stock demande un nom unique, qui deviendra une **variable globale**. Les Stocks requièrent aussi une **valeur initiale** ("Initial value"). Cette dernière peut être un nombre, une variable, une expression NetLogo complexe ou un appel à un reporter NetLogo.
- Les **variable** — Pour créer une variable, pressez le bouton "Variable" et cliquez dans la zone de dessin du diagramme. Chaque variable d'un Système dynamique doit recevoir un nom unique, qui deviendra le nom d'une **procédure** ou d'une **variable globale**. Les variables demandent aussi une **expression**. Cette expression peut être un nombre, une variable, une expression NetLogo complexe ou un appel à un reporter NetLogo.
- Les **flux** — Pour créer un flux, pressez le bouton "Flow". Cliquez et maintenez pressé là où le flux doit commencer – soit sur un Stock, soit sur une zone vide – et tirez jusqu'à l'endroit où vous voulez qu'il aboutisse – sur un Stock ou une zone vide. Chaque flux doit avoir un nom unique, qui deviendra celui d'un **reporter** NetLogo. Les flux demandent une aussi **expression**, qui doit représenter le « débit » du flux, de l'entrée à la sortie du « tuyau ». Cette expression peut être un nombre, une variable, une expression NetLogo complexe ou un appel à un reporter NetLogo. Si sa valeur est négative, le flux s'écoule dans l'autre direction.

  Quand plus d'un flux est connecté à un stock, il est important de prendre en compte la manière dont ces flux vont interagir les uns avec les autres. NetLogo n'oblige pas les flux qui sortent d'un stock de le faire dans un ordre particulier. De plus, NetLogo ne regarde pas si la totalité des flux qui sortent d'un stock est inférieure ou égale au contenu du stock. Ces comportements peuvent être implémentés explicitement dans l'**expression** exigée par le flux.

  Par exemple, si un flux est défini par une valeur constante, disons `10`, vous pouvez vous assurer qu'il ne draine jamais plus que la valeur du stock en utilisant la primitive `min` : `min (list stock 10)`. Si je veux qu'un flux A vide un stock avant que flux B ne soit calculé, je peux lier le flux A au flux B et modifier flux B pour soustraire la valeur de flux A du stock : `min (list (max (list 0 (stock - flow-a))) 10)`.
- Les **liens** — Pour créer un lien, cliquez et maintenez pressé sur le point de départ du lien – une variable, un stock ou un flux – et tirez jusqu'à la variable ou le flux de destination.

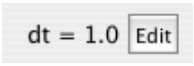
### Travailler avec les éléments du diagramme

Quand vous créez un stock, une variable, ou un flux, NetLogo affiche un point d'interrogation rouge sur le nouvel élément. Ce point d'interrogation indique que l'élément en question n'a pas encore reçu de nom. Un nom d'élément rouge indique que le stock n'est pas complet : il manque une ou plusieurs des valeurs requises pour pouvoir créer un modèle de système dynamique. Quand un élément du diagramme est complet, son nom est affiché en noir.

- **La sélection.** Pour sélectionner un élément d'un diagramme, cliquez-le. Pour sélectionner plusieurs éléments, pressez la touche Majuscule et cliquez sur les éléments à sélectionner. Vous pouvez aussi sélectionner un ou plusieurs éléments en tirant une boîte de sélection avec la souris.
- **L'édition.** Pour éditer un élément du diagramme, sélectionnez l'élément et pressez le bouton "Edit" de la barre d'outils. Il suffit aussi de double-cliquer l'élément. (Vous pouvez éditez les stocks, les flux et les variables, mais vous ne pouvez pas éditer les liens).
- **Le déplacement.** Pour déplacer un élément du diagramme, sélectionnez-le et tirez-le vers son nouvel emplacement.

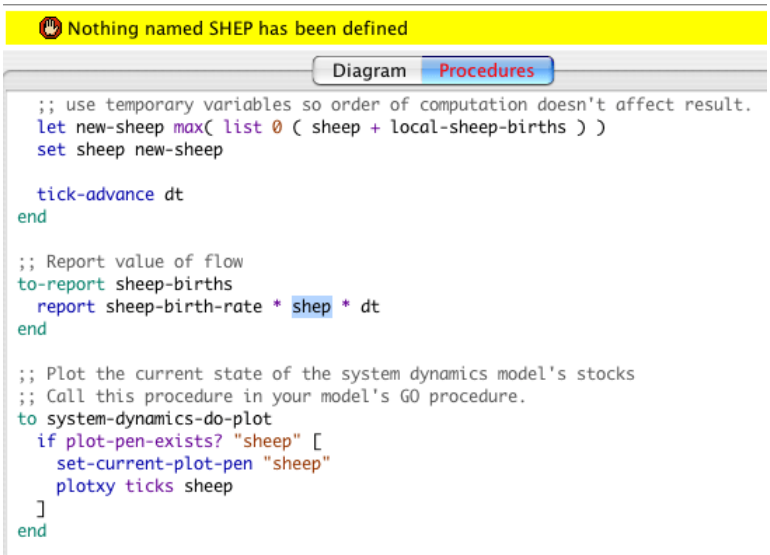
### Editer `dt`

La partie droite de la barre d'outils affiche le **`dt`** par défaut, c'est-à-dire la durée écoulée entre deux calculs des états des paramètres du système dynamique. Cette durée `dt` correspond à un pas de la simulation. Pour modifier cette valeur, pressez le bouton "Edit" (à droite de la valeur de `dt` affichée) et entrez une nouvelle valeur.

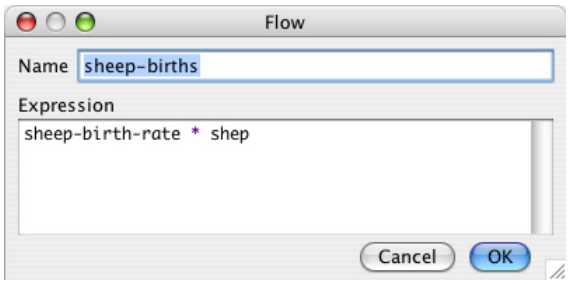



### Les erreurs

Quand vous cliquez le bouton "Check" ou éditez un stock, un flux ou une variable, le modélisateur génère automatiquement le code NetLogo correspondant au diagramme et tente de le compiler. S'il y a une erreur, le nom de l'onglet "Procedures" devient rouge, un message est affiché et la portion de code contenant l'erreur est mise en évidence.




Ces informations devraient vous aider à avoir une meilleure idée de l'élément du diagramme qui est à l'origine du problème.




## Le panneau "Procedures"

Le Modélisateur de Systèmes dynamiques génère des variables et des procédures NetLogo basées sur le contenu du diagramme construit. Ce sont en fait ces procédures qui permettent à NetLogo d'effectuer les calculs nécessaires à la simulation décrite par le diagramme. Le panneau "Procedures" de la fenêtre du Modélisateur de Systèmes dynamiques affiche les procédures NetLogo générées.

Le contenu du panneau "Procedures" n'est pas éditable. Pour modifier le système dynamique, vous devez éditer le diagramme.

Examinons d'un peu plus près comment le code généré se rapporte au diagramme :

- Les stocks correspondent à des variables globales qui ont été initialisées avec la valeur ou l'expression fournie dans le champ "Initial value". Chaque stock est mis à jour à chaque pas, mise à jour basée sur les flux entrant et sortant.
- Les flux correspondent aux procédures contenant les expressions fournies dans "Expression".
- Les variables peuvent être soit des variables globales, soit des procédures. Si l'expression fournie est une constante, elle est devenue une variable globale et a été initialisée avec cette valeur. Si vous avez utilisé une "Expression" plus complexe pour définir la variable, elle a été transformée en une procédure, comme pour les flux.

Les variables et les procédures définies dans ce panneau sont accessibles depuis la fenêtre principale de NetLogo, de la même manière que le sont les procédures et les variables définies dans le panneau "Procedures" principal de NetLogo. Vous pouvez appeler ces procédures depuis le panneau "Procedures" principal, depuis le Centre de commande ou avec les boutons placés sur le panneau "Interface". Vous pouvez faire référence aux variables globales à partir de n'importe où, y compris à partir du panneau "Procedures" principal et dans les moniteurs.

Il y a trois procédures importantes à signaler, qui sont :

- **`system-dynamics-setup`** initialise le modèle de Système dynamique. Il donne sa valeur à `dt`, appelle `reset-ticks` et initialise les stocks et les convertisseurs. Les convertisseurs à valeur constante sont initialisés en premier, suivis par les stocks à valeur constante. Les autres stocks sont initialisés ensuite par ordre alphabétique.
- **`system-dynamics-go`** fait tourner le Système dynamique par pas d'unité de temps `dt`. Il calcule les valeurs des flux et des variables et met à jour la valeur des stocks. Il appelle aussi `tick-advance` avec la valeur de `dt`. Les convertisseurs et les flux avec des "Expressions" non-constantes ne sont calculés que lorsque cette procédure est appelée, toutefois, l'ordre de leur évaluation n'est pas défini.
- **`system-dynamics-do-plot`** affiche les valeurs des stocks du Système dynamique. Pour utiliser cette procédure, il faut d'abord créer un traceur de courbes dans le panneau "Interface" de NetLogo. Il faut définir un crayon pour chaque stock dont vous voulez représenter le graphique. Cette procédure utilise le traceur courant, que vous pouvez changer au moyen de la commande `set-current-plot`.

### Le Modélisateur de Systèmes dynamiques et NetLogo

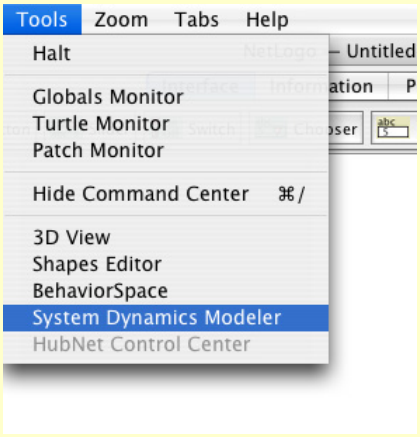
Le diagramme que vous créez dans le Modélisateur de Systèmes dynamiques et les procédures générées à partir de ce diagramme font partie du modèle NetLogo. Quand vous sauvegardez ce modèle, le diagramme est sauvegardé dans le même fichier.

## Tutoriel : Prédation loups-moutons

Nous allons maintenant créer un modèle de « Prédation loups-moutons » avec le Modélisateur de Systèmes dynamiques.

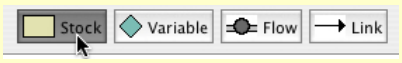
### Première étape : la reproduction des moutons

- Ouvrez un nouveau modèle en NetLogo.
- Lancez le " System Dynamics Modeler " dans le menu " Tools ".



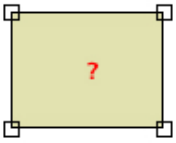

Notre modèle aura une population de loups et une population de moutons. Commençons par les moutons. Il faut d'abord créer le stock qui contiendra la population de moutons.

- Pressez le bouton " Stock " dans la barre d'outils.

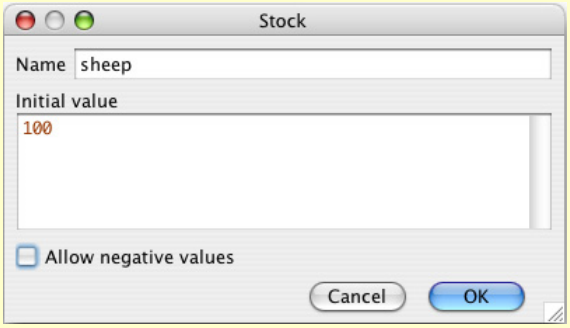



- Cliquez dans la zone du diagramme.

Vous voyez un stock avec un point d'interrogation rouge au centre.



- Double-cliquez ce stock pour l'éditer.
- Nommez ce Stock `sheep` (moutons).
- Mettez la valeur initiale " Initial value " à `100`.
- Désactivez la case à cocher " Allow Negative Values " (autoriser les valeurs négatives). Un nombre de moutons négatif n'a pas de sens!




La population de moutons peut s'accroître si un mouton naît. Pour ajouter ce fait à notre diagramme, il faut créer un flux entrant dans le stock de moutons.

- Cliquez le bouton " Flow " de la barre d'outils et pressez le bouton de la souris dans une partie vide à gauche du stock de moutons. Tirez le flux vers la droite jusqu'à ce qu'il se connecte avec le stock de moutons. Relâchez le bouton de la souris.
- Éditez le flux et nommez-le `sheep-births`.

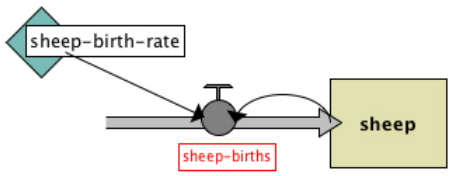
Le nombre de moutons nés pendant une période de temps `dt` dépend du nombre de moutons vivants : plus il y a de moutons vivants, plus il y a de nouveaux-nés.

- Dessinez un lien partant du stock `sheep` et allant vers le flux `sheep-births`.
- Pour le moment, entrez une constante, par exemple `1`, dans le champ " Expression ".

Le taux de natalité des moutons dépend aussi de certains facteurs constants dont la discussion sort du cadre de ce modèle, comme le taux de reproduction par exemple.

- Créez une variable et nommez-la `sheep-birth-rate`.
- Donnez-lui une valeur de `0.04`
- Dessinez un lien de la variable `sheep-birth-rate` au flux `sheep-births`.

Votre diagramme devrait ressembler maintenant à ceci :

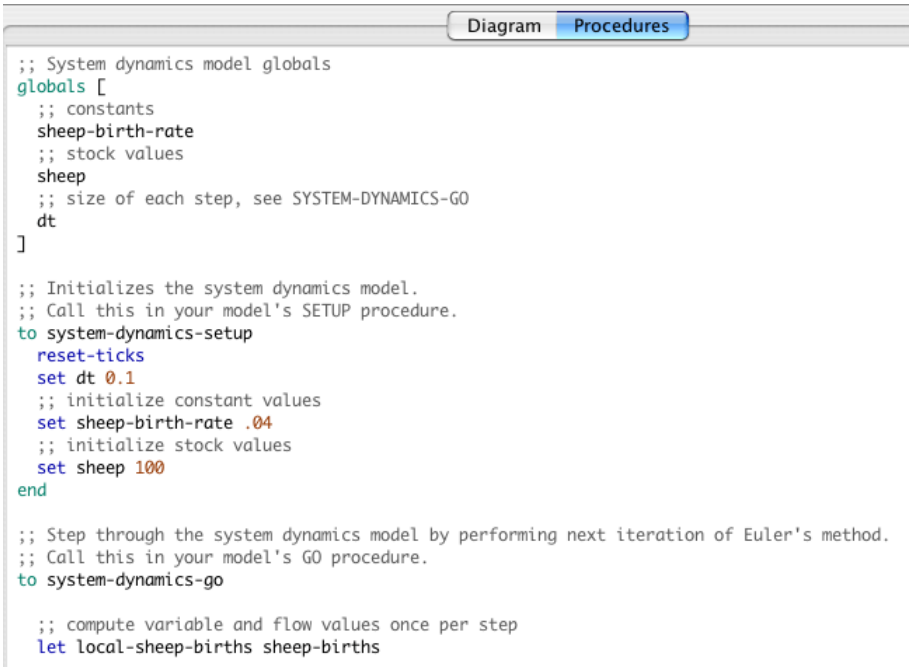



L'étiquette du flux `sheep-births` est rouge parce que nous ne lui avons pas encore donné d'expression. La couleur rouge indique toujours qu'il manque quelque chose à cette partie du diagramme.

Le nombre de moutons entrant dans notre stock dépendra positivement du nombre de moutons et du taux de natalité `sheep-birth-rate`.

- Éditez le flux `sheep-births` et écrivez l'expression suivante : `sheep-birth-rate * sheep`.

Nous avons maintenant un diagramme complet. Pour voir le code NetLogo généré par ce diagramme, cliquez l'onglet "Procedures" de la fenêtre du Modélisateur de Systèmes dynamiques. Le code devrait être le suivant :




```
;; System dynamics model globals
globals [
  ;; constants
  sheep-birth-rate
  ;; stock values
  sheep
  ;; size of each step, see SYSTEM-DYNAMICS-GO
  dt
]

;; Initializes the system dynamics model.
;; Call this in your model's SETUP procedure.
to system-dynamics-setup
  reset-ticks
  set dt 0.1
  ;; initialize constant values
  set sheep-birth-rate .04
  ;; initialize stock values
  set sheep 100
end

;; Step through the system dynamics model by performing next iteration of Euler's method.
;; Call this in your model's GO procedure.
to system-dynamics-go

  ;; compute variable and flow values once per step
  let local-sheep-births sheep-births
```

## Deuxième étape : intégration à NetLogo

Une fois que le modèle d'agrégats a été créé avec le Modélisateur de Systèmes dynamiques, nous pouvons interagir avec ce modèle par l'intermédiaire de la fenêtre "Interface" de NetLogo. Construisons maintenant le modèle NetLogo nous permettant d'exécuter le code généré par le diagramme. Nous avons besoin des boutons `setup` et `go` pour appeler les procédures `system-dynamics-setup` et `system-dynamics-go` créées par le Modélisateur de Systèmes dynamiques. De plus, pour pouvoir bien suivre l'évolution des populations il nous faut aussi un moniteur et un traceur.

- Sélectionnez la fenêtre NetLogo principale.
- Dans le panneau " Procedures ", écrivez :

```
to setup
 ca
 system-dynamics-setup
end

to go
 system-dynamics-go
 system-dynamics-do-plot
end
```

- Allez dans le panneau " Interface ".
- Créez un bouton `setup`.
- Créez un bouton `go` (n'oubliez pas d'en faire un bouton « pour-toujours » dans la case à cocher " Forever ").
- Créez un moniteur `sheep`.
- Créez un traceur appelé `populations` avec un crayon nommé `sheep`.

Nous sommes maintenant prêt à faire tourner notre modèle.

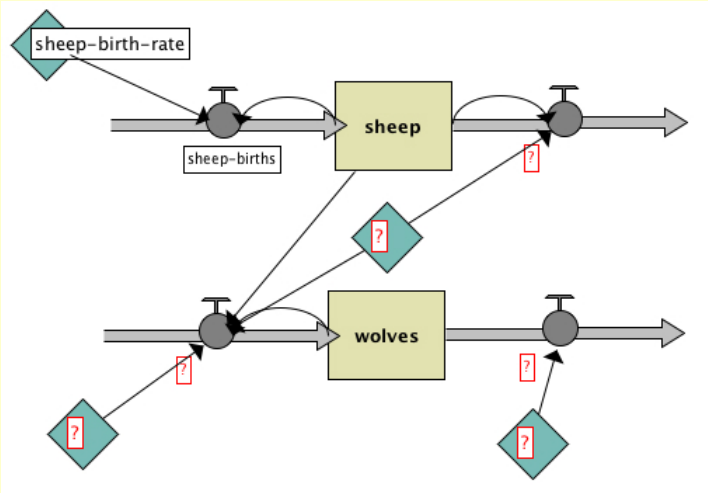
- Pressez le bouton " setup ".
- Ne pressez pas le bouton " go " maintenant. Écrivez plutôt quatre ou cinq fois de suite `go` dans la Ligne de commandes du Centre de commande en pressant la touche Retour après chaque `go`.

Observez ce qui se passe. La population de moutons s'accroît exponentiellement. Après quatre ou cinq itérations, nous avons une énorme quantité de moutons. Nous aboutissons à cette situation parce que nous avons des moutons qui se reproduisent mais qui ne meurent jamais.

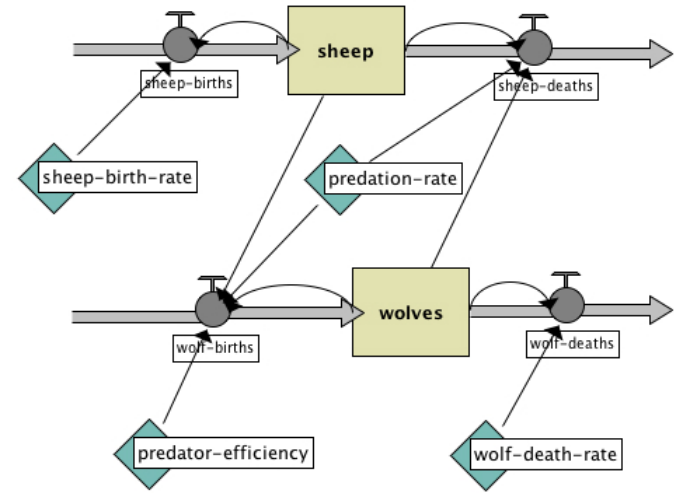
Pour résoudre ce problème, terminons notre diagramme en introduisant une population de loups qui mangent des moutons.

## Troisième étape : la prédation par les loups

- Revenez à la fenêtre " System Dynamics "
- Ajoutez un stock de loups appelé `wolfes`.
- Ajoutez des flux, des variables et des liens pour que le diagramme ressemble à :




- Ajoutez encore un lien allant du stock `wolves` au flux qui sort du stock `sheep`.
- Nommez les éléments du diagramme de la manière suivante :

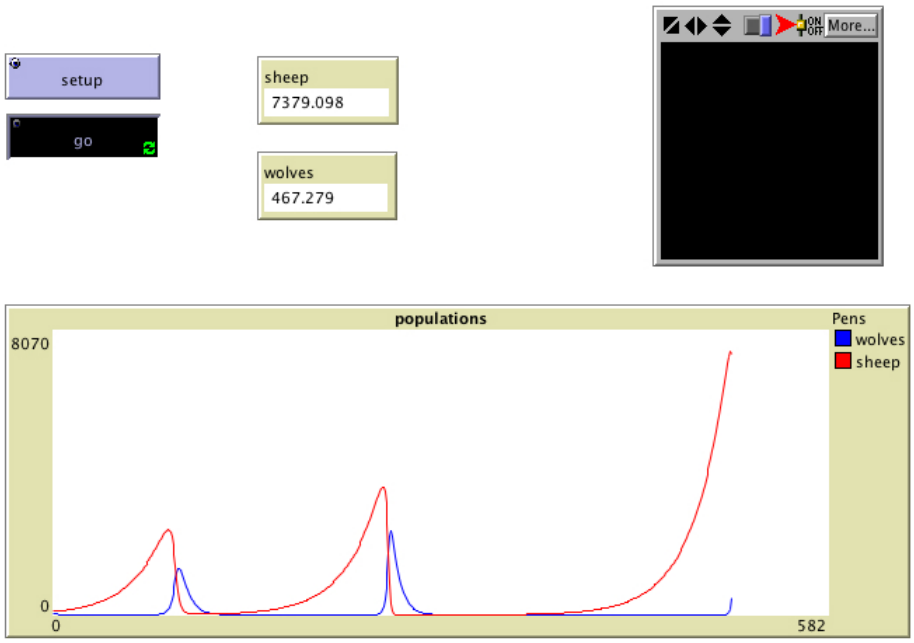


où
la valeur initiale des loups ( `wolves` ) est `30`,
`wolf-deaths` vaut `wolves * wolf-death-rate`,
`wolf-death-rate` vaut `0.15`,
`predator-efficiency` vaut `0.8`,
`wolf-births` vaut `wolves * predator-efficiency * predation-rate * sheep`,
`predation-rate` vaut `3.0E-4`,
et `sheep-deaths` vaut `sheep * predation-rate * wolves`.

Nous en avons maintenant vraiment terminé avec le diagramme, mais il faut encore mettre le modèle NetLogo à jour.

- Retournez dans la fenêtre principale de NetLogo.
- Ajoutez un crayon appelé `wolves` au traceur " populations ".
- Pressez `setup` et `go` pour voir le diagramme de votre Système dynamique en action.

Vous devriez voir un graphique des populations qui ressemble à ceci :




## Conclusion

Le Modélisateur de Systèmes Dynamiques est toujours en cours de développement.

### Améliorations planifiées

Parmi les fonctionnalités, nous avons prévu d'ajouter :

- ✔ la possibilité de copier ou de sauvegarder une image du diagramme du Système dynamique ;
- ✔ l'affichage des variables fournies à l'élément par les liens lors de l'édition d'un stock ou d'un flux ;
- ✔ une sélection des liens plus facile.

## Retour d'informations

Si vous découvrez des bogues (bugs) dans le Modélisateur de Systèmes dynamiques, faites-nous le savoir à l'adresse bugs@ccl.northwestern.edu.

Nous aimerions aussi que vous nous fassiez part de vos désirs concernant de nouvelles fonctionnalités qui pourraient être utiles pour vos travaux. Si vous avez des questions, des commentaires ou des suggestions, écrivez-nous à l'adresse feedback@ccl.northwestern.edu.

# Chapitre 18

# Guide de HubNet

Ce chapitre du Manuel de l'utilisateur présente le système HubNet et donne les instructions nécessaires pour le mettre en place et le faire fonctionner.

**HubNet** est une technologie qui utilise NetLogo pour faire fonctionner des *simulations participatives* (en réseau) dans le cadre de la classe. Dans une simulation participative, toute la classe entre en interaction et peut agir sur le comportement d'un système en ce sens que chaque étudiant contrôle une partie du système en se servant d'un appareil individuel, soit un ordinateur soit une calculatrice graphique Texas Instruments relié au réseau.

Par exemple, dans la simulation "Gridlock", chaque étudiant contrôle un signal lumineux réglant une partie du trafic d'une ville simulée. La classe dans son ensemble essaie de contrôler le trafic de manière à ce qu'il soit fluide dans toute la ville. Des données sont récoltées tout au long de la simulation afin qu'elles puissent, par la suite, être analysées par ordinateur ou calculatrice.

Pour plus d'informations concernant les simulations participatives et leur potentiel de formation, visitez le site Participatory Simulations Project.

## Comprendre HubNet

### NetLogo

**NetLogo** est un environnement de modélisation programmable. Il est accompagné d'une importante bibliothèque de simulations, soit participatives, soit traditionnelles (individuelles), que vous pouvez utiliser ou modifier. Ces simulation couvrent de nombreux domaines tels que les sciences sociales et économiques, la biologie et la médecine, la physique et la chimie, les mathématiques et l'informatique, et même les jeux. Vous et vos étudiants pouvez aussi les utiliser comme base pour développer vos propres simulations. Pour en savoir plus sur NetLogo, lisez le Manuel de l'utilisateur NetLogo.

Dans les simulations NetLogo traditionnelles, la simulation se déroule en fonction des règles spécifiées par l'auteur de la simulation. HubNet ajoute une nouvelle dimension à NetLogo en permettant aux simulations de se dérouler, non seulement en suivant des règles préprogrammées, mais en réagissant directement aux instructions des participants.

HubNet étant construit sur les bases de NetLogo, nous vous recommandons de vous familiariser d'abord avec le fonctionnement de NetLogo avant d'essayer HubNet. Pour apprendre à utiliser les modèles NetLogo, parcourez et testez le Tutoriel 1 : Les Modèles du Manuel de l'utilisateur NetLogo.

### L'architecture de HubNet

Les simulations HubNet sont basées sur une architecture client-serveur. Le directeur de l'activité utilise l'application NetLogo pour faire tourner une activité HubNet. Quand NetLogo fait tourner une activité HubNet, nous y faisons référence en tant que **serveur HubNet**. Les participants utilisent une application **client** pour se connecter (logger) et interagir avec le serveur HubNet.

HubNet existe en deux versions. Avec Computer HubNet, les participants font tourner l'application **HubNet Client** sur des ordinateurs connectés à un réseau informatique normal. Avec Calculator HubNet, créé en collaboration avec Texas Instruments, les participants utilisent des calculatrices graphiques Texas Instruments en tant que clients pour communiquer via le système TI-Navigator.

Nous espérons ajouter par la suite d'autres types de clients tels que téléphones cellulaires et PDA.

## Computer HubNet

### Les activités proposées

Les activités suivantes sont à disposition dans le dossier "Computer HubNet Activities" de la bibliothèque des modèles NetLogo. Pour plusieurs modèles, vous trouverez une présentations de leurs buts éducatifs et des suggestions sur la manière de les incorporer à votre enseignement dans la page "Participatory Simulations Guide" du site web Participatory Simulations Project. Des informations supplémentaires peuvent aussi être trouvées dans le panneau "Information" de chaque modèle.

- "**Disease**" — Une maladie se répand dans une population d'étudiants simulée.
- "**Gridlock**" — Les étudiants utilisent des feux de signalisation pour contrôler les flux de trafic dans une ville.
- "**Polling**" — Pose des questions aux étudiants et affiche leurs réponses.
- "**Tragedy of the Commons**" — Les étudiants travaillent en tant que fermiers pour se partager une ressource commune.

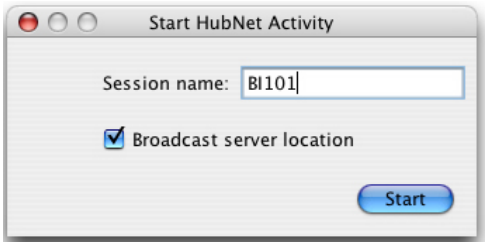
### Matériel nécessaire

Pour utiliser Computer HubNet, vous avez besoin d'un ordinateur connecté au réseau (le serveur HubNet) avec NetLogo installé pour le directeur de l'activité et d'un ordinateur connecté au réseau avec NetLogo installé pour chaque participant. Nous suggérons aussi d'avoir un projecteur vidéo connecté au serveur HubNet afin de projeter toute la simulation aux participants.

### Commencer une activité

Vous trouverez des activités HubNet dans le dossier "Computer HubNet Activities" de la bibliothèque des modèles NetLogo. Nous vous suggérons d'acquérir un peu d'expérience en faisant tourner et en expérimentant pour vous l'activité envisagée avant de l'essayer devant et avec toute la classe.

Démarrez NetLogo puis ouvrez un modèle se trouvant dans le dossier "Computer HubNet Activities". Reconnaissant un modèle HubNet, NetLogo ouvre la fenêtre ci-dessous et vous demande d'entrer le nom de la nouvelle session dans le champ "Session name:". C'est ce nom que les participants devront utiliser pour identifier cette activité. Entrez un nom et pressez le bouton "Start".

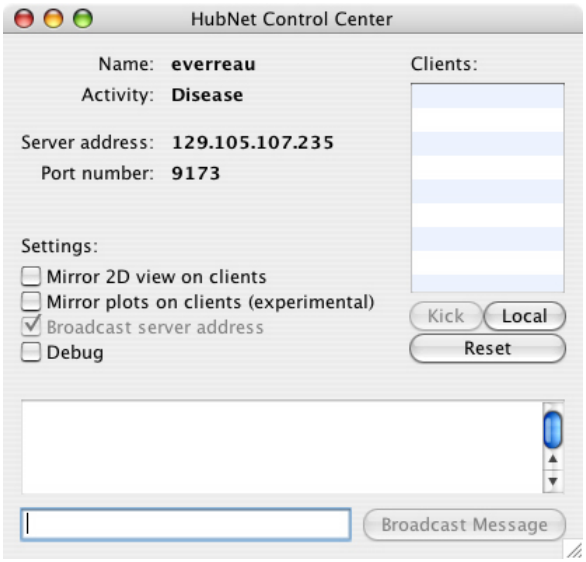



NetLogo ouvre alors le "HubNet Control Center", qui permet d'interagir avec le serveur HubNet.

Vous, en tant que directeur, devez maintenant dire à chaque participant qu'il peut se connecter au serveur HubNet. Pour rejoindre l'activité, les participants doivent démarrer l'application **HubNet Client** et entrer leur nom. Ils devraient alors voir l'activité affichée dans la liste et pouvoir la rejoindre en sélectionnant son nom (le nom de session que vous aviez donné pour démarrer l'activité) puis en pressant "Enter". Si l'activité que vous avez lancée n'est pas dans la liste, l'étudiant peut entrer manuellement l'adresse du serveur, adresse qui est affichée dans le "HubNet Control Center".

## Le "HubNet Control Center"

Le Centre de contrôle de HubNet "HubNet Control Center" vous permet d'interagir avec le serveur HubNet. Il affiche le nom "Name:", l'activité "Activity:", l'adresse du serveur "Server address:" et le numéro de port "Port number:" de votre serveur. La case à cocher "Mirror 2D View on clients" spécifie si les participants à l'activité HubNet peuvent voir la Vue du serveur sur leur machine client, étant entendu qu'il y a une Vue dans l'installation client. La case à cocher "Mirror plots on clients" spécifie si les participants reçoivent les informations destinées aux traceurs de courbes.




La liste des clients "Clients:" sur la droite affiche les noms des clients qui sont actuellement connectés à l'activité en cours. Pour faire sortir un client de l'activité, sélectionnez son nom dans cette liste et pressez le bouton "Kick" . Pour lancez votre propre client HubNet, pressez le bouton "Local" , ce qui est particulièrement utile quand vous devez déverminer (débugguer) une activité. Le bouton "Reset" éjecte tous les clients actuellement connectés et recharge l'interface client.

La partie inférieure du Centre de contrôle affiche des messages quand un client rejoint ou quitte l'activité. Pour envoyer un message à tous les participants, cliquez dans le champ texte inférieur, tapez votre message et pressez le bouton "Broadcast Message".

## Dépannages

***J'ai lancé une activité HubNet, mais quand les participants ouvrent un client HubNet, mon activité n'est pas listée.***

Avec certains réseaux, le client HubNet ne peut pas détecter automatiquement un serveur HubNet. Demandez alors aux participants d'entrer manuellement l'adresse du serveur et le port de votre serveur HubNet, informations qui sont affichées dans le Centre de contrôle HubNet.

**Note** — Les détails techniques sont les suivants : afin que le client puisse détecter le serveur, le routage multi-diffusion (*multicast routing*) doit être possible entre-eux. Tous les réseaux ne supportent pas ce type de routage, en particulier ceux qui utilisent le protocole IPsec. Le protocole IPsec est utilisé sur de nombreux réseaux privés virtuels (VPNs).

***Rien ne se passe quand un participant essaie de se connecter à une activité (le client semble bloqué ou affiche un message d'erreur indiquant que le serveur n'a pas été trouvé).***

Si votre ordinateur ou le réseau a un pare-feu, ce dernier peut empêcher les communications avec le serveur HubNet. Vérifiez que votre ordinateur et le réseau ne bloquent pas les ports utilisés par le serveur HubNet (ports 9173-9180).

***La Vue du client HubNet est grise.***

- ✔ Vérifiez que la case à cocher "Mirror 2D view on clients" du Centre de contrôle HubNet soit activée.
- ✔ Assurez-vous que l'interrupteur d'affichage du modèle soit bien enclenché.
- ✔ Si la grandeur de la Vue sur le serveur a été modifiée, vous devrez certainement presser le bouton "Reset" du Centre de contrôle pour être sûr que la Vue des clients reçoive la nouvelle taille.

***Il n'y a pas de Vue sur le client HubNet.***

Certaines activités n'ouvrent pas de Vue chez le client. Si vous voulez ajouter une Vue, il suffit de sélectionner la commande "HubNet Client Editor" du menu "Tools" et d'ajouter une Vue comme on le fait pour n'importe quel autre contrôle. N'oubliez pas de presser le bouton "Reset" avant de dire aux clients qu'ils peuvent se connecter.

***Je ne peux pas quitter un client HubNet.***

Vous devez forcer (obliger) le client à quitter. Avec Mac OS X, forcez l'application à quitter en sélectionnant "Forcer à quitter…" dans le menu "Apple". Avec Windows, pressez "Ctrl-Alt-Delete" pour ouvrir le Gestionnaire de tâches, sélectionnez "HubNet Client" et pressez "End Task".

***Mon ordinateur s'est« endormi » pendant une activité HubNet. Quand je le réveille, je reçois une erreur et HubNet ne veut plus fonctionner.***

Le serveur HubNet peut s'arrêter si l'ordinateur se met en pose. Si cela se produit, quittez l'application NetLogo et relancez-la. Modifiez les réglages de votre ordinateur de manière à ce qu'il ne s'endorme plus.

***Mon problème n'est pas abordé dans cette page.***

Envoyez-nous un courrier électronique décrivant le problème à l'adresse feedback@ccl.northwestern.edu.

### Limitations connues

Si HubNet fonctionne mal, envoyez-nous un message à l'adresse bugs@ccl.northwestern.edu.

**Notez s'il vous plaît que** :

- ✔ HubNet n'a pas encore été testé intensivement avec un grand nombre de clients (avec plus de 25 clients environ). Des choses inattendues peuvent se produire avec un nombre de clients plus élevé.
- ✔ Les conditions de débordement de mémoire ne sont pas gérées avec élégance.
- ✔ L'envoi de grandes quantités de messages aux traceurs des clients peut prendre beaucoup de temps.
- ✔ NetLogo ne gère pas les clients malveillants de manière robuste (en d'autres mots, il est assez vulnérables aux attaques de type "denial-of-service" [refus de prise en charge ?]).
- ✔ Les performances ne se dégradent pas avec « élégance » avec des réseaux lents ou instables.
- ✔ Si vous êtes sur un réseau sans fil ou sub-LAN, l'adresse IP affichée dans le Centre de contrôle HubNet n'est pas toujours l'adresse IP complète du serveur.
- ✔ Le programme Computer HubNet n'a été testé que sur des réseaux LAN, pas sur des réseaux à connections téléphoniques ou WAN.

## Le "Calculator HubNet"

### Calculator HubNet pour TI-Navigator

Le "TI-Navigator Classroom Learning System" est un réseau sans fil de classe pour les calculatrices graphiques TI. Les utilisateurs de TI-Navigator peuvent installer une extension NetLogo libre qui s'intègre au "TI-Navigator" et permet aux calculatrices de fonctionner en tant que clients pour des simulations participatives telles que celles disponibles pour l'application Computer HubNet. L'extension "Calculator HubNet" est mise à disposition par Inquire Learning, LLC, en collaboration avec Texas Instruments. Inquire Learning offre aussi le support, le matériel didactique et un développement professionnel pour le système Calculator HubNet. Pour plus d'informations sur le système TI-Navigator lui-même, visitez le site web de Texas Instruments à l'adresse http://education.ti.com/navigator. Pour plus d'informations sur l'extension Calculator HubNet pour TI-Navigator, contactez Inquire Learning à l'adresse calc-hubnet@inquirelearning.com ou visitez http://www.inquirelearning.com/calc-hubnet.html.

## Ateliers pour enseignants

Pour des informations concernant des ateliers à venir et l'utilisation de NetLogo et de HubNet en classe, prenez contact avec nous à l'adresse feedback@ccl.northwestern.edu.

## Guide de programmation HubNet

Pour apprendre à concevoir ou à modifier des activités HubNet, voir le Guide de programmation HubNet.

## Obtenir de l'aide

Si vous avez des questions concernant les applications Computer HubNet ou Calculator HubNet ou si vous avez besoin d'aide pour débuter, envoyez-nous un courrier électronique à l'adresse feedback@ccl.northwestern.edu.

# Chapitre 19

# Guide de programmation HubNet

Ce guide contient tout ce que vous devez savoir pour comprendre et modifier le code des activités HubNet existantes ou pour développer vous-même de nouvelles activités HubNet. Il présuppose que vous connaissez le fonctionnement des activités HubNet, les bases de la programmation en NetLogo et l'utilisation (du point de vue du programmeur) des éléments de l'interface de NetLogo. Pour des informations générales concernant HubNet et son utilisation, consultez le Guide HubNet.



## Généralités concernant HubNet

Les informations présentées dans cette section sont spécialement destinées à ceux qui utilisent les clients de type ordinateur (computer clients), cependant, la plus grande partie du code présenté ici peut être réutilisée, moyennant quelques petites modifications, pour travailler avec des clients de type calculatrice (*calculator clients*).

## Programmer des activités HubNet

De nombreuses activités HubNet utilisent les mêmes portions de code. En font partie le code utilisé pour installer le réseau et le code utilisé pour recevoir et envoyer des informations aux clients. Si vous comprenez ce code, vous devriez être capables de modifier sans trop de difficultés les activités existantes et vous devriez également posséder de bonnes bases pour développer vos propres activités. Pour vous mettre le pied à l'étrier, nous avons fourni un modèle appelé `Template` (dans "File/Models Library/HubNet Computer Activities/Code Examples") qui contient les composants de base que l'on trouve dans la majorité des activités HubNet. Vous devriez pouvoir utiliser ce programme comme point de départ pour la plupart de vos nouveaux projets.

## Initialisation du réseau

Pour transformer un modèle NetLogo en activité HubNet, vous devez tout d'abord initialiser le réseau. Pour la plupart des activités HubNet, vous utiliserez la procédure `startup` pour initialiser le réseau. `startup` est une procédure spéciale que NetLogo essaie d'exécuter chaque fois qu'on ouvre un modèle. Ce qui en fait un bon endroit pour y mettre le code que vous ne voulez exécuter qu'une seule fois au lancement du programme (peu importe le nombre de fois que l'utilisateur fasse tourner la simulation du modèle). Pour HubNet, nous mettons les commandes qui initialisent le réseau dans `startup` parce qu'une fois le réseau monté, nous n'avons plus besoin de le refaire. Il faut tout d'abord spécifier le type de clients en utilisant `hubnet-set-client-interface`. Dans notre cas, nous utiliserons des clients de type ordinateur :

```
hubnet-set-client-interface "COMPUTER" []
```

Puis il faut initialiser le système avec `hubnet-reset`, qui demandera un nom de session à l'utilisateur et ouvrira le Centre de contrôle HubNet "HubNet Control Center". NetLogo est maintenant prêt à écouter les messages des clients.

Maintenant que le réseau est complètement installé, vous n'aurez plus besoin de vous soucier d'appeler à nouveau les procédures `hubnet-set-client-interface` ou `hubnet-reset`. Jetez un coup d'oeil à la procédure `setup` du modèle `template` :

```
to setup
 cp
 cd
 clear-output
 ask turtles
 [ set step-size 1
   hubnet-send user-id "step-size" step-size
 ]
end
```

Elle ressemble en grande partie à la plupart des autres procédures `setup`, toutefois vous devez noter qu'elle ne fait pas appel à `clear-all`. Dans ce modèle, et dans la grande majorité des activités HubNet de la Bibliothèque des modèles, nous avons une race de tortues qui représente les clients actuellement connectés. Dans notre cas, nous avons nommé cette race `students`. Chaque fois qu'un client se connecte, nous créons un agent `student` et enregistrons dans une variable tortue toutes les informations concernant ce client, informations dont nous pourrions avoir besoin par la suite. Comme nous ne voulons pas demander à l'utilisateur de se déconnecter et de se re-connecter chaque fois que nous initialisons l'activité, nous ne devons pas tuer toutes les tortues, par contre nous voulons redonner aux variables leurs valeurs initiales et notifier aux clients toutes les modifications que nous avons faites (nous reviendrons sur ce sujet par la suite).

## Recevoir les messages des clients

Au cours de l'activité, des données seront transférées entre les clients HubNet et le serveur. La plupart des activités HubNet appellent, dans une boucle `go`, une procédure qui écoute si de nouveaux messages, en provenance des clients, arrivent. Dans notre cas, cette procédure est appelée `listen clients` :

```
to listen-clients
  while [ hubnet-message-waiting? ]
  [ hubnet-fetch-message
    ifelse hubnet-enter-message
    [ create-new-student ]
    [ ifelse hubnet-exit-message?
      [ remove-student ]
      [ execute-command hubnet-message-tag ]
    ]
  ]
end
```

Aussi longtemps qu'il y a des messages dans la queue (la pile), cette boucle prend et traite ces messages les uns après les autres (autrement dit un seul à la fois, dans l'ordre où ils sont arrivés dans la pile). `hubnet-fetch-message` prend le message suivant dans la queue, en fait le message courant et donne aux reporters `hubnet-message-source`, `hubnet-message-tag` et `hubnet-message` les informations (valeurs) qui les concernent. Les clients envoient des messages quand les utilisateurs se connectent et se déconnectent, ce qui se produit chaque fois qu'ils manipulent l'un des éléments de l'interface, c'est-à-dire pressent un bouton, déplacent un curseur, cliquent dans la vue, etc. Nous parcourons chacun de ces messages et décidons de l'action à entreprendre en fonction du type de message (entrer, quitter, ou autre), de sa partie `hubnet-message-tag` (le nom de l'élément de l'interface actionné) et de sa partie `hubnet-message-source` (le nom du client qui a envoyé le message).

Pour chaque message entrant, nous créons une tortue portant le nom d'utilisateur (stocké dans une variable `user-id`) correspondant au contenu de `hubnet-message-source`. Ce nom est celui que chaque utilisateur a dû donner pour entrer dans l'activité. Cette information est unique.

```
to create-new-student
  create-students 1
  [
    set user-id hubnet-message-source
    set label user-id
    set step-size 1
    send-info-to-clients
  ]
end
```

Arrivés à ce point, nous mettons toutes les autres variables client à leurs valeurs par défaut et les envoyons aux clients si nécessaire. Nous déclarons une variable `students-own` pour chaque élément de l'interface du client qui contient une valeur d'état, c'est-à-dire tout ce qui serait une variable globale sur le serveur, autrement dit les valeurs des curseurs (Sliders), des sélecteurs (Choosers), des commutateurs (Switches) et des boîtes d'entrées (Input boxes). Il est important de s'assurer que ces variables restent synchronisées avec les valeurs visibles sur l'interface du client.

Quand un client se déconnecte, il envoie au serveur un message de sortie (*exit*) qui nous donne la possibilité de nettoyer (supprimer) toutes les informations que nous avions stockées concernant ce client, dans ce cas il vaut mieux tuer (`die`) la tortue correspondant à ce client.

```
to remove-student
  ask students with [user-id = hubnet-message-source]
  [ die ]
end
```

Tous les autres messages proviennent des éléments de l'interface identifiés par `hubnet-message-tag`, identification qui correspond au nom de l'élément tel qu'il apparaît sur l'interface du client. Chaque fois qu'un élément de l'interface change, un message est envoyé au serveur. À moins que vous ne la stockiez, la valeur actuellement affichée sur l'interface du client ne sera plus accessible par d'autres parties du programme (du modèle). C'est la raison pour laquelle nous avons déclaré une variable `students-own` pour chaque élément d'interface qui a un état (curseurs, commutateurs, etc.). Quand nous recevons un message de ce client, nous affectons le contenu de ce message à la variable tortue correspondante :

```
if hubnet-message-tag = "step-size"
[
   ask students with [user-id = hubnet-message-source]
   [ set step-size hubnet-message ]
]
```

Puisque aucune donnée n'est associée aux boutons, ces derniers ne sont généralement pas associés à des variables tortues, mais servent avant tout à indiquer une action effectuée par le client. Comme c'est le cas avec les boutons habituels, il y a souvent une procédure associée à chaque bouton de l'interface, procédure que

vous appelez chaque fois qu'un message reçu indique que le bouton a été pressé. Bien que ce ne soit certainement pas une obligation, cette procédure est souvent une procédure tortue, c'est-à-dire quelque chose que la tortue de l'étudiant qui est à l'origine du message peut exécuter :

```
if command = "move left"
[ set heading 270
  fd 1 ]
```

## Envoyer des messages aux clients

Comme mentionné précédemment, vous pouvez aussi envoyer des données à n'importe quel élément de l'interface du client qui affiche des informations : les moniteurs (Monitors), les curseurs (Cursors), les sélecteurs (Choosers) et les boîtes d'entrées (Input boxes). (Notez que les traceurs de courbes et la Vue sont des cas spéciaux qui seront traités dans leur propre section.) Deux primitives permettent d'envoyer des informations, ce sont : `hubnet-send` et `hubnet-broadcast`. `hubnet-broadcast` envoie l'information à tous les clients alors que `hubnet-send` n'envoie l'information qu'à un client particulier ou qu'à un groupe de clients sélectionnés. Comme indiqué plus haut, rien chez le client ne se met à jour automatiquement. Si une valeur change sur le serveur, il est de votre responsabilité, en tant qu'auteur du modèle, de mettre à jour les moniteurs du client et si vous modifiez sur le serveur une variable qui est associée à un autre élément de l'interface (et non en tant que réponse à un message de modification de la part de cet élément), vous devez aussi mettre à jour cette valeur chez le client. Prenons l'exemple où vous avez un curseur du client appelé `step-size` et un moniteur appelé `Step Size` (notez que ces deux noms doivent êtres différents), vous **devez** écrire le code de mise à jour de la manière suivante :

```
if hubnet-message-tag = "step-size"[
  ask student with [ user-id = hubnet-message-source ]
  [
    set step-size hubnet-message
    hubnet-send user-id "Step Size" step-size
  ]
]
```

Vous pouvez envoyer le type de données que vous voulez : des nombres, des chaînes de caractères, des listes, des listes de listes. Toutefois, si la donnée n'est pas appropriée à l'élément de l'interface visé, (par exemple si vous envoyez une chaîne à un curseur), le message sera ignoré. Voici quelques exemples de code pour différents types de données :

| **Type de données** | **Exemple avec** `hubnet-broadcast` | **Exemple avec** `hubnet-send` |
|---|---|---|
| nombre | `hubnet-broadcast "A" 3.14` | `hubnet-send "jimmy" "A" 3.14` |
| chaîne | `hubnet-broadcast "STR1" "HI THERE"` | `hubnet-send ["12" "15"] "STR1" "HI THERE"` |
| liste de nombres | `hubnet-broadcast "L2" [1 2 3]` | `hubnet-send hubnet-message-source "L2" [1 2 3]` |
| matrice of nombres | `hubnet-broadcast "[A]" [[1 2] [3 4]]` | `hubnet-send "susie" "[A]" [[1 2] [3 4]]` |
| liste de chaînes (seulement pour Computer HubNet) | `hubnet-broadcast "user-names" [["jimmy" "susie"] ["bob" "george"]]` | `hubnet-send "teacher" "user-names" [["jimmy" "susie"] ["bob" "george"]]` |

## Exemples

Étudiez les modèles placés dans les dossiers "HubNet Computer Activities" et "HubNet Calculator Activities" de la Bibliothèque des modèles pour voir, dans le panneau "Procedures", comment ces primitives sont utilisées en pratique. Le modèle "Disease" est idéal pour commencer.

# Informations spécifiques à Calculator HubNet

Pour des informations sur la programmation d'activités HubNet utilisant des clients de type calculatrice, contactez-nous.

# Informations spécifiques à Computer HubNet

Les informations suivantes sont spécifiques à Computer HubNet

## Comment construire une interface côté client

Ouvrez l'éditeur de client HubNet "HubNet Client Editor" à partir du menu "Tools". Ajoutez dans l'interface les boutons "Button", curseurs "Sliders", commutateurs "Switch", moniteurs "Monitor", traceurs de courbes "Plot", sélecteurs "Selector", sorties texte "Output" ou notes "Note" dont vous avez besoin, comme vous le feriez pour une interface d'un modèle normal. Vous verrez que les informations que vous avez à fournir pour chaque type d'éléments sont légèrement différentes de celles à fournir pour ces mêmes éléments placés dans le panneau "Interface" d'un modèle normal. Les contrôles côté client n'interagissent pas avec le modèle de la même manière. Plutôt que d'avoir un lien direct avec les commandes et les reporters, ces contrôles envoient des messages au serveur et c'est là que le modèle détermine comment ces messages affectent son comportement. Tous les contrôles coté clients ont un marqueur (un *tag*) qui est un nom identifiant le contrôle de manière univoque. Quand le serveur reçoit un message de ce contrôle, le tag se trouve dans `hubnet-message-tag`

Par exemple, si vous avez un bouton appelé "move left", un curseur appelé "step-size", un commutateur appelé "all-in-one-step?" et un moniteur appelé "Location:", les tags de ces éléments d'interface seront les suivants :

| Élement de l'interface | tag |
|---|---|
| move left | move left |
| step-size | step-size |
| all-in-one-step? | all-in-one-step? |
| Location: | Location: |

Notez que chaque élément de l'interface doit avoir son propre nom qui **doit être différent** pour chacun. Avoir plus d'un élément d'interface portant le même nom dans une interface côté client conduit à des comportements imprévisibles puisqu'il n'est pas clairement défini de quel élément provient l'information, donc à quelle variable envoyer cette information.

## Mises jour de la Vue côté client

La manière la plus simple d'afficher le monde NetLogo dans la Vue des clients est d'utiliser le « dédoublement » de la vue "*view mirroring*". Quand le dédoublement de la Vue est activé (en cochant la case "Mirror 2D view on clients" dans le Centre de contrôle HubNet, la Vue des clients est mise à jour automatiquement pour refléter l'état actuel du monde. Tous les clients auront la même Vue, qui est aussi celle qui s'affiche dans le panneau "Interface" du serveur. Les mises à jour de la Vue sont déclenchées à peu près cinq fois par seconde, ce qui signifie qu'il peut y avoir un grand nombre de messages à envoyer aux clients. Si vous avez des problèmes de performances, essayer d'utiliser les procédures `no-display` et `display` pour réduire le nombre de mises à jour envoyées aux clients.

Si vous aimeriez avoir un contrôle plus poussé des mises à jour de la Vue, vous pouvez utiliser les commandes `hubnet-broadcast-view` et `hubnet-send-view` pour envoyer explicitement la Vue soit à tous les clients avec `hubnet-broadcast-view`, soit à un ou des clients particuliers avec `hubnet-send-view`. Toutefois, vous devez noter que le dédoublement de la vue ne met à jour que les informations qui ont changé depuis la dernière mise à jour, alors que ces deux commandes envoient toutes les informations du monde à chaque appel. Cela signifie que ces messages peuvent être plus volumineux (bien plus volumineux) et que par conséquent la mise à jour est ralentie.

Si la Vue côté client n'existe pas ou si la case "Mirror 2D view on clients" dans Le Centre de contrôle HubNet n'est pas cochée, aucun message de rafraîchissement de la Vue n'est envoyé aux clients.

**Note 1** : `hubnet-broadcast-view` et `hubnet-send-view` étant des primitives expérimentales, leur comportement pourrait changer dans les futures versions de HubNet.

**Note 2** : Certaines des fonctionnalités de la Vue de NetLogo ne sont pas encore implémentées dans les clients HubNet, par exemple l'« enroulement » (*wrapping*) de la Vue et les points de vue (perspectives) de l'observateur.

## Cliquer dans la Vue du client

Si l'interface côté client possède une Vue, un message est envoyé au serveur chaque fois que l'utilisateur clique dans la Vue. Le marqueur (*tag*) de ce message est "`View`" et le message lui-même consiste en une liste de deux éléments, les coordonnées X et Y du point cliqué. Par exemple, pour colorer en rouge le patch cliqué par un client, vous utiliserez le code NetLogo suivant :

```
if hubnet-message-tag = "View"
[
 ask patches with [ pxcor = (round item 0 hubnet-message) and
 pycor = (round item 1 hubnet-message) ]
 [ set pcolor red ]
]
```

## Mises à jour des traceurs côté clients

Si la mise à jour des traceurs des clients est activée (case à cocher "Mirror plots on clients (experimental)" du Centre de contrôle HubNet est activée), qu'un traceur du modèle NetLogo subit une modification et qu'un traceur ayant exactement le même nom existe dans l'interface des clients, un message comprenant ces modifications est envoyé aux clients, message qui demande au traceur du client de faire les mêmes modifications. Par exemple, supposons que nous ayons un modèle HubNet possédant un traceur nommé `Milk Supply` dans NetLogo et chez les clients, que ce traceur soit le traceur courant dans NetLogo, et que vous écriviez l'instruction suivante dans le Centre de commande :

```
plot 5
```

cette commande provoquera l'envoi d'un message à tous les clients leur demandant de dessiner un point de coordonnée Y = 5 à la prochaine position du crayon du traceur (le long de l'axe X). Notez que si vous dessinez beaucoup avec les graphiques d'un seul coup, cette activité peut générer une grande quantité de messages graphiques à envoyer aux clients.

# Chapitre 20

# Logging

L'outil NetLogo appelé Logging (journal de bord) permet au directeur d'exercice d'enregistrer les actions des étudiants pour une analyse ultérieure.

Une fois lancé et initialisé, l'outil Logging est invisible pour l'utilisateur. L'enseignant peut choisir le type d'événements à enregistrer au moyen d'un fichier de configuration.

La totalité des informations recueillies lors d'une session NetLogo est appelée **journal de bord**. Ce journal est constitué des **rapports** qui sont envoyés chaque fois qu'un étudiant effectue une des manoeuvres spécifiées dans le fichier de configuration. Le coeur de ce fichier est constitué de **rapporteurs** (`"loggers"`) qui spécifient les événements à surveiller, chaque rapporteur envoyant un type de rapport particulier.

NetLogo utilise le paquet de logiciels log4j pour son travail de surveillance et d'enregistrement. Si vous avez déjà travaillé avec cet outil, vous ne serez pas dépaysés avec le Logging de NetLogo.

## Démarrer l'enregistrement

La marche à suivre dépend du système d'exploitation utilisé.

### Mac OS X ou Windows

Le répertoire NetLogo contient un lanceur spécial appelé `NetLogo Logging`. Double-cliquez sur son icône.

Avec Windows, le répertoire `NetLogo` se trouve à l'adresse `C:\Program Files`, à moins que vous n'ayez choisi un autre emplacement au moment de l'installation de NetLogo.

### Linux et autres

Pour activer Logging, appelez le script `netlogo.sh` de la manière suivante :

```
netlogo.sh --logging netlogo_logging.xml
```

Vous pouvez aussi modifier le script pour y inclure ces drapeaux, ou copier le script et modifier la copie (c'est plus prudent).

Vous pouvez remplacer le fichier `netlogo_logging.xml` par tout fichier de configuration log4j valide écrit en XML, fichier dont nous discuterons plus loin les détails.

# Utiliser Logging

Quand NetLogo démarre, il ouvre une boîte de dialogue qui demande un nom d'utilisateur. Ce nom apparaîtra dans tous les rapports générés au cours de la session.

## Où les rapports sont-il stockés ?

Les rapports sont stockés dans le répertoire temp spécifique au système d'exploitation. Sur la plupart des systèmes de type Unix (y compris MacOS), il s'agit du répertoire `temp`. Sur Windows XP, les rapports peuvent être trouvés dans le répertoire `c:\Documents and Settings\<user>\Local Settings\Temp`, où `<user>` est le nom de l'utilisateur donné au démarrage, et sur Windows Vista, ces journaux se trouvent dans `c:\Users\<user>\AppData\Local\Temp`.

Deux commandes pratiques vont vous aider à gérer ces rapports. `__zip-log-files` *`filename`* rassemble tous les rapports présents dans le répertoire temp et les comprime dans un fichier zip, sauvegardé à l'emplacement spécifié. Après l'exécution de `__zip-log-files`, les rapports existants ne sont **pas** effacés, mais vous pouvez le faire explicitement avec la commande `__delete-log-files`.

Le tableau suivant décrit les noms de tous les rapports enregistrables, le type des événements enregistrés dans chaque rapport, à quel niveau, ainsi qu'un exemple de sortie utilisant XMLLayout. Tous les rapports se trouvent dans le fichier `org.nlogo.api.Logger`. Quand vous faites référence aux types de rapports dans le fichier de configuration, vous devez utiliser le nom complet. Ainsi, le rapport `GLOBALS` doit être désigné par `org.nlogo.api.Logger.GLOBALS`.

| Rapport | Événement | Niveau | Exemple |
|---|---|---|---|
| `GLOBALS` | une variable globale est modifiée | info, debug | `<event logger="org.nlogo.api.Logger.GLOBALS" timestamp="1177341065988" level="INFO" type="globals"> <name>FOO</name> <value>51.0</value> </event>` |
| `GREENS` | des curseurs, commutateurs, sélecteurs, boîtes d'entrée ont été modifiés via l'interface | info | `<event logger="org.nlogo.api.Logger.GREENS" timestamp="1177341065988" level="INFO" type="slider"> <action>changed</action> <name>foo</name> <value>51.0</value> <parameters> <min>0.0</min> <max>100.0</max> <inc>1.0</inc> </parameters> </event>` |
| `CODE` | le code est compilé, y compris: centre de commande, panneau "Procedures", limites des curseurs et boutons | info | `<event logger="org.nlogo.api.Logger.CODE" timestamp="1177341072208" level="INFO" type="command center"> <action>compiled</action> <code>crt 1</code> <agentType>O</agentType> <errorMessage>success</errorMessage> </event>` |

| Rapport | Événement | Niveau | Exemple |
|---|---|---|---|
| `WIDGETS` | un objet (*widget*) a été ajouté ou retiré de l'interface | info | `<event logger="org.nlogo.api.Logger.WIDGETS" timestamp="1177341058351" level="INFO" type="slider"> <name></name> <action>added</action> </event>` |
| `BUTTONS` | un bouton a été pressé ou relâché | info | `<event logger="org.nlogo.api.Logger.BUTTONS" timestamp="1177341053679" level="INFO" type="button"> <name>show 1</name> <action>released</action> <releaseType>once</releaseType> </event>` |
| `SPEED_SLIDER` | le curseur vitesse a été modifié | info | `<event logger="org.nlogo.api.Logger.SPEED" timestamp="1177341042202" level="INFO" type="speed"> <value>0.0</value> </event>` |
| `TURTLES` | des tortues sont mortes ou sont nées | info | `<event logger="org.nlogo.api.Logger.TURTLES" timestamp="1177341094342" level="INFO" type="turtle"> <name>turtle 1</name> <action>born</action> <breed>TURTLES</breed> </event>` |
| `LINKS` | des liens sont morts ou sont nés | info | `<event logger="org.nlogo.api.Logger.LINKS" timestamp="1177341094347" level="INFO" type="link"> <name>link 0 1</name> <action>born</action> <breed>LINKS</breed> </event>` |

## Comment configurer la sortie du journal

Le fichier de configuration par défaut du journal (`netlogo_logging.xml`) ressemble à peu près à celui montré ci-dessous.

NetLogo définit 8 rapporteurs, descendant tous du rapporteur racine, ce qui signifie, à moins que vous ne spécifiiez des propriétés (appender, layout et output level) dans le fichier de configuration, propriétés que ces rapporteurs hériteront de la racine. Dans le fichier de configuration par défaut, la racine est placée au niveau INFO, l'appender est `org.nlogo.api.XMLFileAppender` et le layout est `org.nlogo.api.XMLLayout`. Ensemble, ils génèrent un fichier XML joliment formaté comme défini dans le fichier `netlogo_logging.dtd` qui est lui-même basé sur le dtd `log4j`. Si l'appender est un FileAppender (incluant XMLFileAppender), un nouveau fichier est créé chaque fois que l'utilisateur ouvre un modèle.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration debug="false" xmlns:log4j='http://jakarta.apache.org/log4j/'>

 <appender name="A1" class="org.nlogo.api.XMLFileAppender">
 <layout class="org.nlogo.api.XMLLayout"/>
 </appender>

 <category name="org.nlogo.api.Logger.WIDGETS">
 <priority value="off" />
 </category>

 <category name="org.nlogo.api.Logger.TURTLES">
 <priority value="off" />
 </category>

 <category name="org.nlogo.api.Logger.LINKS">
 <priority value="off" />
 </category>

 <root>
 <priority value ="info" />
 <appender-ref ref="A1" />
 </root>
</log4j:configuration>
```

Ce fichier définit d'abord une appender `"A1"` de type `XMLFileAppender` avec un XMLLayout. L'appender définit l'endroit où les données fournies par les rapporteurs doivent aller, ici dans un fichier. En réalité, si NetLogo reçoit un FileAppender, il commence automatiquement un nouveau fichier chaque fois que l'utilisateur ouvre un nouveau modèle. Le FileAppender fait aussi quelques formatages et écrits les en-têtes appropriées dans le fichier. Le layout définit comment écrire chaque message individuel. À moins que vous ne soyez un utilisateur avancé, il n'y a aucune raison de modifier (ou de se soucier) de l'appender et du layout.

Notez la définition du rapporteur de base à la fin du fichier de configuration. Tous les autres rapporteurs descendent de ce rapporteur et, de ce fait, héritent des propriétés de cette racine, à moins qu'il n'en ait été spécifié autrement. Ce cas est remarquablement simple, car, ayant spécifié l'appender `A1`, nous en faisons l'appender par défaut pour la racine (et pour tous les autres rapporteurs) et spécifions `"INFO"` comme propriété par défaut. Les messages qui sont rapportés au niveau `INFO` ou à un niveau supérieur seront notés, ceux qui le sont à un niveau inférieur ne le sont pas. Notez qu'à une seule exception près, NetLogo rapporte toujours au niveau `INFO`. Des réglages au niveau global qui ne changent pas la valeur des variables globales sont reportées au niveau `DEBUG`. Ce qui signifie que ces messages sont désactivés par défaut puisque le niveau `DEBUG` est inférieur au niveau `INFO`. Le reste du corps du fichier de configuration supplante les propriétés du rapporteur racine dans quelques rares rapporteurs spécifiques (ou catégories puisqu'elles sont connues dans le fichier de configuration, ces termes pouvant être considérés comme synonymes dans le cas de ce document). Il s'ensuit qu'il désactive les rapporteurs `WIDGET`, `TURTLES` et `LINKS` par défaut. Pour les réactiver, vous devez remplacer la valeur `off` de `priotity` par la valeur `info`, de la manière suivante :

```
 <category name="org.nlogo.api.Logger.TURTLES">
  <priority value="info" />
 </category>
```

On peut aussi ôter toute la référence à la catégorie dans le fichier, car elle ne sert à rien d'autre.

## Configuration avancée

Ce qui précède n'est qu'une introduction sommaire aux fichiers de configuration Logging de NetLogo. Il y a bien d'autres options de configuration disponibles dans le canevas de log4j. Voir la documentation log4j.

# Chapitre 21

# Guide de Controlling

NetLogo peut être appelé à partir d'un autre programme Java et contrôlé par ce programme. Par exemple, vous pourriez vouloir appeler NetLogo à partir d'un petit programme qui fait quelque chose de simple, comme par exemple automatiser une série de simulations d'un modèle.

Ce chapitre du Manuel de l'utilisateur présente cette possibilité aux programmeurs Java. Nous supposerons donc que vous connaissez le langage Java, ses outils et ses pratiques.

**Note** : l'outil Controlling est encore « expérimental ». Il est susceptible de continuer à changer, évoluer et croître encore. Le code que vous écrivez pour l'utiliser aujourd'hui devra peut-être être mis à jour pour continuer de fonctionner demain avec les futures versions de NetLogo.



Le dossier `javadoc` qui contient les Spécifications de l'API NetLogo contient tous les détails nécessaires (non traduit en français).

## Lancer une Machine Virtuelle Java pour NetLogo

NetLogo fait plusieurs suppositions concernant la Machine Java Virtuelle (JVM) qui le fait fonctionner. C'est pourquoi il y a certains arguments qui doivent être passés au démarrage de la JVM.

### Options recommandées pour NetLogo avec GUI et sans GUI (*headless*)

- ✔ `-server` Utiliser un serveur VM pour de plus hautes performances.
- ✔ `-Xmx1024m` Utiliser jusqu'à `1` GBytes de mémoire pour la mémoire réservée à Java VM. Vous devrez peut-être augmenter cette limite pour faire tourner certains modèles.

### Options supplémentaires recommandées pour la GUI seulement

- ✔ `-XX:MaxPermSize=128m` Empêcher Java de déborder de sa mémoire lors de la compilation répétée d'un modèle avec un code très long.
- ✔ `-Djava.ext.dir=` Ignorer toutes bibliothèques natives existantes dans le système. Ceci afin d'éviter tout conflit avec d'autres versions de JOGL. Vous devrez peut-être laisser cette option de côté, ou la modifier pour pointer vers vos bibliothèques natives si vous utilisez des extensions de la Java VM.
- ✔ `-Djava.library.path=./lib` Non nécessaire avec les systèmes Mac ou Windows; peut être nécessaire avec d'autres systèmes d'opération tels que Linux. Faire en sorte que NetLogo puisse trouver les bibliothèques natives pour JOGL et d'autres extensions. Si vous ne démarrez pas la JVM dans le niveau supérieur du répertoire NetLogo, alors `./lib` doit être modifié pour pointer vers le sous-répertoire `lib` de l'installation NetLogo.

### Répertoire de travail courant

L'application NetLogo part du principe qu'au démarrage du programme, le répertoire de travail courant est au niveau supérieur de l'installation NetLogo.

## Exemple (avec GUI)

Voici un programme, petit mais complet, qui fait démarrer toute l'application NetLogo, ouvre un modèle, déplace un curseur, spécifie une amorce pour le générateur de nombres aléatoires, lance la simulation sur 50 pas (*ticks*) et imprime le résultat :

```
import org.nlogo.app.App;
import java.awt.EventQueue;

public class Example1 {
    public static void main(String[] argv) {
        App.main(argv);
        try {
            EventQueue.invokeAndWait
                ( new Runnable()
                    { public void run() {
                        try {
                          App.app.open
                              ("models/Sample Models/Earth Science/"
                               + "Fire.nlogo");
                        }
                        catch( java.io.IOException ex ) {
                          ex.printStackTrace();
                        }
                    } } );
            App.app.command("set density 62");
            App.app.command("random-seed 0");
            App.app.command("setup");
            App.app.command("repeat 50 [ go ]");
            System.out.println
                (App.app.report("burned-trees"));
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Pour pouvoir compiler et faire tourner ce programme, le fichier `NetLogo.jar` (se trouvant dans la distribution NetLogo) doit se trouver dans le chemin des classes (*classpath*). De plus, le répertoire `lib` (aussi dans la distribution NetLogo) doit aussi se trouver au même endroit; il contient des bibliothèques supplémentaires utilisées par `NetLogo.jar`.

Notez l'utilisation de `EventQueue.invokeAndWait` pour être sûr qu'une méthode est appelée par le bon *thread*. Ceci parce que la plupart des méthodes de la classe `App` ne peuvent être appelées que par certains *threads*. La plupart des méthodes ne peuvent être appelées *que* par le thread de la file d'événements AWT; mais quelques rares méthodes, telles que `commmand()`, ne peuvent être appelées que par des threads *autres* que ceux de la file AWT (tel que, dans cet exemple, le thread principal `main`).

Plutôt que de continuer à discuter de cet exemple en détail, nous vous renvoyons aux Spécifications de l'API NetLogo qui détaillent tous les tenants et aboutissants des classes et des méthodes utilisées ci-dessus. Et il y a encore bien d'autres méthodes à disposition.

## Exemple (*headless*) [sans GUI]

Dans ce cas, le code de l'exemple ressemble fort à celui de l'exemple précédent, mais les méthodes utilisées pour remplacer celles, statiques, de `App`., sont issue d'une instance de la classe `HeadlessWorkspace`

```
import org.nlogo.headless.HeadlessWorkspace;

public class Example2 {
    public static void main(String[] argv) {
        HeadlessWorkspace workspace =
            new HeadlessWorkspace() ;
        try {
            workspace.open
                ("models/Sample Models/Earth Science/"
                 + "Fire.nlogo");
            workspace.command("set density 62");
            workspace.command("random-seed 0");
            workspace.command("setup");
            workspace.command("repeat 50 [ go ]") ;
            System.out.println
                (workspace.report("burned-trees"));
            workspace.dispose();
        }
        catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Pour pouvoir compiler et faire tourner ce programme, le fichier `NetLogo.jar` ou le fichier `NetLogoLite.jar` (se trouvant tous deux dans la distribution NetLogo) doivent être dans le chemin des classes (*classpath*). (Le second fichier `.jar` est plus petit, mais ne peut mener que des opérations *headless*, il lui manque bien des choses pour mener des opérations GUI complètes.) Le répertoire `lib` (aussi dans la distribution NetLogo), contenant des bibliothèques supplémentaires nécessaires, doit aussi être présent. Quand le programme tourne dans un contexte qui ne supporte pas un affichage graphique, la propriété système `java.awt.headless` doit être `true` (vraie) afin d'obliger Java à fonctionner en mode *headless*. `HeadlessWorkspace` initialise automatiquement cette propriété pour vous.

Puisqu'il n'y a pas d'interface utilisateur graphique (GUI), les primitives NetLogo qui envoient normalement des sorties au Centre de commande ou à la zone de sortie de l'interface les envoient maintenant à la sortie standard. Toutefois, la primitive `export-world` peut toujours être utilisée pour sauvegarder l'état du modèle. `export-view` fonctionne toujours pour écrire un fichier image contenant une prise de vue (sinon non visible) de la Vue 2D. La méthode `report()` est utilisée pour obtenir des résultats du modèle et les transférer dans le code Java (autrement dit dans le programme appelant).

Le fichier généré par la primitive `export-world` comprend toutes les données de tous les traceurs de courbes. Vous pouvez exporter les contenus des traceurs individuellement au moyen de la primitive `export-plot`.

Vous pouvez créer plusieurs instances de `HeadlessWorkspace`. Elles fonctionneront dans des threads indépendants et n'interférerons pas les unes avec les autres.

Il y a quelques restrictions quand NetLogo fonctionne en mode headless :

- Les primitives `movie-*` ne sont pas disponibles. Tenter de les utiliser génère une exception Java.
- Les primitives `user-*` qui demandent une entrée de la part de l'utilisateur, telles que `user-yes-or-no` génèrent une exception Java.

Les Specification de l'API NetLogo (non traduites) contiennent d'autres détails.

## Espace de comportements (BehaviorSpace)

L'API Controlling permet de mener des expérimentations de l'Espace de comportements (BehaviorSpace) sans interface utilisateur graphique (*headless*). (Il ne permet pas de les faire tourner dans l'interface graphique de BehaviorSpace, mais vous pouvez écrire votre propre code Java « type-BehaviorSpace » pour mener vos propres expérimentations « type-BehaviorSpace » si vous le désirez.)

Notez qu'il n'est en définitive pas nécessaire d'utiliser l'API pour faire fonctionner BehaviorSpace sans interface graphique (headless). BehaviorSpace headless est supporté directement par la Ligne de commandes de NetLogo sans nécessiter aucune programmation Java. Voir le chapitre Guide de BehaviorSpace pour des instructions détaillées.

Dans la plupart des cas, les possibilités offertes par la Ligne de commandes seront amplement suffisantes sans qu'il soit nécessaire d'utiliser l'API. Toutefois, dans certaines situations, vous aurez besoin de la souplesse supplémentaire offerte par cet API.

- `HeadlessWorkspace` comprend quatre méthodes pour faire tourner des expérimentations : trois variantes de `runExperiment` plus la méthode `runExperimentFromModel`.
- La méthode `runExperimentFromModel` est utilisée quand la description de l'expérimentation est déjà enregistrée dans le fichier du modèle.
- Les deux formes de `runExperiment` qui demandent des arguments `File` sont utilisées quand les descriptions de l'expérimentation sont enregistrées dans un fichier XML indépendant, séparé du fichier du modèle. Si ce fichier XML ne contient qu'une seule description, vous n'avez besoin de transmettre que l'objet `File`. Si ce fichier contient plusieurs descriptions, vous devez aussi transmettre un objet `String` qui contient le nom de l'expérimentation.
- La forme de `runExperiment` qui ne demande qu'un argument `String` (et un argument pour spécifier le format de sortie) est utilisée pour passer directement le fichier XML de la description de l'expérimentation.

Toutes ces méthodes demandent un `PrintWriter` en tant que destination pour les résultats. Si vous vous contentez de ne les envoyer qu'à la sortie standard, il vous suffit de passer `new java.io.PrintWriter( System.out )`.

Le Guide de BehaviorSpace explique comment construire des descriptions d'expérimentations en XML.

Les Spécifications de l'API NetLogo contiennent toutes les informations nécessaires concernant la classe `HeadlessWorkspace` et ses méthodes.

## Autres options

Quand votre programme Java contrôle NetLogo en utilisant la classe `App`, la totalité de l'application NetLogo est présente, y compris les panneaux, la barre des menus, et ainsi de suite. Cet arrangement est approprié au contrôle ou à la programmation d'un modèle NetLogo, mais non idéal pour inclure un modèle NetLogo dans une plus vaste application.

Nous avons aussi d'autres API similaires, indépendantes, qui permettent de n'inclure dans d'autres programmes que des parties de NetLogo, par exemple de ne mettre que les panneaux (et non toute la fenêtre), ou même seulement le contenu du panneau "Interface". Pour le moment, cet API supplémentaire n'est pas documenté. Si vous avez envie de l'utiliser, contactez-nous à l'adresse feedback@ccl.northwestern.edu.

## Conclusion

N'oubliez pas de consulter le dossier NetLogo API Specification pour une description complète de ces classes et méthodes.

Comme nous l'avons déjà mentionné, l'outil Controlling est encore expérimental. Cet API initial ne contient pas nécessairement tout ce dont vous auriez envie. Certaines fonctionnalités existent mais ne sont pas encore documentées. Donc, si vous ne trouvez pas la fonctionnalité dont vous rêvez, contactez-nous; nous pourrons très certainement vous aider à faire ce que vous voulez. S'il-vous-plaît, n'hésitez pas à nous contacter à l'adresse feedback@ccl.northwestern.edu pour nous poser des questions, car nous pourrons alors trouver un solution ou donner des informations supplémentaires pour étoffer la documentation là où elle est encore lacunaire.

# Chapitre 22

# Liaison avec Mathematica

## De quoi s'agit-il ?

La liaison NetLogo-Mathematica offre aux modélisateurs une liaison en temps réel facile à utiliser entre NetLogo et Mathematica. Ensemble, ces deux outils mettent à disposition de l'utilisateur un outil de travail combiné hautement interactif et auto-documentable que ni l'un ni l'autre ne peut offrir séparément.

Mathematica comporte toute une série d'outils dont ont besoin les utilisateurs de modèles à base d'agents tout au long de leur processus de recherches : possibilités d'importations avancées, fonctions statistiques, visualisation des données et création de documents. Grâce à la liaison NetLogo-Mathematica, vous pouvez utiliser tous ces outils en parallèle avec NetLogo.

Comme tous les documents (ou blocs-notes) Mathematica contiennent des commentaires, des images, des annotations et des objets interactifs, l'intégration de NetLogo et de Mathematica constitue une solution complète pour l'exploration de modèles complexes, aussi bien pour les étudiants que pour les chercheurs.

Les fonctionnalités de base de la liaison sont assez semblables à celles de l'API Controlling de NetLogo : vous pouvez charger des modèles, exécuter des commandes et importer des données calculées par NetLogo. Mais, contrairement à l'API Controlling qui est basé sur le langage Java, toutes les interactions avec la liaison sont interprétées, ce qui est idéal non seulement pour un développement rapide d'expérimentations personnalisées de type BehaviorSpace, mais aussi comme compagnon de NetLogo pour déverminer vos modèles.

Pour en savoir plus sur Mathematica, visitez le site de Wolfram Research.

## Que puis-je faire avec cet outil?

Voici quelques exemples de ce que vous pouvez faire avec la liaison Mathematica-NetLogo.

- ✔ Analyser le modèle en temps réel avec conversions de données transparentes dans les deux sens.
- ✔ Développer des présentations personnalisées et de haute qualité des données fournies par le modèle.
- ✔ Collecter des données détaillées de la simulation pour de grands espaces de paramètres multi-dimensionnels.
- ✔ Développer rapidement et interactivement des interfaces pour explorer le comportement des modèles.
- ✔ Avoir un accès direct aux ensembles de données avec les fonctions pré-programmées.

## Utiliser la Liaison NetLogo-Mathematica

Cette section est une brève introduction à l'utilisation de la Liaison NetLogo-Mathematica. Elle montre comment charger le paquet de logiciels Liaison NetLogo-Mathematica, démarrer NetLogo, donner des commandes et recueillir les données fournies par NetLogo.

**Charger les logiciels** : une fois la Liaison NetLogo-Mathematica installée, vous pouvez charger le paquet de logiciels en entrant le code suivant dans le bloc-note de Mathematica :

```
<<NetLogo
```

**Démarrer NetLogo à partir de Mathematica** : Pour démarrer votre session NetLogo dans Mathematica, écrivez ce qui suit dans votre bloc-note :

```
NLStart["votre chemin vers netlogo"];
```

ou "*votre chemin vers netlogo*" est le répertoire dans lequel se trouve NetLogo sur votre machine. Sur un Macintosh, il s'agit habituellement de "`/Applications/NetLogo 4.0.3/`", à moins que vous ne l'ayez mis ailleurs.

**Charger un modèle** : Pour charger un modèle, vous devez spécifier le chemin complet du modèle. Dans cet exemple, nous voulons charger le modèle "Forest Fire" et le chemin sera donné en utilisant l'emplacement correspondant à une installation typique sur Macintosh.

```
NLLoadModel["/Applications/NetLogo 4.0/models/Sample Models/Earth Science/Fire.nlogo"];
```

**Donner une commande NetLogo** : Les commandes peuvent être données à NetLogo en passant une chaîne de commandes à `NLCommand[]`. La fonction `NLCommand[]` découpe automatiquement les types de données Mathematica en chaînes compréhensibles par NetLogo. Les commandes suivantes spécifient la densité en utilisant une seule chaîne, ou spécifient la densité en utilisant une variable Mathematica définie, appelée `myDensity`.

```
NLCommand["set density 50"];
myDensity = 60;
NLCommand["set density", myDensity];
```

**Récupérer des informations de NetLogo** : les données calculées par NetLogo peuvent être récupérées par Mathematica au moyen de `NLReport[]`. Ces données peuvent être des nombres, de chaînes de caractères, des valeurs booléennes et des listes.

```
NLReport["count turtles"];
NLReport["[(list pxcor pycor)] of n-of 10 patches"]
```

Pour en savoir plus, consulter le bloc-note tutoriel NetLogo-Mathematica qui se trouve dans le dossier "`Mathematica Link`" de votre dossier NetLogo. Ce bloc-note vous fait parcourir toutes les facettes de l'utilisation de Link, avec de nombreux exemples tout au long du chemin. Si vous ne possédez pas Mathematica (nécessaire pour lire ce bloc-note) mais que vos envisagez d'utiliser cet outil, vous pouvez télécharger la version PDF de ce tutoriel.

## Installation

L'outil NetLogo-Mathematica Link demande au moins NetLogo 4.0 et Mathematica 6.0 ou plus pour fonctionner. Pour installer l'outil NetLogo-Mathematica Link :

1. Allez dans la barre des menus de Mathematica
2. Cliquez sur "File" et sélectionnez "Install..."
3. Dans la boîte de dialogue "Install" de Mathematica
4. Sélectionnez "Package for Type of item to install"
5. Cliquez "Source", et sélectionnez "From file..."
6. Dans le navigateur de fichiers, allez à l'endroit où votre NetLogo est installé,
7. Cliquez le sous-répertoire "Mathematica Link" pour l'ouvrir,
8. Sélectionnez "NetLogo.m".
9. Pour "Install Name", entrez "`NetLogo`".

Vous pouvez installer l'outil NetLogo-Mathematica Link soit dans votre répertoire de base utilisateur (`user`), soit dans le répertoire système général. Si l'outil Link est installé dans le répertoire de base de l'utilisateur, les autres utilisateurs du système devront aussi passer par le processus d'installation de la liaison NetLogo-Mathematica pour pouvoir l'utiliser. Cette option est préférable si vous n'avez pas l'autorisation de modifier des fichiers hors de votre répertoire `home`. Sinon, vous pouvez installer la liaison NetLogo-Mathematica dans le répertoire Mathematica de base du système.

## Problèmes connus

- ✔ Une session NetLogo ne peut être quittée sans quitter aussi J/Link (la liaison Java-Mathematica) entièrement. Cette action peut interrompre les liaisons d'autres packages qui peuvent aussi utiliser J/Link. Ce problème sera résolu dans une future version.
- ✔ Si un modèle chargé avec NetLogo-Mathematica Link utilise une extension NetLogo, l'extension doit se trouver dans le même répertoire que l'extension elle-même. Si l'extension est située dans le répertoire des extensions général de l'application NetLogo, elle ne sera pas trouvée. Ce problème sera résolu dans une future version.
- ✔ Les appels à NetLogo, tels que `NLCommand[]` et `NLReport[]` ne peuvent être annulés ou interrompus.

## Crédits

Le premier développeur de la liaison NetLogo-Mathematica a été Eytan Bakshy.

Pour référencer ce paquet de logiciel dans les publications académiques, utilisez s'il-vous-plaît : Bakshy, E., Wilensky, U. (2007). NetLogo-Mathematica Link. http://ccl.northwestern.edu/netlogo/mathematica.html. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

# Chapitre 23

# Guide des Extensions

NetLogo permet aux utilisateurs d'écrire de nouvelles commandes et de nouveaux reporters en langage Java et de les utiliser dans leurs modèles. Ce chapitre du Guide de l'utilisateur présente cette fonctionnalité.

La première partie explique comment utiliser une extension dans vos modèles une fois que vous l'avez écrite, ou une fois que quelqu'un d'autre l'a écrite pour vous.

La seconde partie est destinée aux programmeurs Java intéressés à écrire leurs propres extensions en utilisant le NetLogo Extension API.

- ✔ Utiliser des extensions
- ✔ Programmer des extensions

Le document NetLogo Extension API Specification (non traduit) contient d'autres détails.

## Utiliser des extensions

Pour utiliser une extension dans un modèle, ajoutez le mot-clé `extensions` au tout début de votre code (dans le panneau "Procedures"), avant toute déclaration de races et/ou de variables.

Après le mot `extensions` vient une liste (entre crochets) des noms des extensions utilisées. Par exemple :

```
extensions [sound speech]
```

L'utilisation de la primitive `extensions` demande à NetLogo de trouver et d'ouvrir les extensions spécifiées et de mettre les commandes et reporters qui s'y trouvent à disposition du modèle courant. Vous pouvez alors utiliser ces commandes et reporters comme s'ils étaient des primitives NetLogo normales.

### Où se trouvent les extension

NetLogo recherche les extensions à plusieurs endroits :

1. Dans le dossier du modèle courant.
2. Dans le dossier `extensions` qui est au même endroit que l'application NetLogo.

Chaque extension NetLogo consiste en un dossier portant le même nom que l'extension, nom entièrement en minuscules. Ce dossier doit contenir un fichier JAR portant le même nom que le dossier. Par exemple, l'extension `sound` est stockée dans un dossier appelé `sound`, lequel contient un fichier appelé `sound.jar`. Pour plus d'informations concernant le contenu du dossier d'une extension, voyez la section Programmer des extensions de ce manuel.

Pour installer une extension NetLogo qui puisse être utilisée par n'importe quel modèle, placez le dossier de l'extension dans le dossier `extensions` du dossier `NetLogo`. Ou alors, vous pouvez laisser le dossier de l'extension dans le même dossier que le modèle qui l'utilise.

Certaines extensions dépendent de fichiers additionnels. Ces fichiers doivent se trouver dans le dossier de l'extension, avec le fichier JAR. Ce dossier peut aussi contenir d'autres fichiers, tels qu'une documentation et des exemples de modèles.

### Extensions et applets

Les modèles sauvegardés sous forme d'applets (en utilisant la commande "Save as Applet" du menu "File" de NetLogo) peuvent aussi utiliser les extensions. L'extension doit être placée dans le même répertoire que le fichier du modèle. Toutefois, les applets ne peuvent (pour le moment) pas utiliser des extensions qui demandent des fichiers JAR externes. (Cette possibilité est planifiée pour une future version.)

## Programmer des extensions

Nous présupposons que vous avez des notions de programmation en Java.

### Résumé

Une extension NetLogo consiste en un dossier contenant :

**Obligatoirement** :

- ✔ Un fichier JAR ayant le même nom que l'extension, et contenant :
  - ✗ une ou plusieurs classes qui implémentent `org.nlogo.api.Primitive`,
  - ✗ une classe principale (*main class*) qui implémente `org.nlogo.api.ClassManager` et
  - ✗ un fichier `manifest.txt` NetLogo contenant les quatre information suivantes :
    - `Manifest-Version`, toujours 1.0
    - `Extension-Name`, le nom de l'extension.
    - `Class-Manager`, le nom complet d'une classe implémentant `org.nlogo.api.ClassManager`.
    - `NetLogo-Extension-API-Version`, la version de l'API Extension de NetLogo pour lequel le fichier JAR a été écrit. Si un utilisateur ouvre cette extension avec un NetLogo ayant une version d'API Extension différente, un message d'avertissement est émis. Pour savoir quelle version de l'API Extension votre NetLogo supporte, sélectionnez la commande "About NetLogo" du menu "Help" (sur Mac, du menu "NetLogo"), puis cliquez sur l'onglet "System" de la fenêtre qui s'ouvre. Vous pouvez également lancer `NetLogo.jar` avec l'argument `--extension-api-version`.

**En option** :

- ✔ Un ou plusieurs modèles NetLogo montrant comment l'extension est utilisée.
- ✔ Un ou plusieurs fichiers JAR requis par l'extension.
- ✔ Un dossier `lib` contenant toutes les bibliothèques natives requises.
- ✔ Un dossier source `src` contenant le code source pour l'extension.
- ✔ De la documentation.

Pour construire votre extension, vous devez inclure `NetLogo.jar` dans le chemin des classes.

## Exemples

Plusieurs exemples d'extensions avec leurs codes sources complets sont livrées avec NetLogo. D'autres sont disponibles à l'adresse http://ccl.northwestern.edu/netlogo/extensions pour téléchargement.

## Tutoriel

Écrivons une extension, appelée `sample` et ne comprenant qu'un seul reporter appelé `first-n-integers`.

Le reporter `first-n-integers` possède un seul argument `n` qui demande une valeur numérique, et retourne une liste de nombres entiers allant de 0 à n - 1. (Bien entendu, vous pourriez aussi le faire facilement en NetLogo; i l s'agit juste d'un exemple.)

### 1. Créer le dossier de l'extension

Puisqu'une extension est un dossier contenant plusieurs fichiers, nous devons d'abord créer ce dossier. Dans cet exercice, il est nommé `example`. Nous allons mettre tout notre travail dans ce dossier. Il nous faut aussi créer un sous-dossier `src` qui contiendra le code Java et un sous-dossier `classes` pour les classes compilées.

### 2. Écrire les primitives

Les primitives sont implémentées sous la forme d'une ou de plusieurs classes Java. Les fichiers `.java` pour ces classes doivent être placés dans le sous-dossier `src` que nous venons de créer.

Une commande accomplit une action, un reporter retourne une valeur. Pour créer une nouvelle commande ou un nouveau reporter, il faut créer une classe qui implémente l'interface de cette nouvelle primitive : `org.nlogo.api.Command` ou `org.nlogo.api.Reporter`, qui étend `org.nlogo.api.Primitive`. Dans la plupart des cas, vous pouvez étendre la classe abstraite `org.nlogo.api.DefaultReporter` ou `org.nlogo.api.DefaultCommand`.

`DefaultReporter` demande que nous implémentions :

```
Object report (Argument args[], Context context)
  throws ExtensionException;
```

Puisque notre reporter possède un argument, nous devons aussi implémenter :

```
Syntax getSyntax();
```

Voici l'implémentation de notre reporter, dans un fichier appelé `src/IntegerList.java` :

```
import org.nlogo.api.*;

public class IntegerList extends DefaultReporter
{
  // reçoit un nombre en entrée, retourne une liste
  public Syntax getSyntax() {
    return Syntax.reporterSyntax(
      new int[] {Syntax.TYPE_NUMBER}, Syntax.TYPE_LIST
    );
  }

  public Object report(Argument args[], Context context)
  throws ExtensionException
  {
    // crée une liste NetLogo pour le résultat
    LogoList list = new LogoList();
```

```
    int n ;
    // utilise la méthode typesafe helper de
    // org.nlogo.api.Argument pour accéder aux arguments
    try
    {
      n = args[0].getIntValue();
    }
    catch( LogoException e )
    {
      throw new ExtensionException( e.getMessage() ) ;
    }

    if (n < 0) {
      // signale une erreur d'exécution NetLogo au modélisateur
      throw new ExtensionException
      ("l'argument doit être positif");
    }

    // remplit la liste
    // notez que nous utilisons des objets Double car
    // les nombres NetLogo sont toujours des doubles
    for (int i = 0; i < n; i++) {
      list.add(new Double(i));
    }
    return list; }
}
```

**Remarques** :

- ✔ Notez que les objets nombre que nous mettons dans la liste retournée par le reporter sont des doubles et non des entiers. Tous les nombres utilisés comme valeurs en NetLogo doivent être de type double, même s'ils ne semblent pas avoir de partie fractionnaire.
- ✔ Pour accéder aux arguments, utilisez les méthodes « typesafe helper » de `org.nlogo.api.Argument`, par exemple `getDoubleValue()`.
- ✔ Générez une exception `org.nlogo.api.ExtensionException` pour signaler une erreur d'exécution NetLogo au développeur du modèle qui utilisera votre extension.

Une commande est presque semblable à un reporter, sauf que le reporter implémente `Object report(...)` alors qu'une commande implémente `void perform(...)`.

### 3. Écrire un gestionnaire de classes "ClassManager"

Chaque extension doit inclure, en plus d'un certain nombres de classes de commandes et de reporters, une classe qui implémente l'interface `org.nlogo.api.ClassManager`. Ce "ClassManager" indique à NetLogo quelles sont les primitives qui font parties de cette extension. Dans les cas simples, étendez la classe abstraite `org.nlogo.api.DefaultClassManager` qui fournit des implémentations vides pour les méthodes de `ClassManager` dont vous n'avez pas besoin.

Voici le gestionnaire de classes pour notre exemple d'extension `src/SampleExtension.java` :

```
import org.nlogo.api.*;

public class SampleExtension extends DefaultClassManager {
    public void load(PrimitiveManager primitiveManager) {
        primitiveManager.addPrimitive
          ("first-n-integers", new IntegerList());
    }
}
```

`addPrimitive()` dit à NetLogo que notre reporter existe et lui communique son nom.

### 4. Écrire un manifeste

L'extension doit aussi comprendre un manifeste. Ce manifeste est un fichier texte `manifest.txt` qui transmet à NetLogo le nom de l'extension et l'emplacement du gestionnaire de classes `ClassManager`.

Ce manifeste, `manifest.txt` doit contenir les quatre informations suivantes :

- ✔ `Manifest-Version`, toujours 1.0
- ✔ `Extension-Name`, le nom de l'extension.
- ✔ `Class-Manager`, le nom complet d'une classe implémentant `org.nlogo.api.ClassManager`.
- ✔ `NetLogo-Extension-API-Version`, la version de l'API Extension de NetLogo pour lequel le fichier JAR a été écrit. Si un utilisateur ouvre cette extension avec un NetLogo ayant une version d'API Extension différente, un message d'avertissement est émis. Pour savoir quelle version de l'API Extension votre NetLogo supporte, sélectionnez la commande "About NetLogo" du menu "Help" (sur Mac, du menu "NetLogo"), puis cliquez sur l'onglet "System" de la fenêtre qui s'ouvre. Vous pouvez également lancer `NetLogo.jar` avec l'argument `–extension-api-version`.

Voici un manifeste pour notre exemple d'extension, `manifest.txt` :

```
Manifest-Version: 1.0
Extension-Name: example
Class-Manager: SampleExtension
NetLogo-Extension-API-Version: 4.0
```

La ligne NetLogo-Extension-API-Version doit contenir la version de l'Extension API de NetLogo que vous utilisez.

Assurez-vous que même la dernière ligne du fichier se termine par un caractère « newline ».

### 5. Créer un JAR

Pour créer un fichier JAR de l'extension, compilez d'abord vos classes comme d'habitude, soit en utilisant la ligne de commande, soit avec votre environnement de développement (IDE) Java.

**Important** : Vous devez ajouter `NetLogo.jar` (de la distribution NetLogo) dans vos chemins des classes (*classpath*) pour la compilation.

Voici un exemple montrant à quoi pourrait ressembler la compilation de votre extension au moyen de la ligne de commandes :

```
$ mkdir -p classes     # create the classes subfolder if it does not exist
$ javac -classpath NetLogo.jar -d classes src/IntegerList.java src/SampleExtension.java
```

Vous devrez certainement modifier l'argument `classpath` de manière à ce qu'il pointe vers le fichier `NetLogo.jar` de votre installation. Cette ligne de commandes va compiler vos fichiers `.java` et mettre les fichiers `.class` dans le sous-dossier `classes`.

Il faut ensuite créer un JAR contenant les fichiers class résultants ainsi que le manifeste. Par exemple :

```
$ jar cvfm example.jar manifest.txt -C classes .
```

Pour en savoir plus au sujet des fichiers `manifest`, des fichiers JAR et des outils Java, consultez le site java.sun.com.

### 6. Utiliser l'extension créée dans un modèle

Pour pouvoir utiliser l'exemple d'extension dans un modèle, placer le dossier `example` dans le dossiers des extensions de NetLogo ou dans le même dossier que celui contenant le modèle qui fait appel à cette extension. Écrivez ensuite au tout début du code du modèle (dans le panneau "Procedures") :

```
extensions [example]
```

Dès maintenant, vous pouvez utiliser `example:first-n-integers` comme si c'était un reporter pré-programmé de NetLogo. Par exemple, activez le panneau "Interface" et écrivez dans la Ligne de commandes du Centre de commande :

```
observer> show example:first-n-integers 5
observer: [0 1 2 3 4]
```

## Quelques tuyaux pour le développement des extensions

### Instanciation

Votre gestionnaire de classes est instancié au moment où un modèle utilisant l'extension est chargé.

Les objets commande et reporter sont instanciés chaque fois que le code NetLogo qui utilise vos commandes et reporters est compilé.

### Les chemins des classes (*classpath*)

N'oubliez pas d'inclure le fichier `NetLogo.jar` dans le chemin des classes pour la compilation. C'est l'erreur la plus fréquente commise par les nouveaux programmeurs d'extensions. (Si le compilateur ne peut pas trouver le fichier `NetLogo.jar`, il vous envoie des messages d'erreurs indiquant que des classes du paquet `org.nlogo.api` n'ont pas été trouvées).

### Déverminer les extensions

Il existe des primitives NetLogo pour vous aider à développer et déverminer votre extension. Elles sont pour le moment expérimentales et leur implémentation est susceptible de changer dans le futur. (C'est pourquoi leur nom commence par deux caractères soulignement.)

- ✔ `print __dump-extensions` affiche des informations concernant les extensions chargées.
- ✔ `print __dump-extension-prims` affiche des informations concernant les primitives d'extension chargées.
- ✔ `__reload-extensions` oblige NetLogo à recharger toutes les extensions la prochaine fois que vous compilez votre modèle. Sans cette commande, les modifications apportées au JAR de l'extension ne sont pas prises en compte tant que vous n'ouvrez par un modèle ou tant que vous ne redémarrez pas NetLogo.

### JAR provenant de tiers

Si votre extension dépend de code stocké dans un fichier JAR séparé, copiez ce JAR externe dans le dossier extension. Chaque fois qu'une extension est importée, NetLogo fait en sorte que tous les fichiers JAR de ce dossier soient à disposition de l'extension.

Si vous prévoyez de distribuer votre extension à d'autres utilisateurs de NetLogo, n'oubliez pas de fournir des instructions d'installation adéquates.

### Supporter les anciennes versions de Java

NetLogo fonctionne avec les versions 1.4.1 et ultérieures de Java. Si vous voulez que votre extension soit utilisable par tous les utilisateurs de NetLogo, votre extension doit supporter Java 1.4.1.

La façon la plus simple de remplir cette condition est de faire tout le développement avec le JDK Java 1.4.1.

Il est aussi possible de développer pour Java 1.4 en utilisant le compilateur Java 1.5 ou 1.6, mais vous devez alors faire deux choses :

- Utilisez l'option `-target 1.4` pour `javac` (ou l'équivalent dans votre IDE) pour dire au nouveau compilateur de fabriquer des classes compatibles avec les anciennes versions de Java. Cette option garantit que votre code n'utilise pas les nouvelles fonctionnalités des Java 1.5 ou 1.6 inconnues des versions précédentes.
- Utilisez l'option `-bootclasspath` pour `javac` (ou l'équivalent dans votre IDE) pour compiler conformément aux bibliothèques des classes de java 1.4. (Notez que cette option nécessite dans tous les cas l'installation du JDK 1.4.) Cette option garantit que votre code ne fera pas appel à des fonctionnalités réservées aux API Java 1.5 et 1.6.

## Conclusion

N'oubliez pas de consulter les NetLogo API Specification pour tous les détails concernant ces classes, interfaces et méthodes.

Notez que le programmeur de modèles NetLogo n'a aucun moyen d'obtenir la liste des commandes et des reporters fournis par une extension. C'est pourquoi il est important que vous fournissiez la documentation adéquate.

L'outil extension n'est pour le moment pas complet. Son API ne contient pas tout ce que vous pourriez espérer. Certaines fonctionnalités existent mais ne sont pas encore documentées. Si vous n'y trouvez pas une fonctionnalité que vous aimeriez, faites-nous le savoir. N'hésitez pas à nous contacter à l'adresse feedback@ccl.northwestern.edu pour nous posez des questions, car nous pouvons peut-être trouver une solution de remplacement ou fournir des informations supplémentaires là où notre documentation est lacunaire.

Vos avis, questions et commentaires au sujet de cet API nous permettrons de concentrer nos efforts de manière appropriée pour les versions futures. Nous nous sommes engagés à faire de NetLogo un logiciel souple et extensible, et accueillons bien volontiers votre « *feedback* ».

# Chapitre 24

# Les extensions Array et Table

Ces extensions ajoutent deux nouvelles structures de données à NetLogo : les tableaux (*array*) et les tables de hachage (*table*).

## Quand les utiliser

En général, tout ce que vous pouvez faire avec une tableau ou une table de hachage peut aussi se faire avec une simple liste. Mais vous pouvez envisager de la remplacer par un tableau ou une table de hachage pour des raisons de vitesse d'exécution du programme. Ces trois structures de données (liste, tableau et table de hachage) affichent des performances différentes, ce qui permet, en utilisant la structure de données adéquate, de créer un modèle qui tourne plus rapidement.

Les tableaux sont pratiques quand on a besoin d'une collection de données de taille fixe. Ils permettent d'accéder à, ou de lire très rapidement n'importe quel élément si vous connaissez sa position dans le tableau.

Les tables de hachage sont utiles quand il faut associer des valeurs à d'autres valeurs. Vous pouvez par exemple construire une table de mots avec leurs définitions puis rechercher ensuite la définition de n'importe quel mot. Ici, les mots sont les « clés » de la table. La recherche et la récupération de n'importe quelle clé de la table est rapide et facile, ce qui n'est pas le cas des définitions.

## Comment les utiliser

Les deux extensions sont déjà installées dans NetLogo.

Pour utiliser l'extension `array` dans un modèle, ajoutez la ligne suivante au début du code dans le panneau "Procedures" :

```
extensions [array]
```

Vous pouvez même utiliser les deux extensions dans le même modèle, comme suit :

```
extensions [array table]
```

Si le modèle utilise d'autres extensions, son code possède déjà une ligne `extensions`, il suffit simplement d'ajouter `array` et/ou `table` dans la liste des extensions.

Pour plus d'informations concernant les extensions NetLogo, voir le Guide des extensions.

### Limites concernant les clés des tables

Les clés des tables de hachage ne peuvent être que des chaînes de caractères, des nombres, des booléens ou des listes. (Les listes peuvent être des listes imbriquées aussi longtemps que tous leurs éléments sont des chaînes de caractères, des nombres ou des booléens.)

## Exemple de tableau (avec l'extension Array)

```
let a array:from-list n-values 5 [0]
print a
=> {{array: 0 0 0 0 0}}
print array:length a
=> 5
foreach n-values 5 [?] [ array:set a ? ? * ? ]
print a
=> {{array: 0 1 4 9 16}}
print array:item a 0
=> 0
print array:item a 3
=> 9
array:set a 3 50
print a
=> {{array: 0 1 4 50 16}}
```

## Exemple de table de hachage (avec l'extension Table)

```
let dict table:make
table:put dict "turtle" "cute"
table:put dict "bunny" "cutest"
print dict
=> {{table: "turtle" -> "cute", "bunny" -> "cutest" }}
print table:length dict
=> 2
print table:get dict "turtle"
=> "cute"
print table:get dict "leopard"
=> 0
print table:keys dict
=> ["turtle" "bunny"]
```

## Problèmes connus

Quand vous exportez un monde NetLogo (en utilisant la commande `export-world` ou la commande de menu "Export World"), les tableaux et les tables de hachage sont exportés « par valeur ». Ce qui signifie que si vous avez la même table ou le même tableau stocké à plus d'un endroit, il y aura des tableaux ou des tables distinctes aux endroits où apparaissait la table ou le tableau original quand elle ou il sera exporté puis ré-importé. Ces duplicata contiennent initialement les mêmes valeurs, mais si l'un d'eux est modifié par le suite, l'autre ne le sera pas.

## Les primitives de l'extension Array

`array:from-list` `array:item` `array:set` `array:length` `array:to-list`

### array:from-list

`array:from-list` *`list`*

Retourne un nouveau tableau contenant les mêmes éléments, dans le même ordre, que dans la liste *`list`*.

### array:item

`array:item` *`array index`*

Retourne l'élément numéro *`index`* du tableau *`table`* donné (les numéros d'index vont de 0 à nombre d'éléments du tableau moins un).

### array:set

`array:set` *`array index value`*

Donne *`value`* à l'élément numéro *`index`* du tableau *`table`* donné (les numéros d'index vont de 0 à nombre d'éléments du tableau moins un).

Notez que contrairement à la primitive `replace-item` pour les listes, cette commande ne crée pas de nouveau tableau. C'est le tableau existant qui est modifié.

### array:length

`array:length` *`array`*

Retourne la longueur du tableau *`table`* donné, c'est-à-dire le nombre d'éléments contenus dans le tableau.

### array:to-list

`array:to-list` *`array`*

Retourne une nouvelle liste contenant les mêmes éléments, dans le même ordre, que le tableau *`table`* donné.

## Les primitives de l'extension Tables

`table:clear` `table:from-list` `table:get` `table:has-key?` `table:keys` `table:length` `table:make` `table:put` `table:remove` `table:to-list`

### table:clear

`table:clear` *`table`*

Supprime toutes les paires clé-valeur de la *`table`*.

### table:from-list

```
table:from-list list
```

Retourne une nouvelle table formée des éléments de *list*. L'entrée doit être une liste de listes contenant deux éléments. Le premier élément est la clé et le second élément est la valeur qui lui est associée.

### table:get

```
table:get table key
```

Retourne la valeur associée à la clé *key* qui est stockée dans la *table*. Génère une erreur si *table* n'a pas de paire correspondant à la clé fournie.

### table:has-key?

```
table:has-key? table key
```

Retourne `true` (vrai) si la clé *key* possède une entrée (une paire correspondante dans la *table*.

### table:keys

```
table:keys table
```

Retourne une liste de toutes les clés de la *table*.

### table:length

```
table:length table
```

Retourne le nombre de paires d'éléments dans la *table*.

### table:make

```
table:make
```

Retourne une nouvelle table vide.

### table:put

```
table:put table key value
```

Place la clé *key* et sa valeur associée *value* dans la *table* et les met en correspondance. Si *table* contient déjà une entrée pour la clé *key* donnée, sa valeur est remplacée par *value*.

### table:remove

```
table:remove table key
```

Supprime la clé *key* et sa valeur dans la *table*.

### table:to-list

```
table:to-list table
```

Retourne une liste formée des éléments de la table *table*. Les éléments de la liste sont des listes à deux éléments (ou paires). Le premier élément de la paire est la clé et le second est la valeur qui lui est associée.

# Chapitre 25

# L'extension Sound

L'extension Sound de NetLogo offre des primitives permettant d'ajouter du son aux modèles NetLogo. Elle utilise deux méthodes pour produire des sons : les sons Midi et la lecture de fichiers son pré-enregistrés.

## Utiliser l'extension Sound

L'extension Sound est pré-installée dans NetLogo. Pour l'utiliser dans un modèle, il suffit d'ajouter la ligne suivante au début du code du modèle, dans le panneau "Procedures" :

```
extensions [sound]
```

Si le modèle fait déjà appel à d'autres extensions, la ligne `extensions` est déjà présente dans le code. Il suffit alors d'ajouter `sound` dans la liste des extensions.

Pour en savoir plus sur l'utilisation des extensions NetLogo, consultez le Guide des Extensions.

Pour étudier des exemples de modèles utilisant l'extension Sound, allez dans la partie "Sound" de la section "Code Examples" de la Bibliothèque des modèles.

Notez que l'extension Sound ne fonctionne que dans l'application NetLogo. Elle ne fonctionne pas dans les applets des modèles.

## Le support MIDI

La partie MIDI de l'extension simule un clavier électronique à 128 touches avec les 47 percussions et les 128 instruments mélodiques définis par la spécification General MIDI Level 1.

Elle offre 15 canaux d'instruments polyphoniques et un seul canal pour les percussions. L'utilisation en même temps de plus de 15 instruments mélodiques différents dans un modèle provoquera des pertes de sons ou même des coupures du son.

La hauteur du son d'un instrument mélodique est spécifiée par le numéro d'une touche. Les touches du clavier sont numérotées de 0 à 127, successivement de gauche à droite, celle de gauche portant le numéro 0. La touche correspondant à la note C (autrement Do) moyen porte le numéro 60.

La volume (puissance) d'un instrument est spécifiée par une vitesse (vélocité) qui représente la force avec laquelle la touche du clavier a été enfoncée. Les vélocités vont de 0 à 127, où 64 est la vélocité standard. Une vélocité plus grande produit un son plus puissant.

## Les primitives

sound:drums sound:instruments sound:play-drum sound:play-note sound:play-note-later
sound:play-sound sound:play-sound-and-wait sound:play-sound-later sound:start-note
sound:stop-note sound:stop-instrument sound:stop-music

### sound:drums

```
sound:drums
```

Retourne une liste contenant les noms des 47 percussions à utiliser avec la primitive `sound:play-drum`.

### sound:instruments

```
sound:instruments
```

Retourne une liste contenant les noms des 128 instruments à utiliser avec les primitives `sound:play-note`, `sound:play-note-later`, `sound:start-note` et `sound:stop-note`

### sound:play-drum

```
sound:play-drum drum velocity
```

Joue une percussion.

```
sound:play-drum "ACOUSTIC SNARE" 64
```

### sound:play-note

```
sound:play-note instrument keynumber velocity duration
```

Joue la note *keynumber* avec la puissance *velocity* pendant *duration* secondes avec l'instrument *instrument*. L'agent n'attend pas la fin de la note pour passer à la commande suivante.

```
;; jouer la note C moyen (Do) à la trompette pendant 2 secondes
sound:play-note "TRUMPET" 60 64 2
```

### sound:play-note-later

```
sound:play-note-later delay instrument keynumber velocity duration
```

Attend le temps *delay* avant de jouer la note *keynumber* à la puissance *velocity* pendant *duration* secondes avec l'instrument *instrument*. L'agent n'attend pas la fin de la note pour passer à la commande suivante.

```
;; dans une seconde,
;; jouer la note C moyen (Do) à la trompette pendant 2 secondes
sound:play-note-later 1 "TRUMPET" 60 64 2
```

### sound:play-sound

`sound:play-sound` *`filename`*

Lit et joue le fichier son *`filename`*. NetLogo n'attend pas fin de la lecture du fichier son pour passer à la commande suivante. Il peut lire des fichiers son aux formats WAV, AIFF et AU.

```
;; jouer le fichier d'échantillonnage beep.wav
sound:play-sound "beep.wav"
```

### sound:play-sound-and-wait

`sound:play-sound-and-wait` *`filename`*

Lit et joue le fichier son *`filename`* et attend d'arriver à la fin du fichier avant de passer à la commande suivante. Cette primitive peut lire des fichiers son aux formats WAV, AIFF et AU.

```
;; jouer le fichier d'échantillonnage beep.wav et attendre
;; d'en avoir terminé pour jouer le fichier boop.wav
sound:play-sound-and-wait "beep.wav"
sound:play-sound-and-wait "boop.wav"
```

### sound:play-sound-later

`sound:play-sound-later` *`filename delay`*

Attend *`delay`* secondes puis lit et joue le fichier son *`filename`*. NetLogo n'attend pas de jouer le fichier ou même de l'avoir joué entièrement avant de passer à la commande suivante. Cette primitive peut lire des fichiers son aux formats WAV, AIFF et AU.

```
;; jouer le fichier son beep.wav dans une seconde
sound:play-sound-later "beep.wav" 1
```

### sound:start-note

`sound:start-note` *`instrument keynumber velocity`*

Demande à l'instrument *`instrument`* de commencer de jouer la note *`keynumber`* avec le volume *`velocity`*.

La note est tenue (jouée) jusqu'à l'appel de l'une des procédures `sound:stop-note`, `sound:stop-instrument` ou `sound:stop-music`.

```
;; jouer la note C moyen (Do) au violon
sound:start-note "VIOLIN" 60 64
```

```
;; jouer une gamme en C-majeur (Do majeur) avec un xylophone
foreach [60 62 64 65 67 69 71 72]
[
  sound:start-note "XYLOPHONE" ? 65
  wait 0.2
  sound:stop-note "XYLOPHONE" ?
]
```

### sound:stop-note

```
sound:stop-note instrument keynumber
```

Arrête de jouer (tenir) la note *keynumber* avec l'instrument *instrument*.

```
;; arrêter de jouer la note C moyen (Do) avec le violon
sound:stop-note "VIOLIN" 60
```

### sound:stop-instrument

```
sound:stop-instrument instrument
```

Arrête de jouer toutes les notes sur l'instrument *instrument*.

```
;; arrêter de jouer du violoncelle
sound:stop-instrument "CELLO"
```

### sound:stop-music

```
sound:stop-music
```

Arrête de jouer toutes les notes sur tous les instruments.

## Les noms des instruments disponibles

### Les percussions (drums)

```
35. Acoustic Bass Drum          59. Ride Cymbal 2
36. Bass Drum 1                 60. Hi Bongo
37. Side Stick                  61. Low Bongo
38. Acoustic Snare              62. Mute Hi Conga
39. Hand Clap                   63. Open Hi Conga
40. Electric Snare              64. Low Conga
41. Low Floor Tom               65. Hi Timbale
42. Closed Hi Hat               66. Low Timbale
43. Hi Floor Tom                67. Hi Agogo
44. Pedal Hi Hat                68. Low Agogo
45. Low Tom                     69. Cabasa
47. Open Hi Hat                 70. Maracas
47. Low Mid Tom                 71. Short Whistle
48. Hi Mid Tom                  72. Long Whistle
49. Crash Cymbal 1              73. Short Guiro
50. Hi Tom                      74. Long Guiro
51. Ride Cymbal 1               75. Claves
52. Chinese Cymbal              76. Hi Wood Block
53. Ride Bell                   77. Low Wood Block
54. Tambourine                  78. Mute Cuica
55. Splash Cymbal               79. Open Cuica
56. Cowbell                     80. Mute Triangle
57. Crash Cymbal 2              81. Open Triangle
58. Vibraslap
```

## Les instruments

**A. Piano**
1. Acoustic Grand Piano
2. Bright Acoustic Piano
3. Electric Grand Piano
4. Honky-tonk Piano
5. Electric Piano 1
6. Electric Piano 2
7. Harpsichord
8. Clavi

**B. Percussions chromatiques**
9. Celesta
10. Glockenspiel
11. Music Box
12. Vibraphone
13. Marimba
14. Xylophone
15. Tubular Bells
16. Dulcimer

**C. Orgues**
17. Drawbar Organ
18. Percussive Organ
19. Rock Organ
20. Church Organ
21. Reed Organ
22. Accordion
23. Harmonica
24. Tango Accordion

**D. Guitarres**
25. Nylon String Guitar
26. Steel Acoustic Guitar
27. Jazz Electric Guitar
28. Clean Electric Guitar
29. Muted Electric Guitar
30. Overdriven Guitar
31. Distortion Guitar
32. Guitar harmonics

**E. Basses**
33. Acoustic Bass
34. Fingered Electric Bass
35. Picked Electric Bass
36. Fretless Bass
37. Slap Bass 1
38. Slap Bass 2
39. Synth Bass 1
40. Synth Bass 2

**F. Cordes**
41. Violin
42. Viola
43. Cello
44. Contrabass
45. Tremolo Strings
47. Pizzicato Strings
47. Orchestral Harp
48. Timpani

**I. Instruments à hanche**
65. Soprano Sax
66. Alto Sax
67. Tenor Sax
68. Baritone Sax
69. Oboe
70. English Horn
71. Bassoon
72. Clarinet

**J. Flûtes**
73. Piccolo
74. Flute
75. Recorder
76. Pan Flute
77. Blown Bottle
78. Shakuhachi
79. Whistle
80. Ocarina

**K. Synthétiseurs**
81. Square Wave
82. Sawtooth Wave
83. Calliope
84. Chiff
85. Charang
86. Voice
87. Fifths
88. Bass and Lead

**L. Synth Pad**
89. New Age
90. Warm
91. Polysynth
92. Choir
93. Bowed
94. Metal
95. Halo
96. Sweep

**M. Synth Effects**
97. Rain
98. Soundtrack
99. Crystal
100. Atmosphere
101. Brightness
102. Goblins
103. Echoes
104. Sci-fi

**N. Ethniques**
105. Sitar
106. Banjo
107. Shamisen
108. Koto
109. Kalimba
110. Bag pipe
111. Fiddle
112. Shanai

```
 G. Ensembles                    O. Percussions
49. String Ensemble 1           113. Tinkle Bell
50. String Ensemble 2           114. Agogo
51. Synth Strings 1             115. Steel Drums
52. Synth Strings 2             116. Woodblock
53. Choir Aahs                  117. Taiko Drum
54. Voice Oohs                  118. Melodic Tom
55. Synth Voice                 119. Synth Drum
56. Orchestra Hit               120. Reverse Cymbal

 H. Cuivre                      P. Effets sonores
57. Trumpet                     121. Guitar Fret Noise
58. Trombone                    122. Breath Noise
59. Tuba                        123. Seashore
60. Muted Trumpet               124. Bird Tweet
61. French Horn                 125. Telephone Ring
62. Brass Section               126. Helicopter
63. Synth Brass 1               127. Applause
64. Synth Brass 2               128. Gunshot
```

# Chapitre 26

# NetLogoLab et la carte GoGo

## Qu'est-ce que NetLogoLab?

NetLogoLab est l'infrastructure technologique qui relie NetLogo au monde physique (réel). Elle peut être utilisée pour faire de la robotique, des travaux manuels interactifs, pour mener des recherches scientifiques et même pour valider des modèles. Pour en savoir plus, consultez le site NetLogoLab où vous trouverez des documents académiques et des démonstrations.

NetLogoLab se compose des logiciels et matériels suivants :

- ✔ Une extension NetLogo servant à contrôler une carte pilotant un robot ou recueillant des données mesurées par des senseurs.
- ✔ Une carte robotique ou de réception de données (aussi connue sous le nom de carte d'interface série ou de carte digitale-analogique).
- ✔ Des ensembles de capteurs (senseurs) et de d'actionneurs (*actuators*).
- ✔ Des modèles NetLogo.

La carte d'entrées/sorties digitale-analogique préférée de NetLogo est la suivante : GoGo Board — une interface open-source, facile à construire et bon marché conçue à l'origine au Media Lab du MIT par Arnan Sipitakiat. Mais d'autres matériels de robotique peuvent être utilisés avec NetLogo, y compris ceux qui sont disponibles dans le commerce, tels que les capteurs et effecteurs Vernier et Pasco, de même que les kits Phidgets, Lego robotics et VEX — mais des extensions spécifiques doivent encore être développées pour chacune de ces plateformes. Ce qui fait que jusqu'ici, seule l'extension GoGo Board est disponible et utilisable avec la distribution NetLogo standard.

### L'extension NetLogo GoGo Board

L'extension GoGo Board est un composant logiciel qui permet à l'utilisateur de relier le monde virtuel NetLogo au monde réel (physique) au moyen de capteurs, de moteurs, de sources lumineuses, de LEDs, de relais et de bien d'autres dispositifs encore. L'extension GoGo Board pour NetLogo fournit des primitives simples pour communiquer avec une carte d'interface GoGo.

### GoGo Board : une carte d'interface pour la robotique et l'acquisition de données

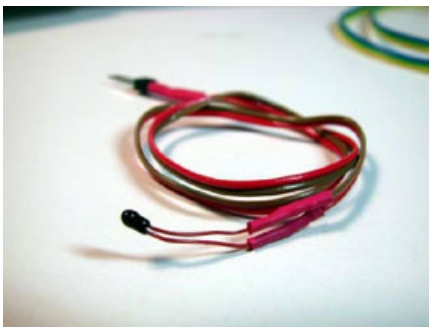
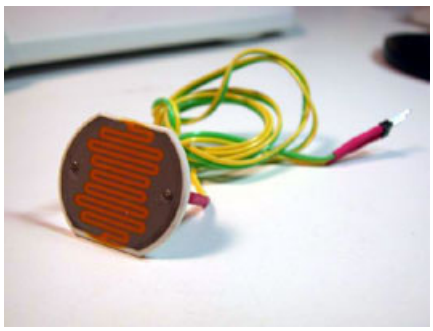
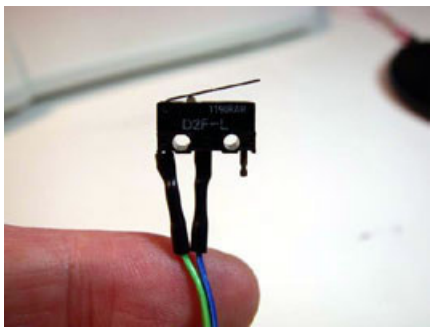
La carte GoGo est une carte d'interface série, open-source, facile à construire, bon marché et multi-usages, spécialement conçue pour être utilisée dans les écoles et les projets à buts éducatifs. Elle a été créée par Ar-

nan Sipitakiat, en collaboration avec Paulo Blikstein au MIT Media Lab en 2001 et a été activement développée en continu jusqu'à aujourd'hui. Elle est actuellement utilisée dans plus de dix pays, tels que : États-Unis, Chine, Thaïlande, Brésil, Portugal, Mexique, Malaisie et Égypte par exemple.

Il est possible d'y connecter jusqu'à huit capteurs (par exemple de température, de lumière, de pression) et quatre effecteurs (par exemple moteurs, sources de lumière, LED, relais) simultanément. La carte possède aussi un connecteur pour y brancher des instruments supplémentaires (tels qu'un petit écran ou un module de communication sans fil). Couplée avec un boîtier série Bluetooth (tel que les modèles Iogear ou WCSC) plutôt que reliée à l'ordinateur par un câble série, elle peut même être utilisée en tant qu'interface sans fil.

## Les capteurs et les effecteurs

Les modèles NetLogo peuvent interagir avec le monde réel (physique, matériel) de deux manières. Premièrement, ils peuvent recueillir des données provenant de l'environnement. Ces informations peuvent être utilisées par le modèle pour modifier ou adapter son comportement. Ces données sont collectées au moyen de senseurs électroniques capables de mesurer toute une série de phénomènes : température, lumière, contact (voir les illustrations ci-dessous), pH, concentrations de produits chimiques divers, pression, etc.

Le second type d'interactions entre NetLogo et le monde réel est le contrôle de dispositifs en sortie, ou effecteurs (*actuators*) tels que moteurs, sources de lumière, LED ou encore de dispositifs plus complexes qui réunissent ces effecteurs en une seule machine tels que des jouets, des véhicules télécommandés, des appareils électriques et des équipements de laboratoires automatisés.



Les enseignants et éducateurs qui veulent se lancer dans des projets de robotique ou de collecte de données environnementales doivent prendre en compte quelques paramètres importantes concernant le type exact des capteurs et effecteurs à utiliser — par exemple leur solidité, leur fiabilité, leur « versatilité » (*openness*) et leur coût. Le prix et la complexité de ces dispositifs varient fortement. Par exemple, pour la plupart des projets éducatifs, les capteurs génériques bon marchés d'usage général peuvent être utilisés avec des résultats très fiables. Un capteur de température générique ayant une précision de ± 0.5 °C ne coûte pas plus de quelques francs chez la plupart des revendeurs de matériel électronique. L'utilisation de capteurs générique bon marchés ne demande que des connaissances de base en électronique. Par exemple, des notions de base en soudure sont nécessaires pour fixer un bout de fil électrique à un capteur. Bien que ce soit réalisable

dans les écoles et ait été testé dans de nombreux établissements éducatifs, certains enseignants préfèrent acheter des capteurs et des effecteurs de marque, qui sont déjà tout assemblés et prêts à l'emploi. Certaines sociétés vendent des systèmes de marque spécialisés pour l'éducation qui sont nettement plus solides et plus fiables que les systèmes génériques, mais qui coûtent aussi beaucoup plus chers. En comparaison, un capteur de température de marque peut coûter jusqu'à 50 US$. Les effecteurs suivent la même règle : par exemple, un moteur générique peut coûter de 3 à 10 US$ alors que la version de marque est vendue pour 30 ou 40 US$.

Les senseurs et les effecteurs peuvent être achetés en ligne chez les commerçants suivants (aux USA) : Digi-key, Mouser, Phidgets, Spark Fun et Solarbotics. Des informations supplémentaires sur la manière de faire des capteurs peuvent être trouvées sur la page "How to Make Sensors" du site web de la carte GoGo.

### Les modèles NetLogo

Pour pouvoir utiliser l'extension GoGo Board et le logiciel NetLogoLab, les utilisateurs doivent créer des modèles NetLogo qui utilisent les primitives spéciales mises à disposition par l'extension. Nous fournirons plus loin dans ce document quelques exemples de modèles qui montrent comment y parvenir.

## Comment obtenir une carte GoGo?

La carte GoGo n'est pas un produit commercial et, de ce fait, ne peut être achetée dans le commerce. Pour avoir une carte GoGo, vous devez la construire vous-même ou demander à quelqu'un de le faire pour vous. Plusieurs sociétés peuvent construire de telles cartes, mais elles demandent en général d'en construire une quantité minimale qui peut aller de 5 à 50. La carte GoGo a été spécialement conçue pour être bon marché et facile à construire, même si vous n'êtes pas un as en électronique. La principale ressource pour ce qui concerne la carte GoGo est le site web www.gogoboard.org où vous trouverez des instructions détaillées vous indiquant pas à pas comment acheter les composants, dessiner et construire la carte du circuit électronique et assembler le tout. La liste de diffusion de la carte GoGo est gogoboard@yahoogroups.com, liste que vous pouvez rejoindre via le groupe Yahoo gogoboard. Le wiki de la carte GoGo se trouve à l'adresse www.edudesign.org/gogowiki.

## Installer et tester l'extension GoGo

Il faut bien que la carte GoGo communique avec l'ordinateur, et pour ce faire, elle utilise le port série. Le choix de ce port plutôt que d'un port USB a été motivé par le principe d'une carte à bas coûts : les composants nécessaires à la construction d'une carte compatible USB sont nettement plus onéreux. Si votre ordinateur n'a pas de port série, il vous faut acheter un adaptateur USB-série, que l'on trouve facilement dans les magasins de matériel informatique pour un prix allant de 15 à 30 US$ (si vous avez un Mac ou une machine Linux, assurez-vous que l'adaptateur est compatible avec votre machine). Le modèle Keyspan (USA-19HS) est un adaptateur qui fonctionne normalement relativement bien avec toutes les plates-formes, mais c'est aussi l'un des plus cher.

Si vous envisagez d'utiliser un adaptateur série-USB, vous devez vous assurer que les pilotes adéquats sont installés avant de commencer. En général, l'adaptateur est livré avec un CD qui contient les pilotes appropriés; si ce n'est pas le cas, il vous faut les télécharger à partir du site du constructeur de l'adaptateur. Commencez par connecter l'adaptateur à l'ordinateur et à la carte GoGo. Enclenchez ensuite la carte GoGo en utilisant le commutateur placé derrière la prise d'alimentation. La carte GoGo émet un bip et la lumière rouge s'allume.

### Avec Windows

L'extension GoGo ne demande aucune autre installation spéciale avec Windows.

### Avec Mac OSX

Il y a un bug, soit dans Mac OSX, soit dans le bibliothèque RXTX que nous utilisons pour la connexion sérielle, qui vous oblige à entrer les commandes suivantes dans l'application Terminal (`/Applications/Utilities/Terminal.app`) avant de pourvoir travailler avec la carte GoGo :

```
sudo mkdir /var/lock
sudo chmod 1777 /var/lock
sudo rm /var/spool/uucp/*
```

Une fois ces commandes passées, vous pouvez démarrer NetLogo. Ouvrez l'exemple `GoGoMonitor` et pressez le bouton "setup". Une boîte de dialogue s'ouvre et vous propose une liste des ports disponibles. Sélectionnez le port auquel la carte GoGo est reliée. Il devrait s'appeler quelque chose comme :

```
/dev/tty.KeySerial1
```

où les mots suivant `/dev/tty.` devraient être quelque chose en rapport avec le type d'adaptateur que vous possédez et inclure le mot `serial`. Si vous n'arrivez pas à savoir quel est le bon port, vous pouvez le trouver dans l'application "Informations Système" que vous ouvrez en sélectionnant "A propos de ce Mac" dans le menu "Pomme" ("Apple") puis en cliquant "Plus d'infos..." dans le fenêtre qui s'ouvre.

Après l'initialisation (setup), pressez le bouton "ping". Devrait alors s'afficher le message `true` dans le Centre de commande. Vous devriez aussi voir des valeurs qui fluctuent légèrement dans les moniteurs des senseurs.

Si cela ne fonctionne pas, contactez-nous à l'adresse bugs@ccl.northwestern.edu.

### Avec Linux

Vous aurez besoin de pouvoir écrire dans les périphériques série, normalement `/dev/ttyS*`. Dans la plupart des distributions Linux, cela peut être spécifié via le "User Manager".

## Utiliser l'extension GoGo

L'extension GoGo est installée automatiquement quand vous téléchargez et installez NetLogo. Pour utiliser l'extension dans un modèle, ajoutez la ligne suivante au tout début du code dans le panneau "Procedures" :

```
extensions [gogo]
```

Si le modèle fait déjà appel à d'autres extensions, la ligne `extensions` est déjà présente dans le code. Il suffit alors d'ajouter `gogo` dans la liste des extensions.

Après chargement de l'extension, voyez quels ports sont disponibles en tapant la commande suivante dans le Centre de commande :

```
print gogo:ports
```

Vous devez ouvrir le port série auquel la carte GoGo est connectée de manière à ce que NetLogo puisse commencer à envoyer des commandes à la carte au moyen de la commande `gogo:open`. Pour être sûr que la carte est connectée correctement, testez si elle répond au reporter `ping`. Notez que pour être en mesure de communiquer avec la carte, vous avez besoin de savoir à quel port elle est connectée. Si vous n'êtes pas sûr du port utilisé, vous pouvez vous servir de la primitive `gogo:ports` (voir ci-dessous), mais vous pouvez aussi le trouvez en utilisant le "Device manager" sur un ordinateur Windows (cliquez l'icône Système dans le Panneau de contrôle) ou les "Informations Système" sur Mac OS X (voir ci-dessus).

Avec Windows :

```
gogo:open "COM1"
print gogo:ping
```

Avec Linux :

```
gogo:open "/dev/ttyS01"
print gogo:ping
```

Avec Mac :

```
gogo:open "/dev/tty.KeySerial1"
print gogo:ping
```

Pour en savoir plus sur les extensions NetLogo, consultez le Guides des extensions.

Les modèles sauvegardés sous formes d'applets (au moyen de la commande "Save as Applet" du menu "File") ne peuvent utiliser l'extension GoGo. Les applets ne peuvent utiliser ni des extensions qui font appel à des bibliothèques additionnelles, ni accéder à des appareils périphériques. Pour voir des exemples de modèles qui utilisent l'extension GoGo, consultez la section "Robotics/NetLogoLab" dans le répertoire "Sample Models" de la Bibliothèques des modèles (*encore inexistant dans la version NetLogo 4.0.4*).

## Exemples de modèles NetLogoLab

### Contrôler une voiture

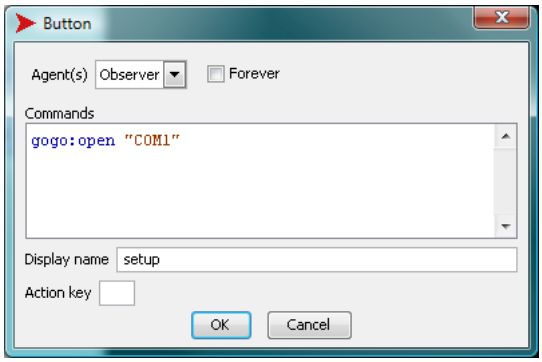
Le premier pas à faire quand on veut créer un modèle NetLogoLab est d'ajouter la commande `extensions` dans le panneau des procédures de NetLogo. Allez dans le panneau "Procedures" et ajoutez la ligne suivante :

```
extensions [gogo]
```

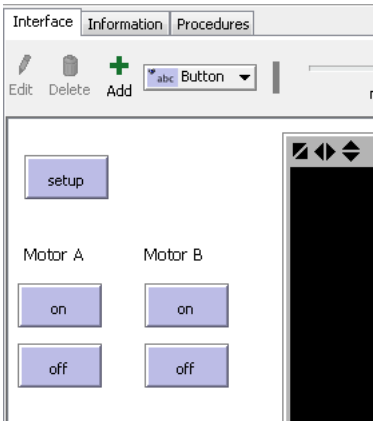
Le second pas consiste à créer un bouton pour connecter NetLogo à la carte GoGo au moyen du port série adéquat de votre système d'exploitation comme décrit ci-dessus.

```
gogo:open "COM1"
;; (pour les machines tournant sous Windows)
```

Une fois le bouton créé, la boîte de dialogue d'édition de bouton "Button" devrait ressembler à :




Commençons maintenant la construction du modèle proprement dit. Imaginez que nous voulions piloter une voiture à quatre roues propulsée par deux moteurs reliés aux roues arrières. Nous supposons que vous avez construit une tel véhicule et connecté les moteurs aux port de sortie "a" et "b" de la carte GoGo. Une approche très simple consisterait à créer deux boutons pour chaque moteur, "on" et "off" :




Le code associé à ces boutons est très simple : pour le bouton "on", nous pouvons simplement avoir :

```
gogo:talk-to-output-ports ["a"]
gogo:output-port-on
```

Pour le bouton "off", le code sera très semblable :

```
gogo:talk-to-output-ports ["a"]
gogo:output-port-off
```

Les boutons "on" et "off" de l'autre ensemble , utilisés pour contrôler le second moteur, auront un code similaire, sauf que nous devons nous adresser au second port de sortie ("b"), de la manière suivante :

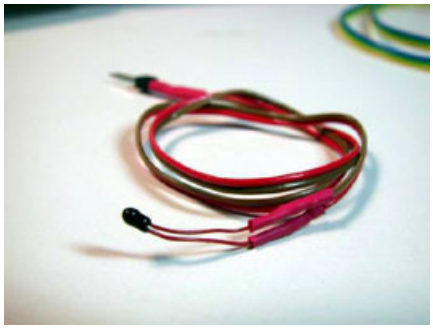
```
gogo:talk-to-output-ports ["b"]
```

Nous pouvons rendre notre modèle plus intéressant en ajoutant un « inverseur de direction ». Ajoutez un bouton avec le code suivant qui inversera le sens de rotation des moteurs "a" et "b" :

```
gogo:talk-to-output-ports ["a" "b"]
gogo:output-port-reverse
```

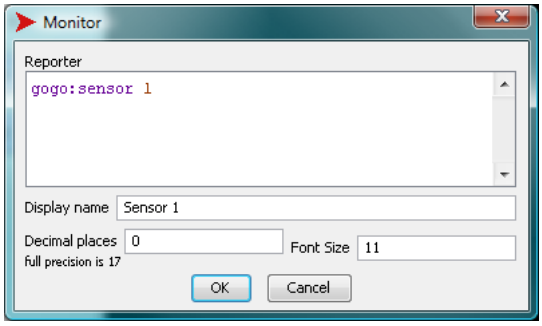
## Un projet de capteur simple

Pour créer un projet de capteur simple, nous supposerons que vous avez ajouté l'extension GoGo à votre modèle et ouvert avec succès une connexion avec la carte GoGo, c'est-à-dire ajouté la commande `extensions` dans le panneau "Procedures" et un bouton "setup" comme décrit précédemment. Pour ce projet de capteur, nul besoin de moteurs; nous avons besoin d'un autre périphérique, un capteur de température (cliquez pour avoir plus d'information sur un capteur de température type sur le site de Digikey). Pour savoir où acheter et comment assembler un capteur de température, lisez le tutoriel "Making Sensors" sur le site web de la carte GoGo. Voilà à quoi ressemble un capteur de température une fois qu'il a été construit :



La manière la plus simple d'utiliser un capteur de température est évidemment d'afficher la température mesurée. Nous pouvons y parvenir en ajoutant un moniteur dans l'interface de NetLogo et en lui fournissant le code suivant :

```
gogo:sensor 1
```




Les valeurs fournies par le capteur sont des nombres arbitraires appartenant à un certain ensemble donné et doivent être converties en véritables unités de température telles que les degrés Celsius, Fahrenheit ou même, pourquoi pas en Kelvin. Pour tous les capteurs, la carte GoGo retourne des valeurs allant de 0 à 1023. Tous les capteurs sont livrés avec une documentation contenant soit une formule de conversion, soit une table de conversion permettant de transformer l'ensemble de valeurs arbitraire 0-1023 en valeurs exprimées en véritables unités physiques. Supposons que la documentation de notre capteur de température contienne une formule de conversion du genre `degrés = valeur arbitraire / 30`. La commande placée dans le moniteur de l'interface NetLogo pourra être transformée en :

```
gogo:sensor 1 / 30
```

La valeur retournée par le capteur pourrait aussi être utilisée pour piloter des objets sur l'écran, tels que des tortues. Créons par exemple deux boutons : un bouton appelé "create one turtle" (créer une tortue) qui efface le monde et crée une tortue et un bouton "forever" "move with heat" (déplace avec la température) qui fait avancer la tortue en fonction de la température lue par le capteur.

Le code pourrait ressembler à :

```
to create-one-turtle
  clear-all
  create-turtle
end

to move-with-heat
  if gogo:sensor 1 < 500
    [ forward 1 ]
end
```

Si le bouton forever "move with heat" est activé et que l'utilisateur chauffe le capteur (en le frottant ou en approchant *lentement* une flamme), le seuil de température sera dépassé (< 500) et la tortue avancera. (Notez que nous utilisons un type de capteur de température dans lequel la résistance *diminue* avec la température, ce qui fait que les valeurs fournies par le capteur diminuent quand nous le réchauffons. Il s'agit d'un capteur très répandu et bon-marché).

Une utilisation plus sophistiquée de cet appareillage de mesure serait de commander des matériels périphérique tels que des moteurs. L'utilisateur pourrait, par exemple, enclencher un moteur quand la valeur fournie par le capteur de température atteint 500, au moyen du code suivant :

```
to turn-motor-on-with-heat
 if gogo:sensor 1 < 500
 [
 gogo:talk-to-output-ports ["a"]
 gogo:output-port-on
 ]
end
```

Une autre utilisation possible des résultats retournés par les primitives des capteurs serait de collecter ces données et de les transcrire en graphiques. La collecte peut se révéler utile pour une analyse ultérieure plus élaborée des données et pour faire des comparaisons. Cette collecte peut être exécutée par les commandes NetLogo dédiés aux listes. Par exemple, si l'utilisateur veut enregistrer les valeurs mesurées par le capteur 1 toutes les 0.5 secondes, le code pourrait ressembler à :

```
to log-data-from-sensor
 set data-vector lput (gogo:sensor 1) data-vector
 wait 0.5
end
```

Finalement, la représentation graphique des données est on ne peut plus directe. Le code suivant, par exemple, fera dessiner un graphique avec les données fournies par le capteur 1 :

```
plot (gogo:sensor 1)
```

Pour de plus amples informations concernant les fonctionnalités et les primitives de l'extension GoGo, étudiez les deux exemples de modèles "GoGoMonitor.nlogo" et "GoGoMonitorSimple.nlogo".

# Les primitives de GoGo

`gogo:close` `gogo:open` `gogo:open?` `gogo:ports` `gogo:output-port-coast` `gogo:output-port-off` `gogo:output-port-reverse` `gogo:output-port-[that|this]way` `gogo:ping` `gogo:sensor` `gogo:set-output-port-power` `gogo:talk-to-output-ports`

## gogo:close

```
gogo:close
```

Ferme la connexion établie avec la carte GoGo.

Voir aussi `gogo:open` et `gogo:open?`

## gogo:open

```
gogo:open port-name
```

Ouvre une connexion avec la carte GoGo reliée au port série nommé *port-name*. Voir la primitive `gogo:ports` pour plus d'informations au sujet des noms des ports.

Si la carte GoGo ne répond pas, ou si vous essayez d'ouvrir un port auquel aucune carte GoGo n'est connectée, NetLogo génère une erreur.

Exemple :

```
gogo:open "COM1"
```

Voir aussi `gogo:open?` et `gogo:close`

## gogo:open?

```
gogo:open?
```

Retourne `true` (vrai) si une connexion avec une carte GoGo est ouverte, sinon retourne `false` (faux).

## gogo:ports

```
gogo:ports
```

Retourne une liste contenant les noms des ports série auxquels une carte GoGo *pourrait* être connectée. Avec certains ordinateurs, vous pouvez obtenir une liste comprenant deux ou trois ports série différents. Dans ce cas, essayez de les ouvrir les uns après les autres jusqu'à ce que vous obteniez une connexion utilisable.

## gogo:output-port-coast

```
gogo:output-port-coast
```

Coupe l'alimentation des ports actifs. Si des moteurs y sont connectés, aucune force de freinage n'est appliquée, contrairement à ce que fait la primitive `gogo:output-port-off`. Il s'ensuit que le moteur ralentit peu à peu jusqu'à l'arrêt complet. Avec la majorité des effecteurs périphériques autres que les moteurs, cette primitive produit le même effet que `gogo:output-port-off`. Les ports de sortie affectés par cette commande sont spécifiés par la commande `gogo:talk-to-output-ports`.

Le code suivant active le port de sortie "a" pendant une seconde puis arrête le moteur graduellement :

```
gogo:talk-to-output-ports ["a"]
gogo:output-port-on
wait 1
gogo:output-port-coast
```

### gogo:output-port-off

```
gogo:output-port-off
```

Coupe l'alimentation des ports de sortie. Si l'on utilise des moteurs, une force de freinage est appliquée. Les ports de sortie affectés par cette commande sont spécifiés par la commande `gogo:talk-to-output-ports`.

### gogo:output-port-on

```
gogo:output-port-on
```

Active et alimente les ports de sortie. Les ports de sortie affectés par cette commande sont spécifiés par la commande `gogo:talk-to-output-ports`.

### gogo:output-port-reverse

```
gogo:output-port-reverse
```

Inverse le sens des ports de sortie. Les ports de sortie affectés par cette commande sont spécifiés par la commande `gogo:talk-to-output-ports`.

### gogo:output-port-[that/this]way

```
gogo:output-port-thatway
gogo:output-port-thisway
```

Alimente le port de sortie dans un sens donné. Les ports de sortie peuvent être alimentés dans deux sens, arbitrairement appelés *thisway* et *thatway*. Les ports de sortie affectés par cette commande sont spécifiés par la commande `gogo:talk-to-output-ports`. Notez que cette commande est différente de la commande `gogo:output-port-reverse` car *thisway* et *thatway* auront toujours le même sens à condition que la polarité du connecteur ne change pas.

### gogo:talk-to-output-ports

```
gogo:talk-to-output-ports output-portlist
```

Cette commande active les ports de sortie spécifiés dans l'entrée *`output-portlist`*. Ces ports pourront alors être commandés par des primitives telles que `gogo:output-port-on` et `gogo:output-port-off`. L'utilisateur peut « parler » à un ou plusieurs ports en même temps. Les ports de sorties sont généralement connectés à des moteurs, mais vous pouvez aussi leur connecter des sources de lumière, des LED et des relais. Les ports de sorties sont identifiés par l'une des lettres "a", "b", "c" et "d".

Exemples :

```
;; activer tous les ports de sortie
gogo:talk-to-output-ports [ "a" "b" "c" "d" ]
;; alimenter tous les ports de sortie
gogo:output-port-on
```

```
;; activer les ports de sortie A et D
gogo:talk-to-output-ports [ "a" "d" ]
;; désactiver les ports de sortie A et D,
;; les autres ports de sortie conservent leur état actuel

gogo:output-port-off

gogo:talk-to-output-ports [ "c" "b" ]
;; désactiver les autres ports de sortie
gogo:output-port-off
```

### gogo:ping

`gogo:ping`

Teste l'état de la carte GoGo. Cette commande est principalement utilisée pour s'assurer que la carte est connectée au bon port série. Elle retourne `true` (vrai) si la carte GoGo répond au message de diagnostique qui lui a été envoyé et `false` autrement.

Exemple :

```
print gogo:ping
```

### gogo:sensor

`gogo:sensor` *`sensor`*

Retourne la valeur numérique du capteur nommé *`sensor`*. Les capteurs sont identifiés par les nombres de 1 à 8. Les valeurs retournées vont de 0 à 1023. 1023 est retourné quand aucun capteur n'est relié au port correspondant (résistance maximale), ou quand le capteur est dans un état « ouvert ». La valeur 0 est retournée quand le capteur est court-circuité (résistance nulle).

Exemples :

```
print gogo:sensor 1
;; affiche la valeur du capteur 1

foreach [ 1 2 3 4 5 6 7 8 ]
  [print (word "Sensor " ? " = " gogo:sensor ?)]
;; affiche la valeur de tous les capteurs

if gogo:sensor 1 < 500 [ ask turtles [ fd 10 ] ]
;; fait avancer toutes les tortues de dix pas
;; si la valeur du capteur 1 est inférieure à 500

loop [if gogo:sensor 1 < 500 [ ask turtles [ fd 10 ] ] ]
;; teste continuellement la valeur du capteur 1 et
;; déplace toutes les tortues de 10 pas chaque fois
;; que la valeur du capteur est inférieure à 500.
```

### gogo:set-output-port-power

`gogo:set-output-port-power` *`power-level`*

Spécifie le niveau de puissance des ports de sortie actifs. L'entrée *`power-level`* doit être un nombre compris entre 0 (off) et 7 (pleine puissance). Les ports de sortie concernés par cette commande sont spécifiés par la commande `gogo:talk-to-output-ports` Notez que pour de nombreuses applications réelles, il est plus efficace d'utiliser des appareillages mécaniques, tels que des engrenages et des poulies pour contrôler le couple des moteurs.

Exemple :

```
gogo:talk-to-motors ["a" "b" "c" "d"]
gogo:set-motor-power 4
;; abaisse la puissance de tous les ports de
;; sortie à la moitié de la puissance maximale.
```

# Chapitre 27

# L'extension Profiler

L'extension Profiler livre des informations qui peuvent vous aider à faire en sorte que votre modèle tourne plus rapidement. Elle est constituée d'un ensemble de primitives servant à mesurer combien de fois les procédures du modèle sont appelées pendant une simulation et combien de temps dure chaque appel.

## Avertissements

**Note concernant la compatibilité** : Cette extension demande que votre système utilise Java 1.5 ou plus lorsque vous faites tourner NetLogo car elle utilise des fonctionnalités qui ne sont pas présentes dans Java 1.4. Avec Windows, vous ne devriez pas avoir de problèmes puisque le paquet NetLogo téléchargé contient déjà Java 1.5. Les utilisateurs de Mac doivent avoir au moins Mac OS X 10.4 ou supérieur puisque Apple n'offre pas Java 1.5 avec ses versions 10.2 et 10.3.

**Attention!** L'extension Profiler est nouvelle et expérimentale. Elle n'a pas encore été bien testée et n'est pas encore très conviviale. Nous pensons toutefois que certains utilisateurs la trouveront utile.

## Utilisation

L'extension Profiler est installée automatiquement avec le reste de NetLogo. Pour utiliser l'extension dans votre modèle, ajoutez la ligne de code suivante tout en haut de panneau "Procedures" avant toute autre commande ou définition :

```
extensions [profiler]
```

Si le modèle fait déjà appel à d'autres extensions, la ligne `extensions` est déjà présente dans le code. Il suffit alors d'ajouter `profiler` dans la liste des extensions. Pour en savoir plus sur l'utilisation des extensions NetLogo, consultez le Guides des extensions. Exemple :

```
setup                  ;; initialise le modèle
profiler:start         ;; démarre le profilage
repeat 20 [ go ]       ;; démarre ce que vous voulez mesurer
profiler:stop          ;; arrête le profilage
print profiler:report  ;; montre les résultats
profiler:reset         ;; efface les données
```

**Exemple de code** : "Profiler Example"

# Les primitives de Profiler

profiler:calls profiler:exclusive-time profiler:inclusive-time profiler:start profiler:stop profiler:reset profiler:report

### profiler:calls

```
profiler:calls procedure-name
```

Retourne le nombre de fois que la procédure *procedure-name* a été appelée. Retourne 0 si la procédure *procedure-name* n'est pas définie.

### profiler:exclusive-time

```
profiler:exclusive-time procedure-name
```

Retourne le temps exclusif, en millisecondes, que la procédure *procedure-name* a mis pour faire son travail. Le temps exclusif est le temps qui s'est écoulé entre l'entrée dans la procédure et le moment où elle termine son travail moins le temps utilisé par les procédures qu'elle appelle éventuellement.

Retourne 0 si la procédure *procedure-name* n'est pas définie.

### profiler:inclusive-time

```
profiler:inclusive-time procedure-name
```

Retourne le temps inclusif, en millisecondes, que la procédure *procedure-name* a mis pour faire son travail. Le temps inclusif est le temps qui s'est écoulé entre l'entrée dans la procédure et le moment où elle termine son travail, y compris le temps utilisé par les procédures qu'elle appelle ou les autres opérations qu'elle exécute ou fait exécuter.

Retourne 0 si la procédure *procedure-name* n'est pas définie.

### profiler:start

```
profiler:start
```

Demande au profileur de commencer d'enregistrer les appels des procédures définies par l'utilisateur.

### profiler:stop

```
profiler:stop
```

Demande au profileur de cesser d'enregistrer les appels des procédures définies par l'utilisateur.

### profiler:reset

```
profiler:reset
```

Demande au profileur d'effacer toutes les données collectées.

### profiler:report

`profiler:report`

Retourne une chaîne de caractères contenant un analyse de tous les appels des procédures définies par l'utilisateur. La colonne `Calls` contient le nombre de fois que la procédure a été appelée. La colonne `Incl T(ms)` montre le temps total, en millisecondes, mis par la procédure pour faire son travail, y compris le temps mis par les procédures appelées par elle-même. La colonne `Excl T(ms)` est le temps total mis par la procédure pour faire son travail, mais sans compter le temps mis par les procédures qu'elle a éventuellement appelées. La colonne `Excl/calls` affiche une estimation du temps moyen, en millisecondes, mis par chaque appel de la procédure.

Voici un exemple de sortie :

```
Sorted by Exclusive Time
Name                              Calls Incl T(ms) Excl T(ms) Excl/calls
CALLTHEM                             13     26.066     19.476      1.498
CALLME                               13      6.413      6.413      0.493
REPORTME                             13      0.177      0.177      0.014

Sorted by Inclusive Time
Name                              Calls Incl T(ms) Excl T(ms) Excl/calls
CALLTHEM                             13     26.066     19.476      1.498
CALLME                               13      6.413      6.413      0.493
REPORTME                             13      0.177      0.177      0.014

Sorted by Number of Calls
Name                              Calls Incl T(ms) Excl T(ms) Excl/calls
CALLTHEM                             13     26.066     19.476      1.498
```

# Chapitre 28

# L'extension GIS

Extension GIS version 1.0, diffusée le 16.07.2008

Cette extension donne la possibilité de charger des données GIS vectorielles (points, lignes et polygones) et des données GIS tramées (*raster* ou *grids*) dans NetLogo. L'extension supporte les données vectorielles sous la forme de fichiers au format shapefile ESRI. Le format shapefile (`.shp`) est le format le plus courant pour stocker et échanger des données vectorielles GIS. L'extension supporte les données tramées (*raster*) sous la forme de fichiers ascii grid ESRI. Le format de fichiers ascii grid (`.asc.` ou `.grd`) n'est pas aussi répandu que format shapefile, mais il est supporté en tant que format d'échanges de fichiers par la plupart des plate-formes GIS.

Nous aimerions que vous nous fassiez part de vos suggestions d'améliorations et d'extension de l'extension GIS — ou simplement savoir dans quels buts vous l'utilisez. Envoyez s'il-vous-plaît toute la correspondance à ccl-gis@northwestern.edu.

## Comment l'utiliser

Étudiez le modèle "GeneralExamples.nlogo" situé dans le dossier "gis" du dossier "extensions" de NetLogo pour des exemples montrant la manière d'utiliser cette extension ou le modèle "GradientExample.nlogo" du même dossier pour un exemple plus sophistiqué d'analyse d'ensemble de données tramées.

De manière générale, il faut d'abord définir une transformation pour passer de l'espace de données GIS à l'espace de données NetLogo, charger ensuite les ensembles de données puis leur appliquer les diverses opérations souhaitées. La manière la plus simple de définir une transformation entre l'espace GIS et l'espace (le monde) NetLogo est de faire l'union des « enveloppes » ou des rectangles frontières de tous vos ensembles de données de l'espace GIS et de les plaquer directement sur les frontières du monde NetLogo. Voir le modèle "GeneralExamples.nlogo" pour un exemple de cette technique.

À la place, vous pouvez aussi définir une projection pour l'espace GIS. Dans ce cas, les ensembles de données seront re-projetés pour correspondre à cette projection lorsqu'ils seront chargés dans NetLogo, à condition que chacun de vos fichiers de données possède un fichier `.prj` associé, fichier qui décrit la projection ou le système de coordonnées géographiques utilisé dans ce fichier de données GIS. Si aucun fichier `.prj` associé n'est trouvé, l'extension partira du principe que l'ensemble de données utilise déjà la projection courante, sans tenir compte de ce qu'est cette projection.

Une fois le système de coordonnées défini, vous pouvez chargez les ensembles de données au moyen de la primitive `gis:load-dataset`. Cette primitive retourne soit un ensemble de données vectorielles (*VectorDataset*), soit un ensemble de données tramées (*RasterDataset*), et ce en fonction du type de fichier que vous lui aviez transmis.

Un ensemble de données vectorielles (*VectorDataset*) est une collection d'« objets » vectoriels (VectorFeatures), dont chacun d'eux est un point, une ligne ou un polygone, accompagné d'un ensemble de valeurs pour leurs différentes propriétés. Un ensemble de données vectorielles ne peut contenir qu'un seul de ces trois types d'objets possibles.

Il y a plusieurs choses que vous pouvez faire avec un de ces ensembles de données vectorielles : lui demander les noms des propriétés de ses objets, lui demander quelle est son « enveloppe » (ses limites ou frontières) ou la liste de tous les objets vectoriels de l'ensemble de données, rechercher un seul objet vectoriel ou une liste d'objets vectoriels dont la valeur d'une certaine propriété est plus petite ou plus grande qu'une valeur donnée, ou se trouve dans un certain ensemble de valeurs, ou correspond à une chaîne de caractères donnée en utilisant une correspondance au moyen du caractère générique " * " (qui correspond à n'importe quel nombre d'occurrences de n'importe quel caractère). Si les objets vectoriels sont des polygones, vous pouvez aussi transmettre les valeurs d'une propriété particulière des objets de l'ensemble de données à une variable patch donnée.

Vous pouvez aussi faire plusieurs choses avec un objet vectoriel appartenant à un ensemble de données vectorielles : lui demander la liste des listes de ses sommets, lui demander la valeur d'une propriété désignée par son nom, lui demander son centre de gravité ou demander un sous-ensemble d'un ensemble d'agents donné dont les agents correspondent à l'objet vectoriel donné. Pour les objet de type point, la liste de chaque point est une liste ne contenant qu'un seul élément. Pour les objets de type ligne, la liste de chaque point contient les points de la ligne qui forme cet objet. Pour les objets de type polygone, chaque point de la liste représente un sommet du polygone et le premier point de la liste est le même que le dernier. Les listes de point sont faites de données (valeurs) de type Vertex, et le centre de gravité (*centroid*) est aussi une donnée de type Vertex.

Il y a également toute une série d'opérations définies pour les ensembles de données tramées (*RasterDataset*). La majorité d'entre elles comprend l'échantillonnage des valeurs de l'ensemble de données ou le ré-échantillonnage d'une trame dans une autre résolution. Vous pouvez aussi transmettre une trame à une variable patch donnée et « envelopper » une trame en utilisant une matrice de convolution arbitraire.

## Problèmes connus

La possibilité d'exporter et d'importer des données de type *RasterDataset*, *VectorDataset*, *VectorFeature* et *Vertex* en utilisant les primitives `export-world` et `import-world` n'a pas encore été implémentée.

Il n'y a, pour le moment, aucune possibilité de distinguer les zones polygonales positives (« couvercles ») des zones polygonales négatives (« trous ») ou de déterminer quels trous sont associés à quel couvercles.

## Les primitives GIS

### Primitives pour les systèmes de coordonnées

`set-transformation` `set-transformation-ds` `set-world-envelope` `set-world-envelope-ds` `world-envelope` `envelope-of` `envelope-union-of` `load-coordinate-system` `set-coordinate-system`

### Primitives pour les ensembles de données (Dataset)

`load-dataset` `store-dataset` `type-of` `patch-dataset` `turtle-dataset` `link-dataset`

### Primitives pour les ensembles de données vectorielles (VectorDataset)

`shape-type-of` `property-names` `feature-list-of` `vertex-lists-of` `centroid-of` `location-of` `property-value` `find-features` `find-one-feature` `find-less-than` `find-greater-than` `find-range` `property-minimum` `property-maximum` `apply-coverage` `coverage-minimum-threshold` `set-coverage-minimum-threshold` `coverage-maximum-threshold` `set-coverage-maximum-threshold` `intersects?` `contains?` `contained-by?` `have-relationship?` `relationship-of`

### Primitives pour les ensembles de données tramées (RasterDataset)

`width-of` `height-of` `raster-value` `set-raster-value` `minimum-of` `maximum-of` `sampling-method-of` `set-sampling-method` `raster-sample` `raster-world-envelope` `create-raster` `resample` `convolve` `apply-raster` `read-raster`

### Primitives de dessin

`drawing-color` `set-drawing-color` `draw` `fill` `paint`

## Primitives pour les systèmes de coordonnées

### gis:set-transformation

`gis:set-transformation` *`enveloppe-gis enveloppe-netlogo`*

Définit une correspondance entre les coordonnées GIS et les coordonnées NetLogo, correspondance permettant de transformer les coordonnées GIS en coordonnées NetLogo. Les arguments *`enveloppe-gis`* et *`enveloppe-netlogo`* doivent être des listes constituées chacune des quatre éléments suivants :

```
[ minimum-x maximum-x minimum-y maximum-y ]
```

La grandeur de la transformation sera égale à la valeur minimale nécessaire pour établir la correspondance entre les deux ensembles des valeurs X et à celle nécessaire pour établir la correspondance entre les deux ensembles des valeurs Y. L'espace GIS sera centré dans l'espace NetLogo.

Par exemple, les deux listes suivantes feront correspondre tout l'espace géographique (les coordonnées géographiques, latitude et longitude, exprimées en degrés) à l'espace du monde NetLogo, sans tenir compte des dimensions courantes du monde NetLogo :

```
(list -180 180 -90 90 )
(list min-pxcor max-pxcor min-pycor max-pycor)
```

Toutefois, si vous mettez en place l'enveloppe du monde NetLogo, vous devrez probablement utiliser la primitive `set-world-envelope` décrite ci-dessous.

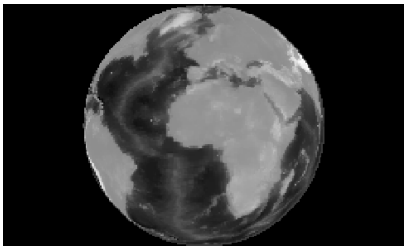
### gis:set-transformation-ds

`gis:set-transformation-ds` *`enveloppe-gis enveloppe-netlogo`*

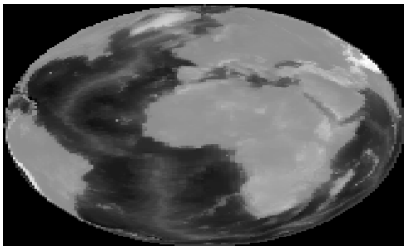
Cette primitive fait la même chose que la primitive `set-transformation` ci-dessus, sauf qu'elle permet d'utiliser des échelles de transformations différentes pour les valeurs X et les valeurs Y. Le suffixe `-ds` signifie "*different scales*" (échelles différentes). L'utilisation d'échelles différentes pour les deux axes provoquera

une distorsion de la forme des objets GIS, c'est pourquoi elle n'est généralement pas recommandée, mais elle peut être utile pour certains modèles.

Voici un exemple montrant la différence entre `set-transformation` et `set-transformation-ds` :



En utilisant `set-transformation`, l'échelle utilisée pour l'axe X est la même que celle utilisée pour l'axe Y, dont préservant la forme ronde de la Terre dans cette projection orthographique.



En utilisant `set-transformation-ds`, l'échelle de l'axe X est étirée de manière à ce que l'image de la Terre couvre la totalité de la Vue NetLogo, provoquant ainsi une déformation de la rotondité de la Terre.

## gis:set-world-envelope

```
gis:set-world-envelope enveloppe-gis
```

Cette primitive est un raccourci pour mettre en place la transformation en plaquant l'enveloppe du monde NetLogo à l'enveloppe donnée dans l'espace GIS, tout en gardant la même échelle de transformation pour les axes X et Y. Elle est l'équivalent de la commande :

```
set-transformation gis-envelope (list min-pxcor max-pxcor min-pycor max-pycor)
```

Cette primitive est fournie parce que, la plupart du temps, on veut faire en sorte que l'enveloppe s'étende à la totalité du monde NetLogo plutôt qu'à une partie de ce monde seulement.

## gis:set-world-envelope-ds

```
gis:set-world-envelope-ds enveloppe-gis
```

Cette primitive est un raccourci pour mettre en place la transformation en plaquant l'enveloppe du monde NetLogo à l'enveloppe donnée dans l'espace GIS, en utilisant des échelles de transformation différentes pour les axes X et Y si nécessaire. Elle est l'équivalent de la commande :

```
set-transformation-ds gis-envelope (list min-pxcor max-pxcor min-pycor max-pycor)
```

Voir les images ci-dessus illustrant la différence entre l'utilisation d'échelles semblables et d'échelles différentes pour les axes de coordonnées X et Y.

## gis:world-envelope

```
gis:world-envelope
```

Ce reporter retourne l'enveloppe (le rectangle englobant) du monde NetLogo, transformée dans l'espace GIS. Cette enveloppe est une liste de quatre éléments de la forme :

```
[ minimum-x maximum-x minimum-y maximum-y ]
```

### gis:envelope-of

```
gis:envelope-of chose
```

Ce reporter retourne l'enveloppe (rectangle englobant) de la *chose* en coordonnées GIS. La *chose* peut être un agent, un ensemble d'agents (*AgentSet*), un ensemble de données raster (*RasterDataset*), un ensemble de données vectorielles (*VectorDataset*) ou un objet vectoriel (*VectorFeature*). Une enveloppe est formée d'une liste de quatre éléments, de la forme :

```
[ minimum-x maximum-x minimum-y maximum-y ]
```

### gis:envelope-union-of

```
gis:envelope-union-of enveloppe1 enveloppe2
(gis:envelope-union-of enveloppe1 ...)
```

Ce reporter retourne une enveloppe (rectangle englobant) qui contient entièrement les enveloppes données. Une enveloppe est formée d'une liste de quatre éléments, de la forme :

```
[ minimum-x maximum-x minimum-y maximum-y ]
```

Aucune supposition n'est faite concernant les systèmes de coordonnées des arguments, ce qui fait que s'ils n'ont pas les mêmes systèmes, les résultats seront imprévisibles.

### gis:load-coordinate-system

```
gis:load-coordinate-system fichier
```

Charge une nouvelle projection globale qui sera utilisée pour projeter ou re-projeter des données GIS quand elles sont chargées à partir d'un fichier. Ce fichier doit contenir une description de projection Well-Known Text (WKT) valable.

Les fichiers de projection WKT sont fréquemment distribués avec les fichiers de données GIS et ont habituellement `.prj` comme extension de nom de fichier.

Les chemins de fichiers relatifs sont résolus relativement à l'emplacement du modèle courant, ou au dossier `home` de l'utilisateur si le modèle courant n'a pas encore été sauvegardé.

L'extension GIS ne supporte pas tous les systèmes de coordonnées et tous les types de projection WKT. Seuls les systèmes de coordonnées géographiques (`"GEOGCS"`) et les systèmes de projection de coordonnées (`"PROJCS"`) sont supportés. Parmi les systèmes de projections de coordonnées, seuls les systèmes suivants sont supportés :

| | | |
|---|---|---|
| Lambert_Conformal_Conic_2SP | Polycônique | Robinson |
| Lambert_Azimuthal_Equal_Area | Mercator_1SP | Stéréographique |
| Azimuthal_Equidistant | Miller | Mercator Transverse |
| Cylindrical_Equal_Area | Mercator Oblique | Orthographique |
| Cônique équidistant | hotine_oblique_mercator | Gnomonique |

Voir le site remotesensing.org pour une liste complète des projections WKT et leurs paramètres.

### gis:set-coordinate-system

```
gis:set-coordinate-system systeme
```

Met en place le système de projection global utilisé pour projeter ou re-projeter les données GIS quand elles sont chargées. Le paramètre *systeme* doit être soit une chaîne au format Well-Known Text (WKT), soit une liste NetLogo formée des données WKT converties en une liste en déplaçant chaque mot-clé entouré de ses parenthèses qu'on entoure encore de guillemets. Cette dernière technique est préférable car elle produit un code bien plus lisible.

Cette primitive est soumise aux mêmes restrictions quant aux WKT supportés que la primitive `load-coordinate-system` décrite ci-dessus.

## Primitives pour les ensembles de données (Dataset)

### gis:load-dataset

```
gis:load-dataset fichier
```

Charge le fichier de données *fichier*, et en cas de nécessité, re-projette les données si une projection globale est définie et si un fichier `".prj"` est associé à *fichier*, puis retourne l'ensemble de données résultant.

S'il n'y a pas de fichier `".prj"`, alors `load-dataset` présuppose que la projection des données chargées est la même que celle du système de coordonnées global courant.

Les chemins de fichiers relatifs sont résolus par rapport à l'emplacement du modèle courant, ou du dossier `home` de l'utilisateur si le modèle courant n'a pas encore été sauvegardé.

Pour le moment, NetLogo supporte deux types de fichiers de données :

- "**.shp**" (shapefile ESRI) — contient des données vectorielles définissant des points, des lignes ou des polygones. Quand le fichier à charger est un fichier « shapefile », `load-dataset` retourne un ensemble de données vectorielles « *VectorDataset* »
- "**.asc**" ou "**.grd**" (ESRI ascii grid) — contient des données tramées (*raster*) consistant en une trame de valeurs. Quand le fichier à charger est un fichier « ascii grid », le reporter `load-dataset` retourne un ensemble de données tramées « *RasterDataset* ».

### gis:store-dataset

```
gis:store-dataset dataset file
```

Sauvegarde l'ensemble de données *dataset* dans le fichier *file*. Si le nom du fichier n'a pas l'extension de nom voulue, cette dernière sera automatiquement ajoutée. Les chemins de fichiers relatifs sont résolus par rapport à l'emplacement du modèle courant, ou du dossier `home` de l'utilisateur si le modèle courant n'a pas encore été sauvegardé.

Pour le moment, cette primitive ne fonctionne que pour les ensembles de données de type « *RasterDataset* » et ne peut enregistrer des ensembles de données que sous la forme de fichiers ESRI ascii grid.

### gis:type-of

```
gis:type-of dataset
```

Retourne le type de l'ensemble de données GIS *dataset* : soit `"VECTOR"`, soit `"RASTER"`.

### gis:patch-dataset

```
gis:patch-dataset variable-patch
```

Retourne une nouvelle trame dont les cellules correspondent directement aux patches de NetLogo, et dont les valeurs des cellules correspondent aux valeurs de la *variable-patch* donnée. Cette primitive est fondamentalement l'inverse de `apply-raster`; `apply-raster` copie les valeurs d'un ensemble de données tramées dans une variable patch, et `read-raster` copie les valeurs d'une variable patch dans un ensemble de données tramées.

### gis:turtle-dataset

```
gis:turtle-dataset ensemble-tortues
```

Retourne un nouvel ensemble de données *VectorDataset* de type point construit à partir des tortues de l'ensemble d'agents *ensemble-tortues*. Les points correspondent aux emplacements des tortues, translatés de l'espace NetLogo dans l'espace GIS en utilisant le système de transformation des coordonnées courant. Les propriétés de l'ensemble de données sont celles de toutes les variables tortues communes à chaque tortue de l'ensemble d'agents.

### gis:link-dataset

```
gis:link-dataset ensemble-liens
```

Retourne un nouvel ensemble de données *VectorDataset* de type ligne construit à partir des liens de l'ensemble d'agents *ensemble-liens* donné. Les points situés aux extrémités de chaque ligne correspondent aux positions des tortues connectées par chaque lien, translatées de l'espace NetLogo dans l'espace GIS en utilisant le système de transformation des coordonnées courant. Les propriétés de l'ensemble de données sont celles de toutes les variables liens communes à chaque lien de l'ensemble d'agents.

## Primitives pour ensembles de données vectorielles (VectorDataset)

### gis:shape-type-of

```
gis:shape-type-of VectorDataset
```

Retourne le type des formes stockées dans l'ensemble de données *VectorDataset*. Les valeurs retournées possibles sont : `"POINT"`, `"LINE"` et `"POLYGON"`.

### gis:property-names

```
gis:property-names VectorDataset
```

Retourne une liste de chaînes de caractères où chaque chaîne est le nom d'une propriété commune à chaque objet vectoriel (*VectorFeature*) de l'ensemble de données *VectorDataset* donné, propre à être utilisée avec la primitive `gis:property-value`.

### gis:feature-list-of

```
gis:feature-list-of VectorDataset
```

Retourne une liste de tous les objets vectoriels (*VectorFeatures*) de l'ensemble de donnée *VectorDataset*.

### gis:vertex-lists-of

```
gis:vertex-lists-of VectorFeature
```

Retourne une liste de listes de valeurs de points (*Vertex*). Pour les ensembles de données décrivant des points, chaque liste de points ne contient qu'un seul point : l'emplacement du point. Pour les ensembles de données décrivant des lignes, chaque liste de points contient au moins deux points et représente une ligne polygonale joignant chaque paire de points adjacents de la liste. Pour les ensembles de données de type polygone, chaque liste de points contient au moins trois points représentant un polygone dont ils sont les sommets; de plus, le premier et le dernier point de la liste sont les mêmes car les objets de type polygones sont des figures « fermées ».

### gis:centroid-of

```
gis:centroid-of VectorFeature
```

Retourne un point unique représentant le centre de gravité (*centroid*) de l'objet `Vectorfeature` donné. Pour des ensembles de données de type point, le centre de gravité est défini comme étant la moyenne des positions de tous les points de l'objet. Pour les ensembles de données de type ligne, le centre de gravité est défini comme étant la moyenne des positions des points médians, pondérés par la longueur du segment, de tous les segments de droite de l'objet. Pour les ensembles de données de type polygone, le centre de gravité est défini comme étant la somme pondérée des centres de gravité des triangles obtenus par la décomposition du polygone en triangles (qui peuvent se recouvrir). Voir le chapitre FAQ pour plus de détails concernant l'algorithme du centre de gravité des polygones.

### gis:location-of

```
gis:location-of Vertex
```

Retourne une liste de deux éléments contenant les valeurs x et y (dans cet ordre) du point `Vertex` donné translaté dans l'espace du monde NetLogo en utilisant la transformation courante, ou une liste vide si le point donné se trouve en dehors des limites du monde NetLogo.

### gis:property-value

```
gis:property-value VectorFeature nom-propriete
```

Retourne la valeur de la propriété nommée `nom-propriete` de l'ensemble de données `VectorFeature`. La valeur retournée peut être un nombre, une chaîne de caractères ou une valeur booléenne, en fonction du type du champ du fichier de données sous-jacent.

Pour les fichiers « *shapefile* », les valeurs des champs dBase `CHARACTER` et `DATE` sont retournées sous forme de chaînes de caractères, celles des champs `NUMBER` et `FLOAT` sous formes de nombres et celles des champs `LOGICAL` sous forme de valeurs booléennes. Les champs `MEMO` ne sont pas supportés. Les valeurs des champs `DATE` sont converties en chaînes de caractères conformément au format ISO 8601 (`AAAA-MM-JJ`).

### gis:find-features

```
gis:find-features VectorDataset nom-propriete chaine
```

Retourne une liste de tous les objets (*VectorFeatures*) de l'ensemble de données `VectorDataset` dont la valeur de la propriété `nom-propriete` correspond la `chaine` donnée. La comparaison ne tient pas compte de la casse des caractères, et le caractère « jocker » `"*"` peut remplacer un nombre quelconque d'occurrences de n'importe quel caractère (y compris le zéro).

### gis:find-one-feature

`gis:find-one-feature` *`VectorDataset nom-propriete chaine`*

Retourne le premier objet (*VectorFeature*) de l'ensemble de données *`VectorDataset`* dont la valeur de la propriété *`nom-propriete`* correspond à la *`chaine`* donnée. La comparaison ne tient pas compte de la casse des caractères et le caractère «jocker » `"*"` peut remplacer un nombre quelconque d'occurrences de n'importe quel caractère (y compris le zéro). Les objets sont explorés en suivant l'ordre dans lequel ils apparaissent dans le fichier de données qui est à l'origine de l'ensemble de données, et la recherche cesse dès qu'une correspondance est trouvée. La primitive retourne `nobody` (personne) si aucun objet satisfaisant la condition n'est trouvé.

### gis:find-less-than

`gis:find-less-than` *`VectorDataset nom-propriete valeur`*

Retourne une liste de tous les objets (*VectorFeature*) de l'ensemble de données *`VectorDataset`* dont la valeur de la propriété *`nom-propriete`* est inférieure à la *`valeur`* donnée. Les chaînes de caractères sont comparées dans l'ordre lexicographique en tenant compte de la casse des caractères tel que défini dans Java Documentation. L'utilisation d'une valeur de type chaîne de caractères pour une propriété numérique ou d'une valeur numérique pour une propriété de type chaîne de caractères provoque une erreur.

### gis:find-greater-than

gis:find-greater-than *VectorDataset nom-propriete valeur*

Retourne une liste de tous les objets (*VectorFeature*) de l'ensemble de données *`VectorDataset`* dont la valeur de la propriété *`nom-propriete`* est plus grande que la *`valeur`* donnée. Les chaînes de caractères sont comparées dans l'ordre lexicographique en tenant compte de la casse des caractères tel que défini dans Java Documentation. L'utilisation d'une valeur de type chaîne de caractères pour une propriété numérique ou d'une valeur numérique pour une propriété de type chaîne de caractères provoque une erreur.

### gis:find-range

`gis:find-range` *`VectorDataset nom-propriete valeur-min valeur-max`*

Retourne une liste de tous les objets (*VectorFeature*) de l'ensemble de données *`VectorDataset`* dont la valeur de la propriété *`nom-propriete`* est strictement supérieure à la *`valeur-min`* et strictement inférieure à la *`valeur-max`* données. Les chaînes de caractères sont comparées dans l'ordre lexicographique en tenant compte de la casse des caractères tel que défini dans Java Documentation. L'utilisation d'une valeur de type chaîne de caractères pour une propriété numérique ou d'une valeur numérique pour une propriété de type chaîne de caractères provoque une erreur.

### gis:property-minimum

`gis:property-minimum` *`VectorDataset nom-propriete`*

Retourne la plus petite valeur pour la propriété *`nom-propriete`* de tous les objets (*VectorFeature*) de l'ensemble de données *`VectorDataset`*. Les chaînes de caractères sont comparées dans l'ordre lexicographique en tenant compte de la casse des caractères tel que défini dans Java Documentation.

### gis:property-maximum

```
gis:property-maximum VectorDataset nom-propriete
```

Retourne la plus grande valeur pour la propriété *nom-propriete* de tous les objets (*VectorFeature*) de l'ensemble de données *VectorDataset*. Les chaînes de caractères sont comparées dans l'ordre lexicographique en tenant compte de la casse des caractères tel que défini dans Java Documentation.

### gis:apply-coverage

```
gis:apply-coverage VectorDataset nom-propriete variable-patch
```

Copie les valeurs de la propriété *nom-propriete* des objets de l'ensemble de données *VectorDataset* dans la variable *variable-patch* donnée. L'ensemble de données doit être un ensemble de type `polygones`; les types `point` et `ligne` n'étant pas supportés.

Pour chaque patch, la primitive cherche tous les objets qui coupent le patch. Ensuite, si la propriété est de type chaîne de caractères, elle cherche la plus grande valeur en calculant l'aire de la partie du patch recouverte par les objets vectoriels ayant chaque valeur possible de la propriété, puis retourne la valeur représentant la plus grande proportion de la surface du patch. Si la propriété est de type numérique, elle calcule une moyenne pondérée des valeurs de la propriété de tous les objets vectoriels qui coupent le patch, pondération basée sur la proportion de la surface du patch que les objets recouvrent.

Il y a deux exceptions à ce comportement par défaut :

- ✔ Si un pourcentage de la surface du patch plus grand que la limite maximale de couverture est recouvert par un seul objet, alors la valeur de la propriété de cet objet est copiée directement. Si plus d'un objet couvre une plus grande proportion de la surface du patch que la limite, seul le premier sera utilisé.
- ✔ Si le pourcentage total de la surface d'un patch recouvert par un objet est inférieure à la limite minimale de couverture, la *variable-patch* du patch cible reçoit la valeur `"Not a Number"`.

Par défaut, la limite minimale est de 10% et la limite maximale de 33%. Ces valeurs peuvent être modifiées en utilisant les primitives qui suivent.

### gis:coverage-minimum-threshold

```
gis:coverage-minimum-threshold
```

Retourne la limite de recouvrement minimale utilisée par la primitive `gis:apply-coverage`.

### gis:set-coverage-minimum-threshold

```
gis:set-coverage-minimum-threshold nouvelle-limite
```

Fixe la limite de recouvrement minimale *nouvelle-limite* devant être utilisée par la primitive `gis:apply-coverage`.

### gis:coverage-maximum-threshold

```
gis:coverage-maximum-threshold
```

Retourne la limite de recouvrement maximale utilisée par la primitive `gis:apply-coverage`.

### gis:set-coverage-maximum-threshold

```
gis:set-coverage-maximum-threshold nouvelle-limite
```

Fixe la limite de recouvrement maximale *nouvelle-limite* devant être utilisée par la primitive `gis:apply-coverage`.

### gis:intersects?

```
gis:intersects? objet-x objet-y
```

Retourne `true` (vrai) si les représentations spatiales des objets *objet-x* et *objet-y* ont au moins un point commun, sinon retourne `false` (faux). Les *objet-x* et *objet-y* peuvent être de plusieurs types.

- ✔ Un ensemble de données vectorielles (*VectorDataset*) : dans ce cas, la représentation spatiale de l'objet correspond à l'union de tous les points, lignes et polygones que contient l'ensemble de données.
- ✔ Un objet vectoriel (*VertorFeature*) : dans ce cas, la représentation spatiale de l'objet est définie par le point, la ligne ou le polygone que l'objet contient.
- ✔ Une tortue : dans ce cas, la représentation spatiale est un point.
- ✔ Un lien : sa représentation spatiale est une ligne reliant les deux points représentant les tortues que le lien relie.
- ✔ Un patch : sa représentation spatiale est un polygone rectangulaire.
- ✔ Un ensemble d'agents : sa représentation spatiale est l'union des représentations spatiales de tous les agents que l'ensemble contient.
- ✔ Une liste pouvant contenir un nombre quelconque des objets décrits ici, y compris d'autres listes. La représentation spatiale d'une telle liste est l'union des représentations spatiales de ses éléments.

### gis:contains?

```
gis:contains? x y
```

Retourne `true` (vrai) si chaque point de la représentation spatiale *x* fait aussi partie de la représentation spatiale *y*. Notez que cela signifie que les polygones contiennent leurs limites. Les objets *x* et *y* possibles sont listés ci-dessous.

- ✔ Un ensemble de données vectorielles (*VectorDataset*) : dans ce cas, la représentation spatiale de l'objet correspond à l'union de tous les points, lignes et polygones que contient l'ensemble de données.
- ✔ Un objet vectoriel (*VertorFeature*) : dans ce cas, la représentation spatiale de l'objet est définie par le point, la ligne ou le polygone que l'objet contient.
- ✔ Une tortue : dans ce cas, la représentation spatiale est un point.
- ✔ Un lien : sa représentation spatiale est une ligne reliant les deux points représentant les tortues que le lien relie.
- ✔ Un patch : sa représentation spatiale est un polygone rectangulaire.
- ✔ Un ensemble d'agents : sa représentation spatiale est l'union des représentations spatiales de tous les agents que l'ensemble contient.
- ✔ Une liste pouvant contenir un nombre quelconque des objets décrits ici, y compris d'autres listes. La représentation spatiale d'une telle liste est l'union des représentations spatiales de ses éléments.

## gis:contained-by?

```
gis:contained-by? x y
```

Retourne `true` (vrai) si chaque point de la représentation spatiale *x* fait aussi partie de la représentation spatiale *y*. Les objets *x* et *y* possibles sont listés ci-dessous.

- Un ensemble de données vectorielles (*VectorDataset*) : dans ce cas, la représentation spatiale de l'objet correspond à l'union de tous les points, lignes et polygones que contient l'ensemble de données.
- Un objet vectoriel (*VertorFeature*) : dans ce cas, la représentation spatiale de l'objet est définie par le point, la ligne ou le polygone que l'objet contient.
- Une tortue : dans ce cas, la représentation spatiale est un point.
- Un lien dont la représentation spatiale est une ligne reliant les deux points représentant les tortues que le lien relie.
- Un patch : sa représentation spatiale est un polygone rectangulaire.
- Un ensemble d'agents : sa représentation spatiale est l'union des représentations spatiales de tous les agents que l'ensemble contient.
- Une liste pouvant contenir un nombre quelconque des objets décrits ici, y compris d'autres listes. La représentation spatiale d'une telle liste est l'union des représentations spatiales de ses éléments.

## gis:have-relationship?

```
gis:have-relationship? x y relations
```

Retourne `true` (vrai) si les représentations spatiales des deux objets *x* et *y* satisfont aux relations spatiales spécifiées par l'argument *relations*, sinon retourne `false` (faux). Les relations spatiales sont spécifiées à l'aide d'une matrice Dimensionally Extended Nine-Intersection Model (DE-9IM). Cette matrice est formée de neuf éléments, chacun spécifiant la relation requise entre les espaces internes, les espaces frontières et les espaces externes des deux objets. Ces éléments doivent avoir l'une des six valeurs possibles suivantes :

- " **T** " signifie qu'il doit y avoir une intersection quelconque entre les deux espaces.
- " **F** " signifie qu'il ne doit y avoir aucune intersection entre les deux objets.
- " **0** " signifie que la dimension de l'espace de l'intersection des deux objets doit être de zéro (c'est à dire que l'intersection doit être un point ou un ensemble non vide de points).
- " **1** " signifie que la dimension de l'espace de l'intersection des deux objets doit être de un (c'est à dire que l'intersection doit être une ligne ou un ensemble non vide de segments de ligne).
- " **2** " signifie que la dimension de l'espace de l'intersection des deux objets doit être de deux (c'est à dire que l'intersection doit être un polygone ou un ensemble non vide de polygones dont l'aire est supérieure à zéro).
- " * " signifie que les deux espaces peuvent avoir n'importe quel type de relations.

Par exemple, la matrice :

| | | x | | |
|---|---|---|---|---|
| | | Intérieur | Frontière | Extérieur |
| y | Intérieur | T | * | * |
| | Frontière | * | * | * |
| | Extérieur | F | F | * |

retournera `true` si et seulement si une partie de l'espace intérieur de l'objet *x* se trouve dans l'espace intérieur de l'objet *y*, et si aucune partie de l'espace intérieur de l'objet *x* ou de son espace frontière n'entre dans l'espace externe de l'objet *y*. C'est essentiellement une forme plus restrictive de la primitive `contains?`, primitive qui considère que les polygones ne contiennent pas leurs frontières.

La matrice est transmise à la primitive `have-relationship?` sous la forme d'une chaîne de caractères dont les éléments sont donnés dans l'ordre suivants :

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

Donc, pour utiliser la matrice de l'exemple ci-dessus, nous devons écrire :

```
gis:have-relationship? x y "T*****FF*"
```

Une description formelle et bien plus détaillée de la matrice DE-9IM et de la théorie des ensembles de points qui lui est associée est disponible à OpenGIS Simple Features Specification for SQL.

Les objets *x* et *y* peuvent être :

- ✔ Un ensemble de données vectorielles (*VectorDataset*) : dans ce cas, la représentation spatiale de l'objet correspond à l'union de tous les points, lignes et polygones que contient l'ensemble de données.
- ✔ Un objet vectoriel (*VertorFeature*) : dans ce cas, la représentation spatiale de l'objet est définie par le point, la ligne ou le polygone que l'objet contient.
- ✔ Une tortue : dans ce cas, la représentation spatiale est un point.
- ✔ Un lien dont la représentation spatiale est une ligne reliant les deux points représentant les tortues que le lien relie.
- ✔ Un patch : sa représentation spatiale est un polygone rectangulaire.
- ✔ Un ensemble d'agents : sa représentation spatiale est l'union des représentations spatiales de tous les agents que l'ensemble contient.
- ✔ Une liste pouvant contenir un nombre quelconque des objets décrits ici, y compris d'autres listes. La représentation spatiale d'une telle liste est l'union des représentations spatiales de ses éléments.

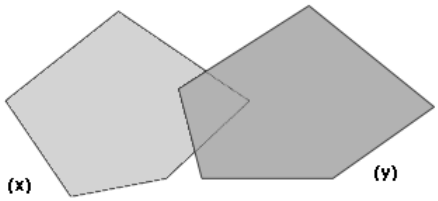
### gis:relationship-of

`gis:relationship-of x y`

Retourne la matrice Dimensionally Extended Nine-Intersection Model (DE-9IM) qui décrit les relations spatiales des deux objets *x* et *y*. Cette matrice est formée de neuf éléments, chacun décrivant la relation entre les espaces internes, les espaces frontières et les espaces externes des deux objets ce qui signifie qu'ils peuvent avoir l'une des quatre valeurs possibles suivantes :

- ✔ " **-1** " signifie qu'il n'y a pas d'intersection entre les espaces.
- ✔ " **0** " signifie que la dimension de l'intersection des espaces est zéro (c'est-à-dire que l'intersection contient un point ou un ensemble de points).
- ✔ " **1** " signifie de la dimension de l'intersection des espaces est 1 (c'est-à-dire que l'intersection contient une ou plusieurs lignes).
- ✔ " **2** " signifie de la dimension de l'intersection des espaces est 2 (c'est-à-dire que l'intersection contient un polygone non vide).

Par exemple, les deux polygones x et y montrés ici :




ont la matrice DE-9IM suivante :

| | | x | | |
|---|---|---|---|---|
| | | Intérieur | Frontière | Extérieur |
| y | Intérieur | 2 | 1 | 2 |
| | Frontière | 1 | 0 | 1 |
| | Extérieur | 2 | 1 | 2 |

qui serait retournée par `relationship-of` sous la forme de la chaîne de caractères `"212101212"`.

Une description formelle et bien plus détaillée de la matrice DE-9IM et de la théorie des ensembles de points qui lui est associée est disponible à OpenGIS Simple Features Specification for SQL.

Les objets *x* et *y* peuvent être :

- ✔ Un ensemble de données vectorielles (*VectorDataset*) : dans ce cas, la représentation spatiale de l'objet correspond à l'union de tous les points, lignes et polygones que contient l'ensemble de données.
- ✔ Un objet vectoriel (*VertorFeature*) : dans ce cas, la représentation spatiale de l'objet est définie par le point, la ligne ou le polygone que l'objet contient.
- ✔ Une tortue : dans ce cas, la représentation spatiale est un point.
- ✔ Un lien dont la représentation spatiale est une ligne reliant les deux points représentant les tortues que le lien relie.
- ✔ Un patch : sa représentation spatiale est un polygone rectangulaire.
- ✔ Un ensemble d'agents : sa représentation spatiale est l'union des représentations spatiales de tous les agents que l'ensemble contient.
- ✔ Une liste pouvant contenir un nombre quelconque des objets décrits ici, y compris d'autres listes. La représentation spatiale d'une telle liste est l'union des représentations spatiales de ses éléments.

### gis:intersecting

*ensemble-de-patch* `gis:intersecting` *donnees*

Retourne un nouvel ensemble d'agents ne contenant que les agents de l'ensemble d'agents *ensemble-de-patch* qui ont une intersection avec les données GIS, *donnees* qui peuvent être un ensemble de données vectorielles *VectorDataset*, un objet vectoriel *VectorFeature*, un agent, un ensemble d'agents ou une liste contenant n'importe lesquels des objets décrits ci-dessus.

## Primitives pour les ensembles de données tramées (RasterDataset)

### gis:width-of

```
gis:width-of RasterDataset
```

Retourne le nombre de colonnes dans l'ensemble de données. Notez que ce nombre représente le nombre de cellules de gauche à droite et non la largeur de l'ensemble de données dans l'espace GIS.

### gis:height-of

```
gis:height-of RasterDataset
```

Retourne le nombre de lignes dans l'ensemble de données. Notez que ce nombre représente le nombre de cellules de haut en bas et non la hauteur de l'ensemble de données dans l'espace GIS.

### gis:raster-value

```
gis:raster-value RasterDataset coord-x coord-y
```

Retourne la valeur d'une cellule de l'ensemble de données *RasterDataset*, cellule spécifiée par ses coordonnées *coord-x* et *coord-y*. Les coordonnées des cellules correspondent à leur numérotation dans le plan. Cette numérotation va de gauche à droite et de haut en bas, en commençant à zéro. Ainsi, la cellule placée dans le coin supérieur gauche a pour coordonnées (0,0) et la cellule placée dans le coin inférieur droit a pour coordonnées (`gis:width-of dataset` - 1, `gis:height-of dataset` - 1).

### gis:set-raster-value

```
gis:set-raster-value RasterDataset coord-x coord-y valeur
```

Donne la nouvelle valeur *valeur* à la cellule de coordonnées (*coord-x*, *coord-y*) de l'ensemble de données *RasterDataset*. Les coordonnées des cellules correspondent à leur numérotation dans le plan. Cette numérotation va de gauche à droite et de haut en bas, en commençant à zéro. Ainsi, la cellule placée dans le coin supérieur gauche a pour coordonnées (0,0) et la cellule placée dans le coin inférieur droit a pour coordonnées (`gis:width-of dataset` - 1, `gis:height-of dataset` - 1).

### gis:minimum-of

```
gis:minimum-of RasterDataset
```

Retourne la valeur la plus basse de l'ensemble de données *RasterDataset* spécifié.

### gis:maximum-of

```
gis:maximum-of RasterDataset
```

Retourne la valeur la plus haute de l'ensemble de données *RasterDataset* spécifié.

### gis:sampling-method-of

```
gis:sampling-method-of RasterDataset
```

Retourne la méthode d'échantillonnage (*sampling*) utilisée pour calculer la valeur de l'ensemble de données *RasterDataset* en un seul point ou pour une surface plus petite que celle d'une cellule de la trame.

L'échantillonnage est effectué par les primitives `raster-sample`, `resample`, `convolve` et `apply-raster` de l'extension GIS. La méthode d'échantillonnage est l'une des suivantes :

- `" NEAREST_NEIGHBOR "` : utilise la valeur de la cellule voisine la plus proche du lieu d'échantillonnage.
- `" BILINEAR "` : les valeurs des quatre cellules les plus proches sont échantillonnées par pondération linéaire en fonction de leur distance au lieu d'échantillonnage.
- `" BICUBIC "` : les valeurs des seize cellules sont échantillonnées, et leurs valeurs sont combinées par importance en accord avec la méthode « *piecewise cubic polynomial* » recommandée par Rifman (voir *Digital Image Warping*, George Wolberg, 1990, pp 129-131, IEEE Computer Society Press).
- `" BICUBIC_2 "` : la valeur est échantillonnée en utilisant la même procédure et la même méthode polynomiale qu'avec `BICUBIC`, mais en utilisant un coefficient différent. Cette méthode peut produire des résultats quelque peu plus exacts que `BICUBIC`, mais le résultat dépend des données.

Pour plus d'information sur les méthodes d'échantillonnage et sur l'échantillonnage des trames en général, voir l'article du Wikipedia Image Scaling.

## gis:set-sampling-method

`gis:set-sampling-method` *`RasterDataset sampling-method`*

Fixe la méthode d'échantillonnage *`sampling-method`* utilisée par l'ensemble de données *`RasterDataset`* pour un point ou pour une surface plus petite que celle d'une cellule de la trame. L'échantillonnage est fait par les primitives `raster-sample`, `resample`, `convolve`, et `apply-raster` de l'extension GIS. La méthode d'échantillonnage doit être l'une des quatre suivantes :

- `" NEAREST_NEIGHBOR "`
- `" BILINEAR "`
- `" BICUBIC "`
- `" BICUBIC_2 "`

Voir la primitive `sampling-method-of` ci-dessus pour une description plus détaillée de chaque méthode d'échantillonnage.

## gis:raster-sample

`gis:raster-sample` *`RasterDataset sample-location`*

Retourne la valeur de la trame *`RasterDataset`* donnée à l'endroit *`sample-location`* donné. La localisation de l'échantillonnage peut être spécifiée par l'une des méthodes suivantes :

- Une liste de deux éléments (`[ xcor ycor ]`), du type de celle retourné par la procédure `location-of`, choisis pour représenter un point dans l'espace NetLogo. L'ensemble de données *`RasterDataset`* est échantillonné pour le point situé à cet endroit.
- Une liste de quatre éléments choisis pour représenter une enveloppe dans l'espace GIS, du type de celle retournée par la primitive `envelope-of`. L'ensemble de données *`RasterDataset`* est échantillonné sur la surface de cette enveloppe.
- Un patch, dans ce cas, l'ensemble de données *`RasterDataset`* est échantillonné sur la surface du patch.
- Une tortue, dans ce cas, l'ensemble de données *`RasterDataset`* est échantillonné pour le point représentant l'emplacement de la tortue.
- Un point (Vertex), dans ce cas, l'ensemble de données *`RasterDataset`* est échantillonné à l'emplacement de ce point.

Si l'emplacement spécifié se trouve en dehors de la surface couverte par l'ensemble de données, cette primitive retourne la valeur spéciale représentant "not a number" (pas un nombre) qui est affichée par NetLogo sous la forme "NaN". L'utilisation de la valeur spéciale Not a Number en tant qu'argument pour une primitive qui demande une valeur numérique peut générer une erreur, mais vous pouvez tester la valeur retournée par cette primitive pour écarter les valeurs Not a Number. Une valeur qui n'est pas un nombre ne sera ni plus petite que ni plus grande qu'une valeur numérique, c'est pourquoi vous pouvez détecter les valeurs Not a Number au moyen du code suivant :

```
let valeur gis:raster-sample dataset turtle 0
; met en bleu si valeur est un nombre, en rouge si valeur est "not a number"
ifelse (value <= 0) or (value >= 0)
[ set color blue ]
[ set color red ]
```

Si l'emplacement spécifié est un point, l'échantillonnage est toujours calculé en utilisant la méthode spécifiée par `set-sampling-method`. Si l'emplacement spécifié est une surface (par exemple une enveloppe ou un patch), l'échantillonnage est calculé en prenant la moyenne de toutes les cellules de la trame couvertes par la surface spécifiée.

### gis:raster-world-envelope

`gis:raster-world-envelope` *`RasterDataset x y`*

Retourne l'enveloppe GIS nécessaire pour faire correspondre les frontières des patches NetLogo avec les cellules de l'ensemble de données de la trame (raster). Cette enveloppe peut ensuite être utilisée comme argument pour la primitive `set-transformation-ds`.

Il est possible qu'il y ait plus de cellules dans la trame que de patches dans le monde NetLogo. Dans ce cas, vous devrez sélectionner un sous-ensemble de cellules dans l'ensemble de données de trame en spécifiant quelle cellule de la trame vous voulez faire correspondre avec le coin supérieur gauche du monde NetLogo. Les cellules sont numérotées de gauche à droite et de haut en bas, en commençant par 0. Ainsi, la cellule située tout en haut à gauche est le cellule `(0,0)` et la cellule située en bas à droite est la cellule `(gis:width-of dataset - 1, gis:height-of dataset - 1)`.

### gis:create-raster

`gis:create-raster` *`width height envelope`*

Crée et retourne un nouvel ensemble de données tramées comprenant *`width`* colonnes et *`height`* lignes qui couvrent l'enveloppe *`envelope`* donnée.

### gis:resample

`gis:resample` *`RasterDataset envelope width height`*

Retourne un nouvel ensemble de données comprenant *`RasterDataset`* ré-échantillonné de manière à couvrir l'enveloppe *`envelope`* donnée et qui contient le nombre de colonnes *`width`* et de lignes *`height`* données. Si les cellules de la nouvelle trame sont plus petites que les cellules existantes, elles seront ré-échantillonnées par la méthode spécifiée par `set-sampling-method`. Si les nouvelles cellules sont plus grandes que les cellules originales, elles sont ré-échantillonnées en utilisant la méthode `"NEAREST_NEIGHBOR"`.

### gis:convolve

`gis:convolve` *`RasterDataset kernel-rows kernel-columns kernel key-column key-row`*

Retourne une nouvelle trame dont les données proviennent de la trame *`RasterDataset`* donnée convolutée autour du noyau *`kernel`* donné, noyau formée de *`kernel-rows`* lignes et de *`kernel-columns`* colonnes.

La convolution est une opération mathématique qui calcule la valeur de chaque cellule en multipliant les éléments d'un noyau par les valeurs des cellules entourant une cellule source particulière. Un noyau est une matrice de valeurs comprenant une valeur particulière définie comme « élément clé », la valeur centrée sur la cellule source correspondant à la cellule de destination dont la valeur est en train d'être calculée.

Les valeurs de la matrice du noyau sont données sous forme de liste qui énumère les éléments de la matrice de gauche à droite et de haut en bas. Les éléments d'une matrice 3-3 seront listés de la manière suivante :

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

L'élément clé est spécifié par le numéro de la colonne *`key-column`* et celui de la ligne *`key-rown`* de la matrice. Les colonnes sont numérotées de gauche à droite, à partir de zéro. Les lignes sont numérotées de haut en bas, à partir de zéro. Par exemple, le noyau pour l'opérateur horizontal Sobel, qui ressemble à :

| | | |
|---|---|---|
| 1 | 0 | -1 |
| 2 | 0<br>(key) | -2 |
| 1 | 0 | -1 |

doit être spécifié de la manière suivante :

```
let horizontal-gradient gis:convolve dataset 3 3 [ 1 0 -1 2 0 -2 1 0 -1 ] 1 1
```

### gis:apply-raster

`gis:apply-raster` *`RasterDataset patch-variable`*

Copie les valeurs de l'ensemble de données tramées *`RasterDataset`* dans la variable patch *`patch-variable`* donnée, ré-échantillonnant si nécessaire la trame de manière à ce que les frontières de sa cellule correspondent avec les frontières du patch NetLogo. Le ré-échantillonnage est effectué comme si l'on utilisait `resample` plutôt que `raster-sample`, ceci pour des raisons d'efficacité. Cependant, les patches non couverts par la trame reçoivent la valeur `not a number` de la même manière que `raster-sample` le fait pour des emplacements situés hors de la trame.

## Primitives de dessin

### gis:drawing-color

`gis:drawing-color`

Retourne la couleur utilisée par l'extension GIS pour dessiner les objets vectoriels dans la couche de dessin de NetLogo. Les couleurs peuvent être représentées soit en tant que couleurs NetLogo (un seul nombre entre 0 et 140) soit en tant que couleurs RGB (une liste de trois nombres). Pour les détails, voir la section Couleurs du Guide de la programmation.

### gis:set-drawing-color

```
gis:set-drawing-color color
```

Spécifie la couleur à utiliser par l'extension GIS pour dessiner les objets vectoriels dans la couche de dessin de NetLogo. L'entrée *color* peut représenter soit une couleur NetLogo (un seul nombre entre 0 et 140) soit une couleur RGB (une liste de trois nombres). Pour les détails, voir la section Couleurs du Guide de la programmation.

### gis:draw

```
gis:draw vector-data line-thickness
```

Dessine l'objet vectoriel *vector-data* sur la couche de dessin de NetLogo en utilisant la couleur de dessin GIS courante, avec l'épaisseur de ligne *line-thickness* donnée. L'objet vectoriel peut être soit un ensemble de données *VectorDataset* complet, soit un objet vectoriel (*VectorFeature*) unique. Cette primitive ne dessine que les contours des objets polygones, et pour les objets points, elle remplit un cercle dont le rayon est égal à l'épaisseur de la ligne *line-thickness*.

### gis:fill

```
gis:fill vector-data line-thickness
```

Remplit (peint) l'objet vectoriel *vector-data* sur la couche de dessin de NetLogo en utilisant la couleur de dessin GIS courante et l'épaisseur de ligne donnée *line-thickness* pour faire de tour de l'objet. L'objet vectoriel peut être soit un ensemble de données *VectorDataset* complet, soit un objet vectoriel (*VectorFeature*) unique. Pour les objets points, elle remplit un cercle dont le rayon est égal à l'épaisseur de la ligne *line-thickness*.

### gis:paint

```
gis:paint RasterDataset
```

Peint la trame *RasterDataset* donnée sur la couche de dessin de NetLogo. NetLogo utilise le blanc pour représenter la valeur la plus élevée de l'ensemble de données, le noir pour représenter la valeur la plus basse, alors que les valeurs intermédiaires sont représentées par des nuances de gris réparties régulièrement entre le blanc et le noir.

### gis:import-wms-drawing

```
gis:import-wms-drawing server-url spatial-reference layers transparency
```

Importe une image sur la couche de dessin de NetLogo en utilisant le protocole Web Mapping Service (WMS) tel que défini par le consortium Open Geospatial Consortium.

Les entrées *spatial reference* et *layers* doivent l'être sous forme de chaînes de caractères. L'entrée *spatial reference* correspond au paramètre SRS de la requête GetMap telle que définie dans la section 7.2.3.5 de la version 1.1.1 du standard WMS.

Vous pouvez trouver la liste des codes de référence spatiale et des noms de layers valides en examinant la réponse à une requête GetCapabilities faite au serveur WMS. Consultez le standard y relatif pour les instructions décrivant comment émettre une requête GetCapabilities à l'intention du serveur et comment interpréter sa réponse.

L'entrée *transparency* spécifie le niveau de transparence que la nouvelle image aura sur la couche de dessin. Les valeurs acceptables vont de 0 (complètement opaque) à 255 (complètement transparent).

## Remerciements

L'extension GIS utilise les bibliothèques logicielles open-source suivantes :

- La suite Java Topology Suite sous licence LGPL.
- La bibliothèque JScience, sous sa propre licence open-source license.
- La bibliothèque Java Advanced Imaging sous sa propre licence open-source license.
- Plusieurs bibliothèques issue du projet Apache Commons, y compris Codec, HttpClient et Logging sous licence Apache open-source license.

Cette extension contient aussi des éléments puisés dans My World GIS.

Cette documentation et les exemples de modèles NetLogo sont mis dans le domaine public, alors que l'extension GIS elle-même est disponible sous licence open-source. Voir le fichier `LICENSE.txt` dans le répertoire de l'extension GIS pour plus de détails.

# Chapitre 29

# FAQ (Foire Aux Questions)

Le retour d'informations et les questions des utilisateurs nous sont d'une aide précieuse pour poursuivre le développement et l'amélioration de NetLogo. Nous aimerions connaître vos avis. Envoyez s'il-vous-plaît vos commentaires, suggestions et questions à feedback@ccl.northwestern.edu et vos rapports de bugs à bugs@ccl.northwestern.edu.

## Les questions

### Généralités



### Téléchargement

## Applets



## Fonctionnement



## Utilisation



## Programmation

### BehaviorSpace



### Extensions



## Généralités

### Pourquoi s'appelle-t-il NetLogo?

La partie "Logo" du nom parce que NetLogo est un dialecte du langage Logo.

"Net" (réseau) est là pour évoquer la nature décentralisée mais inter-connectée des phénomènes que vous pouvez modéliser avec NetLogo, y compris des phénomènes de réseau. Il fait aussi référence à HubNet, l'environnement de simulation participative multi-utilisateurs inclus dans NetLogo.

### Comment faire référence à NetLogo dans les publications à caractère académique?

*Pour **NetLogo** lui-même* : Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

*Pour **HubNet*** : Wilensky, U. & Stroup, W., 1999. HubNet. http://ccl.northwestern.edu/netlogo/hubnet.html. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

### Comment citer un modèle de la Bibliothèque des modèles dans une publication?

La référence correcte est incluse dans la section CREDITS AND REFERENCES de la description de chaque modèle dans le panneau "Information"

### Où et quand NetLogo a-t-il été créé?

NetLogo a d'abord été créé en 1999 par Uri Wilensky au Center for Connected Learning and Computer-Based Modeling, puis à la Tufts University dans la région de Boston. NetLogo a grandi à partir de StarLogoT, qui a été développé en 1997 par Wilensky. En 2000, le CCL a pris ses quartiers à la Northwestern University, dans la région de Chicago. NetLogo 1.0 est apparu en 2002, 2.0 en 2003, 3.0 en 2005 et 4.0 en 2007.

## Dans quel langage de programmation a été écrit NetLogo?

NetLogo est entièrement écrit en Java (version 1.4.1).

## Quel est la différence entre StarLogo, MacStarLogo, StarLogoT et NetLogo?

Le StarLogo original a été développé au Media Lab du MIT en 1989-1990 et tournait sur un super-ordinateur massivement parallèle appelé Connection Machine. Quelques années plus tard (1994), une version parallèle simulée a été développée pour l'ordinateur Macintosh. Cette version est devenue pas la suite le MacStarLogo. StarLogoT (1997), développé au Center for Connected Learning and Computer-Based Modeling (CCL), est avant tout une version améliorée et étendue de MacStarLogo dotée de fonctionnalités et de possibilités supplémentaires.

Depuis lors, deux Logo multi-plateformes et multi-agents écrits en Java ont été développés : NetLogo (au CCL) et une version de StarLogo basée sur Java (au MIT).

Le langage et l'environnement de NetLogo diffèrent par bien des aspects du StarLogo du MIT. Les deux langages ont été inspirés par le StarLogo original, mais ont été redéfinis de manières différentes. La conception de NetLogo a été guidée par la nécessité de réviser et d'étendre le langage de manière à le rendre plus puissant et plus facile à utiliser, mais aussi par la nécessité de supporter l'architecture HubNet. NetLogo incorpore presque toutes les fonctionnalités étendues de notre ancien StarLogoT aussi bien qu'une grande quantité de nouvelles fonctionnalités.

## Quelle est la licence d'utilisation de NetLogo? Son utilisation, sa redistribution, etc., est-elle soumise à quelque restriction légale?

La licence est donnée dans la section "Copyright" du Manuel de l'utilisateur NetLogo, aussi bien que dans la fenêtre "About NetLogo" de l'application et le fichier `README` qui accompagne le téléchargement.

Un résumé rapide de la licence est que son utilisation n'est soumise à aucune restriction, mais qu'il y a quelques restrictions quand à sa redistribution et/ou modification (à moins que vous ne preniez contact avec Uri Wilensky pour en régler les différents termes).

Nous sommes en cours de réévaluer le contenu de la licence en réponse aux demandes des utilisateurs. Dans le futur, nous avons l'intention de sortir une licence révisée.

## Est-ce que le code source de NetLogo est disponible?

Pour le moment, non. Nous songeons à une future publication du code source sous une licence de type open source.

Toutefois, en attendant, NetLogo n'est pas un logiciel fermé. Nous offrons un API pour contrôler NetLogo à partir d'un code Java externe et un autre API pour permettre aux utilisateurs d'écrire de nouvelles commandes et reporters en Java. (Voir les chapitres "Controlling" et "Extension" dans le Manuel de l'utilisateur.) Nous encourageons les utilisateurs à écrire des extension pour NetLogo et à les partager avec la communauté des utilisateurs de NetLogo.

## Organisez-vous des ateliers ou autres possibilités de formations pour NetLogo?

Nous organisons des ateliers de temps en temps. Si un atelier est programmé, nous l'annoncerons dans la page d'accueil de NetLogo et dans le groupe des utilisateurs de NetLogo. Si vous êtes intéressés par ce genre d'opportunités, contactez-nous à l'adresse feedback@ccl.northwestern.edu.

## Existe-t-il des livres consacrés à NetLogo?

Nous, au CCL, avons il y a déjà quelque temps espéré écrire plusieurs livres. Ils seraient destinés à différents publics tels que : écoles moyennes et moyennes supérieures, cours de modélisation ou étude de phénomènes complexes du premier cycle ou encore un guide pratique pour adultes intéressés par le sujet.

Malheureusement, nous n'avons pas encore été capables de trouver le temps de réaliser ces projets. Si quelqu'un de la communauté des utilisateurs a envie de participer à une telle aventure, qu'il nous le fasse savoir. Nous l'accueillerons bien volontiers.

## Est que NetLogo est disponible dans d'autres langues ?

Jusqu'à maintenant, NetLogo n'existe qu'en anglais.

Nous envisageons pour l'avenir d'offrir aux utilisateurs la possibilité de produire leurs propres packs de langues pour NetLogo et de les mettre ensuite à disposition des autres utilisateurs. Pour rendre cela possible, nous devons séparer tout le texte anglais du reste du code source de manière à ce qu'il soit éditable séparément. Mais nous ne savons pas quand nous le ferons.

## NetLogo, langage compilé ou interprété?

Réponse courte : partiellement compilé; nous travaillons en direction d'un compilateur complet.

Réponse longue : NetLogo possède un compilateur qui génère du byte code Java à partir du code tapé dans le panneau "Procedures". Toutefois, ce compilateur ne supporte pas encore la totalité du langage, ce qui fait que certaines parties du code écrit par l'utilisateur sont interprétées. Nous travaillons à l'extension du compilateur afin qu'il puisse compiler la totalité du langage. Notez que notre compilateur génère du byte code Java, et que les machines virtuelles Java ont des compilateurs "just-in-time" qui a leur tour compilent le byte code Java en code natif, ce qui fait que le code de l'utilisateur est finalement traduit en code natif.

## Quelqu'un a-t-il programmé un modèle au sujet de <x>?

La meilleure place pour poser cette question est le NetLogo Users Group. Vous devriez aussi allez voir la section "Community Models" de notre page web Models Library.

## Est-ce que les simulations des modèles NetLogo sont scientifiquement reproductibles?

Oui. Les algorithmes activant les agents de NetLogo sont déterministes et NetLogo utilise toujours la bibliothèque Java "strict math" qui donne des résultats identiques au bit près quel que soit le système d'exploitation utilisé. Mais gardez quand même les principes de précautions suivants à l'esprit :

- ✔ Si votre modèle utilise des nombres aléatoires, alors, pour obtenir un comportement reproductible, vous devez utiliser la commande `random-seed` pour définir par avance l'amorce du générateur de nombres aléatoires, de manière à ce que votre modèle reçoive à chaque fois la même séquence de nombres aléatoires. Souvenez-vous que les agents des ensembles d'agents sont toujours rangés dans un ordre aléatoire, il s'ensuit que tout ce que vous faites avec les ensembles d'agents utilise les nombres aléatoires.
- ✔ Si le modèle utilise les commandes `every` ou `wait` de manière telle qu'elles influencent le déroulement de la simulation, vous pourrez obtenir des résultats différents sur des ordinateurs différents, voire sur le même ordinateur puisque la simulation peut se dérouler à des vitesses différentes. (De tels modèles sont plutôt rares. Ces deux commandes sont souvent utilisées, mais d'une manière telle qu'elles n'affectent pas le déroulement de la simulation.)

- ✔ De manière a obtenir des simulations parfaitement identiques, vous devez aussi utiliser exactement la même version de NetLogo. Les détails du mécanisme de gestion des actions des agents et de celui du générateur de nombres aléatoires peuvent changer d'une version à l'autre de NetLogo et d'autres modifications (corrections de bugs dans la machine virtuelle, changements dans le langage, etc.) peuvent aussi influencer le comportement de votre modèle. (Encore une fois, ce n'est pas toujours le cas.)
- ✔ Nous avons déployé tous les efforts possibles pour faire en sorte que le fonctionnement des modèles soit totalement reproductible, mais bien entendu, ceci ne pourra jamais être une garantie en béton armé, dû non seulement aux possibles erreurs aléatoires de fonctionnement du matériel, mais aussi à la possibilité d'erreurs humaines dans la conception de : votre modèle, NetLogo, votre machine Java virtuelle, votre matériel, et ainsi de suite.

### NetLogo et NetLogo 3D seront-ils toujours des programmes distincts?

Non. Cette séparation est temporaire. Dans le futur, une version unifiée de NetLogo supportera aussi bien la modélisation en 2D que la modélisation en 3D. Nous voulons nous assurer de concevoir le support du monde 3D de manière telle qu'il ne se mette pas en travers de votre chemin quand vous construisez des modèles 2D.

Les modèles construits avec les versions préparatoires de NetLogo 3D demanderons certainement des modifications pour pouvoir fonctionner avec la future version unifiée.

### Les anciennes versions de NetLogo sont-elles toujours supportées

Oui. Nous supportons toujours NetLogo 1.3.1 (pour les utilisateurs de Mac OS 8 et 9 et de Windows 95), NetLogo 2.0.2, NetLogo 2.1, NetLogo 3.0.2 et NetLogo 3.1.5, et nous continuerons de les supporter tant que les gens les utiliserons.

Il pourra même y avoir des versions partielles de la série 3.1.x si des utilisateurs signalent des bugs ou des incompatibilités entre versions qui nécessitent d'être corrigés.

Pour éviter de noyer les utilisateurs sous un flot d'option, la page de téléchargement du site de NetLogo n'offre qu'une sélection limitée des anciennes versions (en fait, les versions citées ci-dessus), mais si vous avez besoin d'une version spécifique absente de la liste, contactez-nous et nous serons heureux de pouvoir vous la procurer.

## Téléchargement

### Le téléchargement ne fonctionne pas pour moi. Comment avoir un lien direct avec le logiciel?

Écrivez-nous s'il-vous-plaît à l'adresse bugs@ccl.northwestern.edu et soit nous corrigerons le formulaire de téléchargement, soit nous vous fournirons une méthode de remplacement pour télécharger le programme.

### Le téléchargement de NetLogo est trop lent. Peut-on l'obtenir par une autre voie, sur un CD par exemple?

Pas pour le moment. Si cela vous pose un problème, contactez-nous à feedback@ccl.northwestern.edu.

### J'ai téléchargé et installé NetLogo, mais la Bibliothèque des modèle ne comporte que peu, voire pas de modèles. Comment y remédier?

Jusqu'ici, les utilisateurs qui nous ont fait part de ce problème utilisaient tous l'option de téléchargement "without VM" pour Windows. Si c'est aussi votre cas, dé-installez NetLogo et essayez avec le téléchargement "with VM".

Même si le téléchargement "with VM" résout votre problème, contactez-nous à l'adresse bugs@ccl.northwestern.edu afin que nous puissions en savoir plus sur les détails de votre installation. Nous aimerions corriger ce défaut pour les versions futures, mais pour y arriver, nous avons besoin de l'aide des utilisateurs.

### Puis-je avoir plusieurs versions de NetLogo installées en même temps?

Oui. Quand vous installez NetLogo, le répertoire qui est créé et qui contient NetLogo a le numéro de la version dans son nom, ce qui fait que plusieurs versions peuvent coexister.

Avec les systèmes Windows, quelle que soit le numéro de la version que vous avez installée en dernier, c'est cette version qui sera utilisée quand vous double-cliquerez un modèle dans l'Explorateur de fenêtres. Avec les Macs, vous pouvez contrôler quelle version s'ouvre via la fenêtre "Info sur" du Finder.

### Je suis sur un système UNIX et ne peux dé-archiver (untar) le téléchargement. Pourquoi?

Certains des fichiers de l'archive ont de très longs noms de chemin, trop longs pour le format standard. Vous devez utiliser la version GNU de l'utilitaire tar (ou un autre programme qui comprend les extensions du tar de GNU). Sur certains systèmes, la version GNU de tar est disponible sous le nom de "gnutar". Vous pouvez savoir si vous utilisez déjà la version GNU en tapant `tar --version` et en contrôlant si la sortie dit "`tar (GNU tar)`".

### Comment installer NetLogo

Cela dépend de la plateforme que vous utilisez.

- **Linux** : Dé-archivez (*untar*) NetLogo à l'emplacement approprié.
- **Mac** : Copiez le répertoire NetLogo directement de l'image disque dans le répertoire "Applications".
- **Windows** : Installez NetLogo puis copiez le répertoire résultant sur les autres machines. Malheureusement, NetLogo n'apparaîtra pas dans le menu de démarrage et les fichiers `.nlogo` ne démarrerons pas automatiquement NetLogo quand ils seront double-cliqués. Toutefois, Anders Martinusen écrit :

```
Also found that is possible to export a few keys from the windows
registry into a .reg file. This file can then be pushed out on the
target machines along with the files. This will solve the issue about
double clicking the .nlogo files. It also gives the possibility to later
on uninstall the program from the Control Panel.

The content of the reg file is:

[HKEY_CLASSES_ROOT\.nlogo]
@="NetLogoModelFile"

[HKEY_CLASSES_ROOT\NetLogoModelFile]
@="NetLogo model file"

[HKEY_CLASSES_ROOT\NetLogoModelFile\DefaultIcon]
@="%TARGET_PATH%\\Model icon.ico,0"
```

```
[HKEY_CLASSES_ROOT\NetLogoModelFile\shell\open\command]
@="\"%TARGET_PATH%\\NetLogo 4.0.2.exe\" --launch \"%1\""

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\NetLogo
4.0.2]"DisplayName"="NetLogo 4.0.2""UninstallString"="\"%TARGET_PATH
%\\UninstallerData\\Uninstall
NetLogo.exe\""
```

### J'ai téléchargé l'archive Windows without-Java. Pourquoi NetLogo ne veut-il pas démarrer?

Nous recommandons le téléchargement de la version Widows normale, qui inclut le paquet Java. Si vous avez des problèmes avec l'archive sans Java, consultez le chapitre 4 Configuration minimale pour plus de détails.

## Les applets

### J'ai essayé de faire tourner l'une des applets de votre site, mais ça ne marche pas. Que faire?

Les versions actuelles de NetLogo demandent que votre navigateur supporte Java 1.4.1 ou supérieur. Pour plus de détails sur les exigences en matière de Java, voir le Guide Applet.

Certaines applets NetLogo peuvent demander plus de mémoire que le navigateur ne met habituellement à disposition. Voir la section mémoire du guide Applet pour voir comment augmenter la mémoire allouée à l'applet.

### Puis-je diffuser mon modèle sous forme d'applet tout en gardant mon code secret?

Non. Pour être en mesure de fonctionner, l'applet doit pouvoir aussi accéder au fichier du modèle.

Quand vous utilisez la commande "Save as applet" du menu "File", la page HTML générée contient un lien vers le fichier, lien que l'utilisateur peut cliquer pour télécharger le fichier du modèle. Vous pouvez supprimer ce lien si vous le désirez. Cet effacement rendra l'accès à votre modèle plus difficile, mais pas impossible.

### Est-ce qu'un modèle enregistré sous forme d'applet peut utiliser `import-world`, `file-open` et d'autres commandes qui lisent les fichiers?

Oui, mais seulement pour lire des fichiers qui sont stockés dans le même répertoire de votre serveur web que le fichier HTML et le fichier du modèle. Les applets ne peuvent pas lire de fichiers sur la machine de l'utilisateur, seulement sur le serveur web.

### Quand mon modèle tourne sous forme d'applet, j'obtiens l'erreur suivante : `java.lang.OutOfMemoryError: Java heap space`.

Le plug-in Java n'a pas reçu suffisamment d'espace mémoire pour faire tourner le modèle. Des détails sur la grandeur que peut avoir un modèle en NetLogo peuvent être trouvés ici. Vous devez augmenter la quantité de mémoire mise à disposition du plug-in Java. Vous trouverez les instructions ici.

### Quand j'essaie de charger mon modèle sous forme d'applet, j'obtiens un erreur du type `java.lang.ClassFormatError: Incompatible magic value`.

Si votre serveur web retourne des messages d'erreur personnalisés en réponse à des requêtes pour des pages non existantes, il doit aussi retourner un code de statut "404 Not found", sinon NetLogo pense que les données suivantes sont ce qui a été demandé et tente de les lire en tant que telles. Cela peut arriver même si tous les fichiers requis pour faire tourner l'applet sont présents.

Si vous n'avez pas le contrôle des messages d'erreurs de votre serveur web, vous pouvez utiliser ce qui suit pour contourner cette difficulté :

- ✔ Créez un répertoire nommé `META-INF` dans le même répertoire que les fichiers applet.
- ✔ Créez un sous-répertoire de `META-INF` appelé `services`.
- ✔ Créez un fichier appelé `org.apache.commons.logging.LogFactory` dans le sous-répertoire `services`
- ✔ Mettez la ligne suivante dans le fichier `org.apache.commons.logging.LogFactory` :

```
org.apache.commons.logging.impl.LogFactoryImpl
```

Notez que tous les noms des répertoires et des fichiers sont sensibles à la casse et doivent être écrits exactement comme ci-dessus.

## Fonctionnement

### Puis-je faire fonctionner NetLogo à partir d'un CD?

Oui. NetLogo fonctionne bien sur un système à lecture de fichiers seule.

### Pourquoi NetLogo est-il si lent quand je débranche mon portable Windows?

Quand votre ordinateur portable n'est plus alimenté par le réseau, il commute en mode d'économie d'énergie. Il est donc normal qu'il réduise un petit peu sa vitesse, mais il y a malheureusement un bug dans Java qui, dans ce cas, ralentit fortement les applications Swing, y compris NetLogo.

Une manière de contourner ce problème est de modifier les paramètres d'économies d'énergie de votre ordinateur de manière à ce qu'il n'entre pas en mode économique quand vous le débranchez du réseau. (Mais si vous le faites, votre batterie se déchargera rapidement.)

Une autre manière de contourner ce problème est de faire tourner NetLogo avec une option recommandée par Sun, en éditant le fichier `NetLogo.lax` qui se trouve dans le répertoire NetLogo (dans le répertoire `Program Files` de votre disque dur, à moins que vous n'ayez installé NetLogo à un autre endroit). Éditez cette ligne :

```
lax.nl.java.option.additional=-Djava.ext.dirs= -server -Dsun.java2d.noddraw=true
```

et ajoutez

```
-Dsun.java2d.ddoffscreen=false
```

à la fin de la dernière ligne.

Vous pouvez prendre connaissance des détails concernant ce bug Java et demandez à Sun de le corriger.

### Comment se fait-il que NetLogo ne veuille pas démarrer sur la machine Linux?

Dans l'idéal, toute version de l'exécutable Java 1.4.1 ou supérieure peut faire tourner NetLogo. Toutefois, certaines implémentations de Java ne supportent pas certaines fonctionnalités utilisées par NetLogo, telles que Java2D et Swing. C'est le cas de l'exécutable libgcj de GNU pour Linux qui est pré-installé dans certaines versions de Linux. NetLogo ne fonctionne pas avec cet exécutable.

Nous recommandons les exécutables Java de Sun ou d'IBM quand vous utilisez NetLogo avec Linux.

### Quand j'essaie de démarrer NetLogo sur Windows, j'obtiens l'erreur `"could not create Java virtual machine"`. À l'aide!

Nous ne sommes pas sûrs que ce message n'ait qu'une seule origine, mais une cause possible dont nous avons connaissance est un bug dans le Service Pack 2 de Windows XP (et peut-être dans d'autres versions de Windows telles que Windows Server 2003) qui ne permet pas l'allocation de grandes quantités de mémoire virtuelle contigüe sur certaines machines équipées de ces systèmes d'exploitation.

Une solution possible consiste à utiliser un éditeur de texte pour éditer le fichier NetLogo `4.0.4.lax` (dans le répertoire NetLogo, placé par défaut dans `C:\Program Files`) :

```
#   LAX.NL.JAVA.OPTION.JAVA.HEAP.SIZE.MAX
#   -------------------------------------
#   allow the heap to get huge

lax.nl.java.option.java.heap.size.max=900000000
```

Essayez de remplacer les 900000000 par un nombre plus petit tel que 500000000. Il n'est peut-être pas nécessaire de diminuer aussi fortement la mémoire allouée, vous pouvez faire des tests et la baisser progressivement pour commencer. Cette manoeuvre devrait permettre à NetLogo de démarrer, bien que la plus petite taille de la mémoire allouée puisse limiter les possibilités de faire tourner des modèles ayant un très grand nombre d'agents. (Voir Quelle peut être la grandeur de mon modèle?.)

Un autre remède que vous pourriez essayer, bien que nous ne soyons pas certains du résultat, est de passer à Windows Vista ou au Service Pack 3 de Windows XP.

### Puis-je lancer NetLogo à partir de la ligne de commande, sans la GUI?

Oui. La méthode la plus simple simple consiste à paramétrer votre modèle pour le faire fonctionner en tant qu'expérimentation BehaviorSpace. Aucun programme supplémentaire n'est nécessaire. Voir à ce sujet le chapitre BehaviorSpace du Manuel de l'utilisateur pour plus de détails.

Une autre option consiste à utiliser notre API Controlling, mais il demande un peu de programmation en Java. Voir à ce sujet le chapitre Controlling du Manuel de l'utilisateur pour plus de détails et un exemple de code.

### Est-ce que NetLogo tire parti des systèmes multi-processeurs/coeurs?

Non, pas lorsqu'il n'y a qu'un modèle qui tourne. Le moteur de NetLogo est mono-thread et nous pensons rester dans cette voie. Nous n'avons aucun projet allant dans le sens de rendre possible le partage d'un seul modèle pour lui permettre de tourner sur des processeurs multiples ou sur plusieurs ordinateurs.

Un moyen de tirer avantage des multiprocesseurs ou des processeurs multi-coeurs est de faire tourner plusieurs modèles en parallèle en faisant tourner plusieurs instances de NetLogo en même temps, chacun dans sa propre machine virtuelle Java :

- ✔ Pour faire tourner plus d'une application NetLogo complète en même temps, voyez cette réponse pour les instructions.
- ✔ Vous pouvez aussi faire tourner les modèles à partir de la Ligne de commandes en vous servant de BehaviorSpace ou de l'API Controlling, et démarrer ainsi simultanément autant de processus NetLogo "headless" que vous voulez.

Dans une future version de NetLogo, nous espérons pouvoir améliorer le support des processeurs multiples/multi-coeurs de la manière suivante :

- ✔ Permettre l'ouverture simultanée de plusieurs modèles, chacun tournant dans un *thread* différent, et par conséquent dans un processeur/coeur différent.
- ✔ Modifier BehaviorSpace pour lui permettre, optionnellement, de faire tourner en parallèle plusieurs modèles dans un nombre de *threads* configurable, ainsi les simulations seront réparties entre les processeurs/coeurs disponibles.

## Puis-je distribuer un modèle NetLogo pour le faire tourner sur un groupe d'ordinateurs?

La plus grande partie des réponses données à la question précédente s'appliquent ici. Il n'est pas possible de diviser un unique modèle pour répartir les « morceaux » entre plusieurs ordinateurs, mais vous pouvez faire en sorte que chaque machine du groupe fasse tourner indépendamment un ou plusieurs modèles en utilisant soit BehaviorSpace soit notre API Controlling.

De nombreux utilisateurs ont déjà fait tourner NetLogo sur des groupes d'ordinateurs. Vous pouvez les contacter dans le NetLogo Users Group.

## Je veux essayer HubNet. Puis-je?

Oui. Deux types de HubNet sont disponibles. Avec Computer HubNet, les participants font tourner l'application HubNet Client sur des ordinateurs connectés en réseau « normal ». Avec Calculator HubNet, créé en collaboration avec Texas Instruments, les participants utilisent des calculatrices graphiques Texas Instruments et le Classroom Learning System TI-Navigator.

Voir la partie Calculateur HubNet pour TI-Navigator du chapitre HubNet du Manuel de l'utilisateur pour plus de détails.

Pour des informations supplémentaires concernant HubNet en général, voir le Guide HubNet.

## Existe-t-il un moyen de récupérer le travail perdu quand NetLogo se plante ou se fige?

Oui. NetLogo sauvegarde automatiquement les fichiers quand vous les travaillez. Ces fichiers d'auto-sauvegarde se trouvent dans le répertoire temporaire (temp) spécifique à votre système. Sur la plupart des systèmes de type UNIX (y compris MacOS), il est appelé `temp`. Sur Windows XP, ils peuvent être trouvés dans `C:\Documents and Settings\<user>\Local Settings\Temp` où `<user>` est l'utilisateur courant et dans Windows Vista ils sont dans le répertoire `C:\Users\<user>\AppData\Local\Temp`. Ces fichiers sont nommés selon le format suivant : `autosave_yyyy-MM-dd.HH_mm_ss.nlogo` où `time` et `date` sont l'heure et le jour où le fichier a été ouvert.

# Utilisation

### Pourquoi le modèle semble-il se figer quand je place le curseur vitesse tout à droite?

Le seul moyen dont dispose NetLogo pour faire tourner votre modèle plus rapidement est de mettre à jour la Vue moins fréquemment. Quand vous déplacez le curseur vers la droite, les mises à jour deviennent de moins en moins fréquentes. Comme les mises à jour de la Vue prennent du temps, cela signifie une augmentation de la vitesse.

Cependant, moins de mises à jour signifie aussi que les mises à jour sont de plus en plus espacées dans le temps. Quand il s'écoule plusieurs secondes entre deux mises à jour, on peut avoir l'impression que le modèle s'est figé, mais ce n'est pas le cas. Il tourne à pleine vitesse. Observez le compteur de cycles (tick counter). (Si votre modèle l'utilise. Si ce n'est pas le cas, observez quelque chose d'autre, tel qu'un traceur de courbes.)

Pour se rendre compte de ce qui se passe, il peut être utile de déplacer le curseur vers la droite de manière graduelle plutôt que d'un seul coup. Si vous trouvez que les mises à jour ne sont pas assez fréquentes, ne poussez pas le curseur aussi loin.

### Comment modifier le nombre de patches du monde?

Pressez le bouton "Settings..." de la barre d'outils du panneau "Interface". La boîte de dialogue qui apparaît vous permet de modifier les dimensions du monde.

Une méthode plus rapide consiste à utiliser les trois ensembles de flèches noires placées dans le coin supérieur gauche de la Vue 2D.

### Puis-je utiliser la souris pour « dessiner » dans la Vue?

NetLogo ne possède pas d'outils de dessin préprogrammés pour peindre dans la Vue. Mais avec seulement quelques lignes de code, vous pouvez doter votre modèle de capacités de dessin. Pour voir comment y arriver, étudiez le modèle "Mouse Example" dans la section "Code Examples" de la Bibliothèques des modèles. Les mêmes techniques peuvent être utilisées pour permettre à l'utilisateur d'interagir avec votre modèle en se servant de la souris d'une autre manière.

Une autre possibilité est d'utiliser un modèle spécialement destiné au dessin, tel que le modèle "Drawing Tool" de James Steiner disponible à l'adresse http://ccl.northwestern.edu/netlogo/models/community/.

Une troisième possibilité est de créer une image dans un autre programme puis de l'importer. Voir la réponse à Puis-je importer un dessin dans NetLogo?.

### Quelle grandeur peut avoir mon modèle? Combien de tortues, de patches, de procédures, de boutons, etc., mon modèle peut-il contenir?

Nous avons testé NetLogo avec des modèles qui utilisent des centaines de méga-octets de RAM et ils fonctionnent parfaitement. Mais nous n'avons pas testé de modèles qui utilisent des giga-octets de mémoire. Cela devrait théoriquement marcher, mais vous pouvez rencontrer certaines limites inhérentes à la machine virtuelle Java et/ou au système d'exploitation sous-jacent (soit des limites voulues, soit des bugs).

La machine NetLogo n'a pas de limite de taille fixée. Par défaut, toutefois, NetLogo démarre avec une mémoire réservée maximale de un giga-octets.

Voici comment élever cette limite si nécessaire :

- **Windows** : Éditez cette section du fichier "`NetLogo.lax`" dans le répertoire de NetLogo :

```
# LAX.NL.JAVA.OPTION.JAVA.HEAP.SIZE.MAX
# -------------------------------------
# allow the heap to get huge

lax.nl.java.option.java.heap.size.max=1073741824
```

  Note: cela peut être inopérant avec certains Windows 98 ou Windows ME.

- **Macintosh** : Éditez le fichier `Contents/Info.plist` du paquet de l'application NetLogo. (Vous pouvez atteindre ce fichier par un Ctrl-clic sur l'icône de l'application dans le Finder puis en sélectionnant "Afficher le contenu du paquet" dans le menu qui s'ouvre.) La section concernée est celle-ci, le second nombre est le maximum :

```
<key>VMOptions</key>
<string>-XX:MaxPermSize=128m -Xmx1024M</string>
```

  Vous pouvez aller jusqu'à deux giga-octets par cette méthode. Si votre Mac a un processeur Intel 64-bits et que vous avez Mac OS X 10.5, il peut même être possible d'aller plus haut que les deux giga-octets si vous modifiez d'autres options. Écrivez à feedback@ccl.northwestern.edu pour obtenir de l'aide.

- **Autres systèmes** : Éditez le script `netlogo.sh` (ou une copie), et remplacez le nombre suivant `-Xmx` par la valeur souhaitée.

## Puis-je importer des données GIS dans NetLogo?

Oui. Essayez l'extension GIS (nouvelle depuis NetLogo 4.0.3). Consultez la section GIS de manuel de l'utilisateur.

Une méthode plus simple consiste à utiliser `import-pcolors`, mais cela ne marche que pour l'importation de cartes sous forme d'images bitmap, les autres formats n'étant pas pris en charge.

Il est aussi possible d'écrire du code NetLogo qui lise les données GIS en utilisant nos primitives d'entrées/sorties telles que `file-open`.

## Mon modèle tourne lentement. Comment puis-je l'accélérer?

Voici quelques moyens de le faire tourner plus vite sans modifier la structure du code :

- Utilisez les mise à jour de la Vue basées sur les cycles et non des mises à jour continues.
- Déplacez le curseur vitesse vers la droite. (Cela réduit la fréquence des mises à jour.)
- Si votre modèle utilise toute la mémoire RAM disponible de votre ordinateur, installer de la mémoire supplémentaire peut aider. Si votre disque dur rame (fait beaucoup de bruit) quand votre modèle tourne, vous avez probablement besoin de plus de mémoire RAM.
- Utilisez des tailles de tortues de 1, 1.5 ou 2 car les images de ces tailles sont mises en cache par NetLogo.

Dans bien des cas, toutefois, vous devrez procéder à des modifications du code si vous voulez que votre modèle tourne plus vite. En général, l'opportunité la plus évidente pour accélérer le fonctionnement du modèle provient du fait que vous effectuez trop d'opérations qui concernent toutes les tortues et tous les patches. Souvent, ces trop nombreuses opérations peuvent être réduites en revoyant la structure du modèle de manière à ce qu'il fasse moins de calculs par cycle. Si vous avez besoin d'aide et si vous nous contactez à feed-

back@ccl.northwestern.edu nous pourrons peut-être vous aider si vous nous envoyez votre modèle ou nous expliquez comment il fonctionne. Les membres du NetLogo Users Group sont aussi susceptibles de vous aider à résoudre votre problème.

Notez que l'utilisation de `run` et `runresult` est beaucoup plus lente que de faire tourner le code directement. Vous devriez éviter d'utiliser ces primitives dans du code où les performances sont un point critique

### Puis-je avoir plusieurs modèles ouverts en même temps?

Une instance de NetLogo ne peut avoir qu'un modèle ouvert à la fois. (Nous prévoyons de modifier ceci dans une future version.)

Vous pouvez avoir plusieurs modèles ouverts en ouvrant plusieurs instances de NetLogo. Avec Windows et Linux, il suffit de lancer plusieurs fois application, une fois pour chaque modèle. Avec Mac, vous devez dupliquer l'application dans le Finder puis ouvrir la copie. (La copie ne prend que peu de place sur le disque dur.)

### Puis-je modifier la position d'un sélecteur à la volée?

Pour le moment, non. Nous prévoyons cette possibilité dans une future version de NetLogo.

### Puis-je diviser le code de mon modèle en plusieurs fichiers?

Oui, c'est possible, mais sur une base expérimentale en utilisant le mot-clé `__includes`.

## Programmation

### En quoi le langage NetLogo diffère-t-il des langages StarLogo et StarLogoT? Comment convertir mon modèle StarLogo ou StarLogoT en NetLogo?

Nous n'avons pas de document spécial qui contienne une liste des différences entre ces programmes. Si vous avez des modèles écrits en StarLogo ou en StarLogoT, nous vous suggérons de lire le chapitre Guide de programmation de ce Manuel pour apprendre le langage NetLogo, particulièrement les sections consacrées à `ask` et aux ensembles d'agents. Étudier quelques exemples de code et de modèles dans la bibliothèque des modèles peut aussi vous apporter beaucoup d'informations.

Si vous avez besoin d'aide pour convertir vos modèles StarLogo ou StarLogoT en modèles NetLogo, ne vos gênez pas de demander de l'aide au NetLogo Users Group. Vous pouvez aussi nous posez des questions à l'adresse feedback@ccl.northwestern.edu.

### En quoi le langage NetLogo diffère-t-il des autres Logo?

Il n'existe pas de définition standard du langage Logo. Ce terme recouvre plutôt une famille de langages assez disparate ayant une origine commune, le langage Lisp. Nous croyons toutefois que NetLogo possède suffisamment de caractéristiques communes avec les autres dialectes Logo pour mériter aussi le nom de Logo.

Il n'en reste pas moins que NetLogo diffère par certains aspects de la plupart des autres Logo. Les différences les plus importantes sont énumérées ci-dessous.

**Les différences superficielles** :

- ✔ La priorité des opérateurs mathématiques est différente. Les opérateurs mathématiques infixés (tels que `+`, `*`, etc.) ont une priorité inférieure à celle des reporters portant un nom. Par exemple, dans la plupart des Logo, si on écrit `sin x + 1`, cette expression est interprétée comme `sin (x + 1)`. Mais NetLogo l'interprète de la même manière que le font la plupart des autres langages, qui est aussi la manière dont la même expression est interprétée dans la notation mathématique standard, c'est-à-dire comme `(sin x) + 1`.
- ✔ Les reporters `and` et `or` sont des formes spéciales et non des fonctions ordinaires : elles « court-circuitent », c'est-à-dire qu'elles n'évaluent leur deuxième argument que si nécessaire (donc en fonction du résultat de l'évaluation du premier argument).
- ✔ Les procédures ne peuvent être définies que dans le panneau "Procedures", et non interactivement dans le Centre de commande.
- ✔ Les procédures reporters, c'est-à-dire les procédures qui retournent une valeur (appelées fonctions dans les autres langages de programmation), doivent être définie par `to-report` plutôt que par `to`. La commande servant à exporter (retourner) la valeur calculée par la procédure reporter est `report` et pas `output`.
- ✔ Lors de la définition de la procédure, les arguments (décrivant les valeurs passées en entrée) de la procédure doivent être entourés de crochets, par exemple `to square [ x ]`.
- ✔ Les noms des variables sont toujours utilisés sans ponctuation : il faut toujours écrire `foo`, jamais `:foo` ou `"foo`. (Afin que cela fonctionne, nous avons créé une forme spéciale `set` qui n'évalue pas sa première entrée, plutôt qu'une commande `make` qui demande un argument avec ponctuation.) Il en résulte que les procédures et les variables partagent le même espace de noms.

Les trois dernières différences sont illustrées dans les définitions des procédures suivantes :

| La plupart des Logo | NetLogo |
|---|---|
| `to square :x`<br>`output :x * :x`<br>`end` | `to-report square [x]`<br>`report x * x`<br>`end` |

**Des différences plus « profondes »** :

- ✔ En NetLogo, les portées (domaines de validité) sont définies lexicalement, pas dynamiquement.
- ✔ NetLogo n'a pas de type de données « `word` » (ce que Lisp appelle des « symbols »). Nous pourrions éventuellement l'ajouter un jour, mais comme il n'est que rarement demandé, il est fort probable que sa présence ne soit pas souvent nécessaire pour la modélisation à base d'agents. Nous avons les chaînes. Dans la plupart des situations où les Logo traditionnels utilisent le type word, nous utilisons simplement des chaînes à la place. Par exemple, en Logo, vous pourriez écrire `[see spot run]` (une liste de mots), mais en NetLogo, vous devez écrire `"see spot run"` (une chaîne) ou `["see" "spot" "run"]` (une liste de chaînes).
- ✔ La commande NetLogo `run` fonctionne avec des chaînes, pas avec des listes (puisque nous n'avons pas le type de données « word »), et ne permet pas la définition ou la redéfinition des procédures.
- ✔ Les structures de contrôle telles que `if` et `while` sont des formes spéciales, pas des fonctions ordinaires. Vous ne pouvez pas définir vos propres formes spéciales, ce qui fait que vous ne pouvez pas non plus définir vos propres structures de contrôles. (La commande NetLogo `run` ne vous est ici d'aucune utilité.)
- ✔ Comme avec la plupart des Logo, les fonctions par valeurs ne sont pas supportées. La plupart des Logo offrent des fonctionnalités générales similaires, voire moins performantes, en permettant le

passage et la manipulation de fragments de code sous forme de liste. Les capacités de NetLogo dans ce domaine sont actuellement limitées. Quelques-unes de nos formes spéciales utilisent le style "templates" de l'UCBLogo pour accomplir des choses semblables, par exemple `sort-by [length ?1 < length ?2] string-list`. Dans certaines circonstances, l'utilisation de `run` et de `runresult` est possible, mais contrairement à la plupart des Logo, ils opèrent sur des chaînes, pas sur des listes.

Bien entendu, le langage NetLogo offre encore beaucoup d'autres fonctionnalités que l'on ne trouve pas dans les autres Logo, les plus importantes étant les agents et les ensembles d'agents.

## Comment se fait-il que mon modèle d'une version antérieure ne fonctionne pas correctement?

Consultez la partie Guide de transition du Manuel de l'utilisateur.

## Pourquoi mon code contient-il des caractères bizarres?

Malheureusement, NetLogo peut ne pas bien fonctionner avec des langues (*locale*) autres que `en`. (c'est-à-dire en anglais). `locale` est un réglage qui indique à NetLogo quelle langue est utilisée par votre système, tout comme il lui indique dans quel format (comment) afficher les dates et les nombres. Si vous avez des problèmes avec l'apparition de caractères bizarres dans votre code, utilisez la localisation "`U.S. English`" en modifiant vos options Java.

Les options Java se trouvent à des endroits différents en fonction du système d'exploitation utilisé.

**Windows** : Ouvrez `NetLogo 4.0.3.lax` dans un éditeur de texte tel que Notepad et modifiez cette ligne :

```
lax.nl.java.option.additional=-Djava.ext.dirs= -server -Dsun.java2d.noddraw=true
```

en ajoutant à la fin

```
-Duser.country=US -Duser.language=en
```

**Mac :** Ctrl-clic sur l'icône de l'application NetLogo 4.0.3, sélectionner "Afficher le contenu du paquet" dans le menu qui s'ouvre puis ouvrez "`Contents -> Info.plist`". Éditez l'option `VMOptions` (sous "`Root -> Java`") et ajouter à la fin de la liste

```
-Duser.country=US -Duser.language=en
```

**Autres systèmes** : ouvrez le script "`netlogo.sh`" dans un éditeur de texte et modifiez-le de la manière suivante (à mettre sur une seule ligne) :

```
java -server -Djava.library.path=./lib -Djava.ext.dir= -XX:MaxPermSize=128m
-Xmx1024m -Duser.country=US -Duser.language=en -jar NetLogo.jar"$@"
```

Dans une future version de NetLogo, nous espérons supporter d'autres langues et localisations.

## Comment obtenir l'opposé (le négatif) d'un nombre?

Par l'une des trois méthodes suivantes :

```
(- x)
-1 * x
0 - x
```

Les parenthèses sont nécessaires avec la première méthode.

## Ma tortue avance de 1 mais est toujours sur le même patch. Pourquoi?

Faire avancer la tortue de 1 ne la déplace sur un autre patch que si son cap est un multiple de 90 (autrement dit exactement dans les directions nord, est, sud et ouest).

La raison est que la tortue ne doit pas obligatoirement se trouver au centre d'un patch. Elle peut très bien se tenir près d'un coin. Supposez par exemple que la tortue se trouve près du coin sud-ouest et qu'elle soit orientée face au nord-est. La longueur de la diagonale du patch est de 1.414... (la racine carrée de deux). Il s'ensuit que la commande `fd 1` laissera la tortue près du coin nord-est du même patch.

Si vous ne voulez pas devoir toujours penser à ces problèmes, une des possibilités qui s'offrent à vous est de concevoir votre modèle de manière telle que les tortues restent toujours au centre des patches. Voir la question suivante.

## Comment garder mes tortues au centre des patches?

Une tortue est au centre d'un patch quand ses coordonnées `xcor` et `ycor` sont des nombres entiers.

Vous pouvez déplacer une tortue au centre de son patch courant au moyen de l'une des deux commandes équivalentes suivantes :

```
move-to patch-here
setxy pxcor pycor
```

Mais vous n'aurez jamais besoin de le faire si vous ne permettez jamais à vos tortues de quitter le centre des patches dès le départ.

La commande `sprout` place toujours les tortues qu'elle génère au centre des patches. Par exemple :

```
ask n-of 50 patches [ sprout 1 [ face one-of neighbors4 ] ]
```

Un autre moyen pour forcer une tortue à commencer au centre d'un patch est d'utiliser une commande telle que cette ligne de code tortue qui déplace la tortue au centre d'un patch choisi aléatoirement :

```
move-to one-of patches
```

Une fois qu'une tortue est au centre d'un patch, elle atterrira toujours au centre d'un patch tant que son cap (orientation) sera toujours un multiple de 90 (autrement dit orientée vers le nord, l'est, le sud ou l'ouest) et tant qu'elle avancera ou reculera d'un nombre entier de pas.

Étudiez le code de "Random Grid Walk Example" dans la section "Code Examples" de la Bibliothèque des modèles.

### `patch-ahead 1` retourne le patch sur lequel ma tortue se trouve déjà. Pourquoi?

Reportez-vous deux réponses plus haut. Il s'agit du même problème.

Ce n'est peut-être pas la signification de "*ahead*" à laquelle vous vous attendiez. Avec `patch-ahead`, vous devez spécifier la distance à laquelle il faut regarder vers l'avant. Si vous voulez savoir sur quel patch pénétrera la tortue quand elle avance de manière continue, c'est tout à fait possible. Voir à ce sujet le code de "Next Patch Example" dans la section "Code Examples" de la Bibliothèque des modèles.

### Comment donner la « vue » à ma tortue?

Vous pouvez utiliser la primitive `in-radius` pour permettre à une tortue de voir dans une zone circulaire autour d'elle.

Plusieurs primitives permettent à la tortue de « voir » spécifiquement certaines choses. La primitive `patch-ahead` est utile pour permettre à la tortue de voir ce qui se trouve directement devant elle. Si vous voulez que la tortue puisse voir dans une autre direction que « droit devant elle », essayez les primitives `patch-left-and-ahead` et `patch-right-and-ahead`.

Si vous voulez que la tortue possède un véritable « cône de vision », utilisez la primitive `in-cone`.

Vous pouvez aussi savoir dans quel patch pénétrera la tortue si elle avance de manière continue. Voyez à ce sujet le code de "Next Patch Example" dans la section "Code Examples" de la Bibliothèque des modèles.

### Est-ce que les agents peuvent détecter (sentir) ce qui se trouve dans la couche de dessin?

Non. Si vous voulez faire des marques que les agents puissent « lire », utilisez les couleurs des patches.

### J'obtiens des nombres tels que 0.10000000004 and 0.799999999999 plutôt que 0.1 ou 0.8. Pourquoi?

Consultez la section Mathématiques du Guide de la programmation dans le Manuel de l'utilisateur pour le traitement de ce sujet.

### La documentation dit que `random-float 1` peut retourner 0 mais ne retournera jamais 1. Que faire si je veux aussi recevoir le 1?

Cela n'a réellement aucune importance. Même si 1 était un résultat possible, il ne sortirait qu'approximativement 1 fois sur 2^64, ce qui signifie que vous devriez attendre des centaines d'années avant qu'il apparaissent exactement une fois.

Toutefois, si vous êtes convaincus qu'il doit réellement être possible d'obtenir 1, vous pouvez utiliser `precision` pour arrondir la réponse à un certain nombre de décimales près. Par exemple :

```
print precision (random-float 1) 10
0.2745173723
```

Si vous utilisez cette méthode, notez que 0 et 1 n'ont que la moitié des chances que les autres nombres ont d'apparaître. Pour voir pourquoi il en est ainsi, considérez le cas où vous ne gardez qu'un seul nombre après le point décimal. Les résultats entre 0 et 0.5 sont arrondis à 0 tandis que les résultats entre 0.5 et 1.5 sont arrondis à 1. Ce deuxième ensemble est deux fois aussi grand que le premier. Si vous voulez que 0, 0.1, 0.2, ..., 0.9 aient tous exactement les mêmes chances, vous pouvez écrire `random 11 / 10` à la place, ce qui vous donne 11 réponses ayant les mêmes probabilités d'apparition.

### Comment puis-je empêcher deux tortues d'occuper le même patch?

Voyez le code de "One Turtle Per Patch Example" dans "Code Examples" de la Bibliothèques des modèles.

### Comment puis-je savoir si une tortue est morte?

Quand une tortue meurt, elle se transforme en `nobody`. `nobody` est une valeur spéciale utilisée par NetLogo pour indiquer l'absence d'une tortue ou d'un patch. Ainsi, par exemple :

```
if turtle 0 != nobody [ ... ]
```

Mais vous pouvez aussi utiliser `is-turtle?` :

```
if is-turtle? turtle 0 [ ... ]
```

### Est-ce que NetLogo a des tableaux uni-dimentionnels?

Dans la version actuelle de NetLogo, les listes sont de véritables listes liées plutôt que d'avoir une implémentation sous-jacente sous forme de tableau uni-dimentionnel comme dans les versions précédentes.

Les vrais tableaux sont disponibles dans l'extension Array. Voir la section Arrays & Tables du Manuel de l'utilisateur.

### NetLogo a-t-il des tables de hachage ou des tables associatives?

Oui, mais il faut utiliser l'extension Tables. Voir la section Arrays & Tables du Manuel de l'utilisateur.

### Comment puis-je utiliser différents types de « voisinages » pour les patches (circulaire, Von Neumann, Moore, etc.)?

Les primitives `in-radius` vous donnent accès aux voisinages circulaires de n'importe quel rayon.

La primitive `neighbors` vous donne un voisinage de Moore de rayon 1 et la primitive `neighbors4` un voisinage de Von Neumann de rayon 1.

Pour des voisinages de Moore et de Von Neumann de plus grand rayon, consultez le code de "Moore & Von Neumann Example" dans la section "Code Examples" de la Bibliothèques des modèles.

### Comment convertir un ensemble d'agents en liste, ou l'inverse?

Si vous voulez une liste dans un ordre particulier, utilisez les primitives `sort` ou `sort-by`. La section Listes du Guide de la programmation vous explique comment faire. Voyez aussi l'exemple "Ask Ordering Example" de la section "code Examples" de la Bibliothèque des modèles.

Si vous désirez une liste dans un ordre aléatoire, voici comment :

```
[self] of <agentset>
```

Puisque toutes les opérations sur les ensembles d'agents livrent des ensembles d'agents aléatoirement ordonnées, le résultat de cette commande est une liste rangée aléatoirement.

Pour convertir une liste d'agents en un ensemble d'agents, utilisez les primitives `turtle-set`, `patch-set` ou `link-set`.

### Comment arrêter `foreach`?

Pour arrêter l'exécution de `foreach` vous devez définir une procédure séparée qui ne contient que foreach, par exemple :

```
to test
  foreach [1 2 3] [
    if ? = 2 [ stop ]
    print ?
  ]
end
```

Ce code n'affiche que le nombre 1. La primitive `stop` fait sortir de la procédure courante ce qui fait qu'après le premier `foreach` plus rien ne sera exécuté par cette procédure. (Si la procédure est une procédure reporter, utilisez `report` à la place de `stop`.)

### Parfois, sur mon ordinateur, `distance` et `in-radius` retournent une réponse fausse. Que se passe-t-il?

Il y a un bug dans le serveur VM de Sun (utilisé par NetLogo) qui affecte un petit nombre d'utilisateurs de Windows (en fonction du type de CPU que vous avez dans votre ordinateur).

Si vous n'êtes pas sûrs d'avoir ce problème, faites tourner le modèle "Ants" (dans la Bibliothèques des modèles, section "Sample Models", sous-section "Biology") et pressez plusieurs fois de suite le bouton "SETUP". Le modèle peut s'initialiser un petit nombre de fois correctement, mais si vous avez le bug, l'initialisation échouera (les tas de nourriture ne seront pas placés correctement).

Si vous rencontrez ce problème, vous pouvez le contourner en utilisant à la place le client VM normal ou en utilisant Java 6 (plutôt que Java 5 qui est pré-installé avec NetLogo).

Pour changer le client VM, ouvrez le fichier `NetLogo 4.0.3.lax` (dans le répertoire de l'application NetLogo) et effacer `-server` de cette ligne :

```
lax.nl.java.option.additional=-Djava.ext.dirs= -server -Dsun.java2d.noddraw=true
```

Notez que l'utilisation du VM client ralentira le fonctionnement de vos modèles (mais ils fonctionneront).

L'utilisation de Java 6 vous donne aussi bien le fonctionnement correct que la pleine vitesse. Pour utiliser Java 6, téléchargez et installez le Java 6 JDK de Sun. Notez que vous devez prendre le JDK complet, pas seulement le JRE, car le JRE ne contient pas le serveur VM rapide. Une fois que le Java 6 JDK est installé, allez dans la page web de NetLogo et téléchargez puis installez le NetLogo alternatif qui ne contient pas le paquet Java. Vous trouverez ce téléchargement où la page de téléchargement dit "*Want to run NetLogo using a Java VM you have already installed yourself?*". Durant le processus d'installation de NetLogo, sélectionnez votre installation Java 6 quand on vous le demande.

## BehaviorSpace

### Comment récolter les données tous les n cycles?

Utilisez `repeat` dans les commandes `go` de votre expérimentation, par exemple :

```
repeat 100 [ go ]
```

pour mesurer les paramètre de la simulation tous les 100 pas (cycles ou ticks). Vous obtenez ainsi qu'un pas de l'expérimentation corresponde à 100 pas du modèle.

### Je fais modifier la valeur d'une variable globale que j'ai déclarée dans le panneau "Procedures", mais cela ne fonctionne pas. Pourquoi?

C'est probablement parce que vos commandes "setup" ou votre procédure `setup` utilise la commande `clear-all` qui efface les valeurs spécifiées par BehaviorSpace.

Une solution possible consiste à modifier les commandes "setup" de l'expérimentation de manière à préserver la valeur de la variable, par exemple :

```
let old-var1 var1
setup
set var1 old-var1
```

Ce truc fonctionne parce que `clear-all` n'efface pas les valeurs des variables locales construites avec `let`.

Une autre solution possible est de modifier la procédure `setup` de votre modèle pour utiliser des commandes de nettoyages qui ne nettoient que ce que vous voulez précisément nettoyer.

### Pourquoi Excel coupe-t-il certains de mes résultats?

Dans certaines versions de Excel, les feuilles de calculs ne peuvent avoir plus de 256 colonnes. (Voyez un document Microsoft qui traite du sujet.)

Les solutions possibles incluent :

- ✔ Utiliser une version plus récente d'Excel, telle que Excel 2007.
- ✔ Utiliser un autre programme qu'Excel.
- ✔ Demander à BehaviorSpace de générer ses résultats dans le format tableau plutôt que dans le, ou en plus du, format feuilles de calculs. (Excel peut aussi lire notre format tableau.)
- ✔ Modifier votre expérimentation de manière à ce que les résultats occupent moins de colonnes.

## Extensions

### J'écris une extension. Pourquoi le compilateur me dit-il qu'il ne peut pas trouver `org.nlogo.api`?

Vous devez ajoutez `NetLogo.jar` à votre chemin des classes pour la compilation. Le fichier `NetLogo.jar` fait partie de l'installation de NetLogo.

# Chapitre 30

# Dictionnaire du langage NetLogo

## Les catégories de primitives

Ce qui suit n'est qu'un regroupement approximatif. Souvenez-vous qu'une primitive dédiée à la tortue peut parfois aussi être utilisée par les patches et l'observateur et vice-versa. Pour savoir quel agent (tortues, patches, liens, observateur) peut réellement exécuter une primitive, consultez son entrée dans le dictionnaire.

### Pour les tortues

back (bk) *<breeds>*-at *<breeds>*-here *<breeds>*-on can-move? clear-turtles (ct) create-*<breeds>* create-ordered-*<breeds>* create-ordered-turtles (cro) create-turtles (crt) die distance distancexy downhill downhill4 dx dy face facexy forward (fd) hatch hatch-*<breeds>* hide-turtle (ht) home inspect is-*<breed>*? is-turtle? jump left (lt) move-to myself nobody no-turtles of other patch-ahead patch-at patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead pen-down (pd) pen-erase (pe) pen-up (pu) random-xcor random-ycor right (rt) self set-default-shape __set-line-thickness setxy shapes show-turtle (st) sprout sprout-*<breeds>* stamp stamp-erase subject subtract-headings tie towards towardsxy turtle turtle-set turtles turtles-at turtles-here turtles-on turtles-own untie uphill uphill4

### Pour les patches

clear-patches (cp) diffuse diffuse4 distance distancexy import-pcolors import-pcolors-rgb inspect is-patch? myself neighbors neighbors4 nobody no-patches of other patch patch-at patch-ahead patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead patch-set patches patches-own random-pxcor random-pycor self sprout sprout-*<breeds>* subject

### Pour les ensembles d'agents

all? any? ask ask-concurrent at-points *<breeds>*-at *<breeds>*-here *<breeds>*-on count in-cone in-radius is-agent? is-agentset? is-patch-set? is-turtle-set? link-heading link-length link-set link-shapes max-n-of max-one-of min-n-of min-one-of n-of neighbors neighbors4 no-patches no-turtles of one-of other patch-set patches sort sort-by turtle-set turtles with with-max with-min turtles-at turtles-here turtles-on

### Pour les couleurs

approximate-hsb approximate-rgb base-colors color extract-hsb extract-rgb hsb import-pcolors import-pcolors-rgb pcolor rgb scale-color shade-of? wrap-color

### Pour le contrôle du flux du programme et la logique

and ask ask-concurrent carefully end error-message every foreach if ifelse ifelse-value let loop map not or repeat report run runresult ; (semicolon) set stop startup to to-report wait while with-local-randomness without-interruption xor

### Pour le monde

clear-all (ca) clear-drawing (cd) clear-patches (cp) clear-turtles (ct) display import-drawing import-pcolors import-pcolors-rgb no-display max-pxcor max-pycor min-pxcor min-pycor reset-ticks tick tick-advance ticks world-width world-height

### Pour la perspective (le point-de-vue)

follow follow-me reset-perspective (rp) ride ride-me subject watch watch-me

### Pour le travail en réseau HubNet

hubnet-broadcast hubnet-broadcast-view hubnet-enter-message? hubnet-exit-message? hubnet-fetch-message hubnet-message hubnet-message-source hubnet-message-tag hubnet-message-waiting? hubnet-reset hubnet-send hubnet-send-view hubnet-set-client-interface

### Pour les entrées/sorties

beep clear-output date-and-time export-view export-interface export-output export-plot export-all-plots export-world import-drawing import-pcolors import-pcolors-rgb import-world mouse-down? mouse-inside? mouse-patch mouse-xcor mouse-ycor output-print output-show output-type output-write print read-from-string reset-timer set-current-directory show timer type user-directory user-file user-new-file user-input user-message user-one-of user-yes-or-no? write

### Pour les fichiers

file-at-end? file-close file-close-all file-delete file-exists? file-flush file-open file-print file-read file-read-characters file-read-line file-show file-type file-write user-directory user-file user-new-file

### Pour les listes

but-first but-last empty? filter first foreach fput histogram is-list? item last length list lput map member? modes n-of n-values of position one-of reduce remove remove-duplicates remove-item replace-item reverse sentence shuffle sort sort-by sublist

### Pour les chaînes de caractères

Operators (<, >, =, !=, <=, >=) but-first but-last empty? first is-string? item last length member? position remove remove-item read-from-string replace-item reverse substring word

### Pour les opérations mathématiques

Arithmetic Operators (+, *, -, /, ^, <, >, =, !=, <=, >=) abs acos asin atan ceiling cos e exp floor int is-number? ln log max mean median min mod modes new-seed pi precision random random-exponential random-float random-gamma random-normal random-poisson random-seed remainder round sin sqrt standard-deviation subtract-headings sum tan variance

### Pour les graphiques

autoplot? auto-plot-off auto-plot-on clear-all-plots clear-plot create-temporary-plot-pen export-plot export-all-plots histogram plot plot-name plot-pen-exists? plot-pen-down plot-pen-reset plot-pen-up plot-x-max plot-x-min plot-y-max plot-y-min plotxy set-current-plot set-current-plot-pen set-histogram-num-bars set-plot-pen-color set-plot-pen-interval set-plot-pen-mode set-plot-x-range set-plot-y-range

### Pour les liens

both-ends clear-links create-<breed>-from create-<breeds>-from create-<breed>-to create-<breeds>-to create-<breed>-with create-<breeds>-with create-link-from create-links-from create-link-to create-links-to create-link-with create-links-with die hide-link in-<breed>-neighbor? in-<breed>-neighbors in-<breed>-from in-link-neighbor? in-link-neighbors in-link-from is-directed-link? is-link? is-link-set? is-undirected-link? layout-circle __layout-magspring layout-radial layout-spring layout-tutte <breed>-neighbor? <breed>-neighbors <breed>-with link-heading link-length link-neighbor? link "#links">links links-own <link-breeds>-own link-neighbors link-with my-<breeds> my-in-<breeds> my-in-links my-links my-out-<breeds> my-out-links no-links other-end out-<breed>-neighbor? out-<breed>-neighbors out-<breed>-to out-link-neighbor? out-link-neighbors out-link-to show-link tie untie

### Pour les vidéos

movie-cancel movie-close movie-grab-view movie-grab-interface movie-set-frame-rate movie-start movie-status

### Pour le système

netlogo-applet? netlogo-version

## Les variables préprogrammées…

### des tortues

breed color heading hidden? label label-color pen-mode pen-size shape size who xcor ycor

### des patches

pcolor plabel plabel-color pxcor pycor

### des liens

breed color end1 end2 hidden? label label-color shape thickness tie-mode

### Autres variables

?

## Les mots-clés

breed directed-link-breed end extensions globals __includes patches-own to to-report turtles-own undirected-link-breed

## Les constantes

### Les constantes mathématiques

**e** = 2.718281828459045
**pi** = 3.141592653589793

### Les constantes booléennes

**false**
**true**

### Les constantes pour les couleurs

**black** = 0
**gray** = 5
**white** = 9.9
**red** = 15
**orange** = 25
**brown** = 35
**yellow** = 45
**green** = 55
**lime** = 65
**turquoise** = 75
**cyan** = 85
**sky** = 95
**blue** = 105
**violet** = 115
**magenta** = 125
**pink** = 135

Voir la section Couleurs du Guide de programmation pour de plus amples détails.

# A

## abs

```
abs number
```

Retourne la valeur absolue de *number*.

```
show abs -7
=> 7
show abs 5
=> 5
```

## acos

```
acos number
```

Retourne l'arc cosinus (l'inverse du cosinus) du nombre *number* donné. L'entrée doit appartenir à l'intervalle -1 à 1. Le résultat est en degrés et se trouve dans l'intervalle 0 à 180.

## all?

```
all? agentset [reporter]
```

Retourne `true` (vrai) si tous les agents de l'ensemble *agentset* retournent `true` pour le *reporter* donné. Autrement, retourne `false` (faux) si un contre-exemple est trouvé.

Le *reporter* doit retourner une valeur booléenne (soit `true`, soit `false`) pour chaque agent, sinon un erreur d'exécution est générée.

```
if all? turtles [color = red]
  [ show "every turtle is red!" ]
```

Voir aussi any?

## and

```
condition1 and condition2
```

Retourne `true` (vrai) si *condition1* ET *condition2* sont toutes deux vraies.

Notez que si *condition1* est `false`, alors *condition2* ne sera même pas testée (puisqu'elle ne peut pas influencer le résultat).

```
if (pxcor > 0) and (pycor > 0)
 [ set pcolor blue ]  ;; le quadrant supérieur droit des
                      ;; patches est coloré en bleu
```

## any?

```
any? agentset
```

Retourne `true` (vrai) si l'ensemble d'agents *agentset* n'est pas vide, sinon retourne `false` (faux).

Équivalent à `count ensemble-agents > 0`, mais plus efficace (et, c'est discutable, plus lisible).

```
if any? turtles with [color = red]
  [ show "au moins une tortue est rouge!" ]
```

**Note** : `nobody` n'est pas un ensemble d'agents. Vous ne recevez `nobody` que dans les situations où vous n'attendez qu'un agent unique, pas un ensemble d'agents. Si `any?` reçoit `nobody` comme entrée, une erreur est générée.

Voir aussi `all` et `nobody`

## approximate-hsb

`approximate-hsb` *`hue saturation brightness`*

Retourne un nombre, appartenant à l'intervalle 0 à 140 (140 exclu) de l'espace de couleurs de NetLogo, représentant la couleur donnée dans le système HSB.

Les trois paramètres *`hue saturation brightness`* dont les valeurs peuvent aller de 0 à 255, décrivent la couleur dans le système HSB.

La couleur retournée peut n'être qu'une approximation de la couleur spécifiée en entrée puisque l'espace de couleurs de NetLogo ne comprend pas toutes les couleurs possibles. (Il ne contient qu'un certains nombre de teintes discrètes et, pour chacune de ces teintes, soit la saturation, soit la brillance peut varier, mais pas les deux, autrement dit, l'une des deux vaut toujours 255.)

```
show approximate-hsb 0 0 0
=> 0  ;; black (noir)
show approximate-hsb 127.5 255 255
=> 85.2 ;; cyan
```

Voir aussi `extract-hsb`, `approximate-rgb` et `extract-rgb`

## approximate-rgb

`approximate-rgb`

Retourne un nombre, appartenant à l'intervalle 0 à 140 (140 exclu) de l'espace de couleurs de NetLogo, représentant la couleur donnée dans le système RGB.

Les trois paramètres *`red green blue`* dont les valeurs peuvent aller de 0 à 255, décrivent la couleur dans le système RGB.

La couleur retournée peut n'être qu'une approximation de la couleur spécifiée en entrée puisque l'espace de couleurs de NetLogo ne comprend pas toutes les couleurs possibles. (Voir `approximate-hsb` pour une description de la partie de l'espace de couleurs HSB que NetLogo peut couvrir car c'est difficile de le faire en termes RGB.)

```
show approximate-rgb 0 0 0
=> 0  ;; black (noir)
show approximate-rgb 0 255 255
=> 85.2 ;; cyan
```

Voir aussi `extract-rgb`, `approximate-hsb` et `extract-hsb`

## Les opérateurs arithmétiques (+, *, -, /, ^, <, >, =, !=, <=, >=)

Tous ces opérateurs demandent deux entrées, et tous fonctionnent comme des opérateurs « infixés » (c'est-à-dire qu'ils doivent être placés entre les deux entrées, comme c'est l'usage en notation mathématique traditionnelle). NetLogo gère correctement la priorité des opérations avec les opérateurs infixés.

Les opérateurs fonctionnent de la manière suivante : `+` pour l'addition, `*` pour la multiplication, `-` pour la soustraction, `/` pour la division, `^` pour l'exponentiation, `<` est plus petit que, `>` est plus grand que, `=` est égal à, `!=` n'est pas égal à, `<=` est plus petit ou égal à, `>=` est plus grand ou égal à.

L'opérateur de soustraction (`-`) demande toujours deux entrées à moins qu'on utilise des parenthèses, dans ce cas, il ne demande qu'une entrée. Ainsi, pour prendre l'inverse de `x`, écrivez `(-x)` avec les parenthèses.

Tous les opérateurs de comparaison fonctionnent avec les chaînes. Tous les opérateurs de comparaison fonctionnent aussi avec les agents. Les tortues sont comparées par numéro d'identification. Les patches sont comparés de haut en bas, de gauche à droite, ainsi, le patch 0 10 est plus petit que le patch 0 9 et le patch 9 0 est plus petit que le patch 10 0. Les liens sont ordonnés en fonction du noeud terminal et en cas d'attache, par race. Ainsi, le lien 0 9 est avant le lien 1 10 car l'extrémité 1 est plus petite, et le lien 0 8 est inférieur au lien 0 9. S'il y a plusieurs races de liens, les liens sans races viennent avant les liens appartenant à des races ayant les mêmes points terminaux et les liens avec races seront rangés dans l'ordre où ils ont été déclarés dans le panneau "Procedures".

On peut tester l'égalité ou l'inégalité des ensembles d'agents. Deux ensembles d'agents sont égaux s'ils sont de même type (tortue ou patch) et contiennent les mêmes agents.

Si vous n'êtes pas sûr de la manière dont NetLogo interprétera votre code, ajoutez des parenthèses.

```
show 5 * 6 + 6 / 3
=> 32
show 5 * (6 + 6) / 3
=> 20
```

## asin

```
asin number
```

Retourne l'arc sinus (l'inverse du sinus) du nombre *number* donné. L'entrée doit appartenir à l'intervalle -1 à 1. Le résultat est en degrés et appartient à l'intervalle -90 à 90.

## ask

```
ask agentset [commands]
ask agent [commands]
```

L'agent *agent* ou les agents de l'ensemble d'agents *agenset* exécutent les commandes *commands* données.

```
ask turtles [ fd 1 ]
  ;; toutes les tortues avancent d'un pas
ask patches [ set pcolor red ]
  ;; tous les patches se colorent en rouge
ask turtle 4 [ rt 90 ]
  ;; seule la tortue numéro 4 tourne à droite
```

**Note 1** : seul l'observateur (observer) peut s'adresser à toutes les tortues ou à tous les patches. Ceci afin d'empêcher d'avoir par inadvertance toutes les tortues commander à toutes les tortues ou tous les patches commander à tous les patches, ce qui est une erreur courante si vous ne faites pas attention au type d'agents qui devra exécuter le code que vous écrivez.

**Note 2** : Seuls les agents appartenant à l'ensemble *au moment* où la demande est faite exécutent les commandes.

## ask-concurrent

```
ask-concurrent agentset [commands]
```

Les agents de l'ensemble d'agents *agenset* exécutent les commandes *commands* données en utilisant le mécanisme du « chacun son tour » pour simuler un comportement concurrentiel. Voir la section Ask-Concurrent du Guide de programmation pour les détails de fonctionnement.

**Note** : seuls les agents appartenant à l'ensemble *au moment* où la demande est faite exécutent les commandes

Voir aussi `without-interruption`

## at-points

```
agentset at-points [[x1 y1] [x2 y2] ...]
```

Retourne un sous-ensemble de l'ensemble d'agents *agenset* spécifié ne comprenant que les agents se trouvant sur les patches situés aux distances données, distances mesurées selon les axes x et y par rapport à l'agent appelant. Les distances sont spécifiées dans des listes de deux éléments, où les deux éléments sont les décalages (les distances horizontale et verticale) x et y.

Si la commande vient de l'observateur, les distances sont mesurées par rapport à l'origine, en d'autres mots, les points correspondent aux coordonnées absolues des patches

Si la commande vient d'une tortue, les distances sont mesurées relativement à la position exacte de la tortue et non à partir du centre du patch sur lequel se trouve la tortue.

```
ask turtles at-points [[2 4] [1 2] [10 15]]
 [ fd 1 ]  ;; seules les tortues situées sur les patches aux distances (2,4), (1,2)
           ;; et (10,15) relatives à l'appelant se déplacent
```

## atan

```
atan x y
```

Retourne l'arc tangente en degrés (de 0 à 360), de *x* divisé par *y*.

Quand *y* vaut 0 et si *x* est positif, la primitive retourne 90; si *x* est négatif, elle retourne 270; si *x* vaut zéro, vous obtenez une erreur.

Notez que cette version de la fonction `atan` a été développée pour se conformer à la géométrie du monde NetLogo où une orientation ou un cap (heading) 0 pointe vers le haut (le Nord), 90 vers la droite (l'Est), 180 vers le bas (le Sud) et 270 vers la gauche (l'Ouest). (En géométrie « normale », un angle de 0 pointe à droite, de 90 vers le haut, et ainsi de suite, dans le sens inverse des aiguilles d'une montre tout autour du cercle, et atan est alors défini en conséquence.)

```
show atan 1 -1
=> 135
show atan -1 1
=> 315
```

## autoplot?

```
autoplot?
```

Retourne `true` (vrai) si l'auto-plotting (ajustement des axes du traceur de courbes pour tenir compte de la taille croissante du graphique en train de se dessiner) est activé pour le traceur courant, sinon retourne `false` (faux).

## auto-plot-off
## auto-plot-on

```
auto-plot-off
auto-plot-on
```

Cette paire de commandes est utilisée pour contrôler la fonctionnalité NetLogo d'auto-plotting dans le traceur courant. Quand elle est active (`auto-plot-on`), cette fonctionnalité ajuste automatiquement les axes x et y du traceur de courbes chaque fois que le crayon courant dépasse les limites de la zone de dessin du graphique. Elle est utile quand on veut dessiner toutes les valeurs dans le traceur courant quelles que soient les dimensions du graphiques.

# B

## back
## bk

```
back number
```


La tortue recule du nombre de pas *number*. (Si *number* est négatif, la tortue avance.)

Les tortues qui utilisent cette primitive ne peuvent avancer au maximum que d'une unité par incrément de temps. Ainsi, les commandes `bk 0.5` et `bk 1` nécessitent chacune une unité de temps, mais `bk 3` en demande trois.

Si la tortue ne peut avancer du nombre de pas demandés parce que la topologie courante du monde NetLogo ne le permet pas, la tortue fait autant de pas de une unité que possible puis s'arrête.

Voir aussi `forward`, `jump` et `can-move?`

## base-colors

```
base-colors
```

Retourne une liste contenant les 14 teintes de base de NetLogo.

```
print base-colors
=> [5 15 25 35 45 55 65 75 85 95 105 115 125 135]
ask turtles [ set color one-of base-colors ]
;; chaque tortue est colorée avec l'une des teintes de
;; base choisie au hasard
ask turtles [ set color one-of remove gray base-colors ]
;; chaque tortue est colorée avec l'une des teintes de
;; base choisie au hasard sauf avec le gris
```

## beep

```
beep
```

Émet un bip sonore. Notez que le bip retentit immédiatement, ainsi, plusieurs commandes `beep` successives peuvent produire un son paraissant continu.

Exemple :

```
beep                        ;; émet un seul bip
repeat 3 [ beep ]           ;; émet 3 bips successifs,
                            ;; vous n'entendez donc qu'un seul signal sonore
repeat 3 [ beep wait 0.1 ] ;; produit 3 bips successifs, mais
                            ;; séparés par 1/10 de seconde
```

## both-ends

```
both-ends
```


Retourne un ensemble d'agents formé des deux noeuds connectés par le lien courant.

```
crt 2  ;; créer 2 tortues
ask turtle 0 [ create-link-with turtle 1 ]
;; demander à la tortue 0 de se lier à la tortue 1
ask link 0 1 [ ask both-ends [ set color red ] ]
;; demander aux deux extrémités de se colorer en rouge
```

## breed

`breed`

C'est une variable tortue et lien préprogrammée. Elle contient l'ensemble d'agents formé de toutes les tortues ou de tous les liens de même race que cette tortue ou ce lien. (Pour les tortues ou les liens n'appartenant pas à race particulière, il s'agit de l'ensemble d'agents `turtles` contenant toutes les tortues et de l'ensemble d'agents `links` contenant tous les liens.) Vous pouvez donner une autre valeur à cette variable pour modifier une race de tortues ou une race de liens.

Voir aussi `breed`, `directed-link-breed` et `undirected-link-breed`

Exemple :

```
breed [cats cat]
breed [dogs dog]
;; code pour la tortue :
if breed = cats [ show "miaou!" ]
set breed dogs

show "whaff!"
directed-link-breed [ roads road ]
;; code pour le lien
if breed = roads [ set color gray ]
```

## breed

`breed [<breeds> <breed>]`

Ce mot-clé, tout comme les mots-clés `globals`, `turtles-own` et `patches-own`, ne peut être utilisé qu'au début du panneau "Procedures", avant les définitions des procédures. Il permet de définir une race de tortue. La première entrée définit le nom de l'ensemble d'agents associé à la race (en général, un nom au pluriel). La seconde entrée définit le nom d'un membre quelconque de cette race (en général, au singulier).

Toute tortue d'une race donnée :

- appartient à l'ensemble d'agents désigné par le nom de sa race,
- a sa variable préprogrammée `breed` (voir définition ci-dessus) initialisée avec le nom de cet ensemble d'agents.

Le plus souvent, l'ensemble d'agents est utilisé avec la commande `ask` pour ne donner des commandes qu'aux tortues appartenant à une race particulière.

```
breed [mice mouse] ;; définit la race souris
breed [frogs frog] ;; définit la race grenouilles

to setup
  clear-all
  create-mice 50                ;; créer 50 tortues de race souris
  ask mice [ set color white ]  ;; en faire des souris blanches
  create-frogs 50               ;; créer 50 tortues de race grenouilles
  ask frogs [ set color green ] ;; en faire des grenouilles vertes
  show [breed] of one-of mice   ;; afficher mice (dans le Centre de commande)
```

```
  show [breed] of one-of frogs  ;; afficher frogs (dans le Centre de commande)
end

show mouse 1
;; écrire (mouse 1)
show frog 51
;; écrire (frog 51)
show turtle 51
;; écrire (frog 51)
```

Voir aussi `globals`, `patches-own`, `turtles-own`, `<breeds>-own`, `create-<breeds>`, `<breeds>-at`, `<breeds>-here`

## but-first
## bf
## but-last
## bl

```
but-first list
but-first string
but-last list
but-last string
```

Utilisé avec une liste, `but-first` retourne tous les éléments de la liste *`list`* sauf le premier, et `but-last` retourne tous les éléments de la liste *`list`* sauf le dernier.

Utilisés avec une chaîne *`string`*, `but-first` et `but-last` retournent cette chaîne raccourcie en omettant le premier, respectivement le dernier caractère de la chaîne originale.

```
;; ma-liste contient [2 4 6 5 8 12]
set ma-liste but-first ma-liste
;; ma-liste contient maintenant [4 6 5 8 12]
set ma-liste but-last ma-liste
ma-liste contient maintenant [4 6 5 8]
show but-first "string"
;; écrit "tring"
show but-last "string"
;; écrit "strin"
```

# C

## can-move?

```
can-move? distance
```


Retourne `true` (vrai) si la tortue appelée peut se déplacer de la distance *`distance`* en suivant son cap actuel sans violer la topologie, sinon retourne `false` (faux).

Cette primitive est l'équivalent de la commande :

```
patch-ahead distance != nobody
```

## carefully

```
carefully [ commands1 ] [ commands2 ]
```

Exécute les commandes *`commands1`*. Si une erreur d'exécution se produit avec *`commands1`*, NetLogo ne s'arrête pas et n'informe pas l'utilisateur de la survenue de l'erreur. Il supprime l'erreur et exécute les commandes *`commands2`* à la place.

Le reporter `error-message` peut être utilisé dans l'entrée *`commands2`* pour savoir quelle erreur a été supprimée dans *`commands1`*. Voir `error-message`.

**Note** : les deux listes de commandes *`commands1`* et *`commands2`* sont exécutées sans interruption (comme avec la commande `without-interruption`).

```
carefully [ show 1 / 1 ] [ print error-message ]
=> 1
carefully [ show 1 / 0 ] [ print error-message ]
=> division by zero      ;; division par zéro
```

## ceiling

```
ceiling number
```

Retourne le plus petit nombre entier plus grand ou égal au nombre *`number`*.

```
show ceiling 4.5
=> 5
show ceiling -4.5
=> -4
```

## clear-all
## ca

```
clear-all
```


Réinitialise toutes les variables globales à zéro et appelle les primitives `reset-ticks`, `clear-turtles`, `clear-patches`, `clear-drawing`, `clear-all-plots` et `clear-output`.

### clear-all-plots

```
clear-all-plots
```


Efface le contenu de tous les traceurs de graphiques du modèle. Voir `clear-plot` pour plus d'informations.

### clear-drawing
### cd

```
clear-drawing
```


Efface toutes les lignes et tous les tampons dessinés par les tortues sur la couche de dessin.

### clear-links

```
clear-links
```


Supprime tous les liens.

Voir aussi `die`

### clear-output

```
clear-output
```


Efface tous les textes affichés dans la zone de sortie du modèle, s'il y en a une. Sinon, ne fait rien.

### clear-patches
### cp

```
clear-patches
```


Efface tous les patches et remet toutes les variables patch à leur valeur initiale par défaut, y compris en redessinant tous les patches en noir.

### clear-plot

```
clear-plot
```

Dans le traceur de graphiques courant seulement, `clear-plot` efface tous les graphiques (les courbes), supprime tous les crayons temporaires, redonne au traceur ses valeurs par défaut (dimension en x, dimension en y, etc.) et réinitialise tous les crayons permanents à leurs valeurs par défaut. Les valeurs par défaut pour le traceur et ses crayons permanents sont spécifiées dans la fenêtre d'édition des traceurs "Plot", fenêtre qui est affichée quand vous éditez le traceur. S'il n'y a plus de crayons une fois tous les crayons temporaires supprimés, autrement dit s'il n'y avait pas de crayon permanent, un crayon permanent par défaut est créé. Ce nouveau crayon reçoit les réglages initiaux suivants :

- Pen : down ; Color : black : Mode : 0 (line mode) ; Name : "default" ; Interval : 1

Voir aussi `clear-all-plots`

## clear-turtles
## ct

```
clear-turtles
```


Supprime toutes les tortues. Cette commande réinitialise aussi le numéro d'identification (`who number`), de manière à ce que la prochaine tortue créée porte le numéro 0.

Voir aussi `die`

## color

```
color
```


C'est une variable tortue ou une variable lien préprogrammée. Elle contient la couleur de la tortue ou du lien. Vous pouvez modifier la valeur de cette variable pour changer la couleur de la tortue ou du lien. La couleur peut être représentée soit par une couleur NetLogo (un seul nombre), soit par une couleur RGB (une liste de trois nombres). Pour une explication détaillée, voir la section Couleurs du Guide de programmation.

Voir aussi `pcolor`

## cos

```
cos number
```

Retourne le cosinus de l'angle *number* donné. S'attend à recevoir un angle exprimé en degrés.

```
show cos 180
=> -1
```

## count

```
count agentset
```

Retourne le nombre d'agents appartenant à l'ensemble d'agents *agentset* donné.

```
show count turtles
;; affiche le nombre total des tortues
show count patches with [pcolor = red]
;; affiche le nombre total des patches rouges
```

## create-ordered-turtles
## cro
## create-ordered-<*breeds*>

```
create-ordered-turtles number
create-ordered-turtles number [ commands ]
create-ordered<breeds> number
create-ordered<breeds> number [ commands ]
```


Crée *number* nouvelles tortues. Toutes les tortues sont créées à la position (0, 0), reçoivent l'une des 14 couleurs primaires NetLogo et ont des orientations (des caps) régulièrement espacés de 0 à 360 (l'angle entre les caps de deux tortues successives dépend du nombre de tortues *number* à créer).

Si la forme `create-ordered-<breeds>` est utilisée, les nouvelles tortues appartiennent à la race *<breeds>* spécifiée dans la commande.

Si une liste *commands* est fournie, les nouvelles tortues exécutent immédiatement les commandes placées entre les crochets (les commandes de la liste). Cette forme est très utile pour donner aux nouvelles tortues une même couleur, un même cap ou quoi que ce soit d'autre. (Les nouvelles tortues sont créées toutes d'un seul coup, puis elles exécutent chacune, les unes après les autres et dans un ordre aléatoire, les commandes transmises.)

```
cro 100 [ fd 10 ]  ;; crée un cercle de tortues régulièrement espacées
```

**Note** : Pendant que ces commandes sont exécutées, aucun autre agent n'a l'autorisation d'exécuter quelque code que ce soit (comme avec la commande `without-interruption`). Cette précaution garantit que si `ask-concurrent` est utilisé, les nouvelles tortues ne puissent pas interagir avec n'importe quel autre agent avant qu'elles n'aient toutes été complètement initialisées.

## create-<breed>-to
## create-<breeds>-to
## create-<breed>-from
## create-<breeds>-from
## create-<breed>-with
## create-<breeds>-with
## create-link-to
## create-links-to
## create-link-from
## create-links-from
## create-link-with
## create-links-with

```
create-<breed>-to turtle
create-<breed>-to turtle [ commands ]
create-<breed>-from turtle
create-<breed>-from turtle [ commands ]
create-<breed>-with turtle
create-<breed>-with turtle [ commands ]
create-<breeds>-to turtleset
create-<breeds>-to turtleset [ commands ]
create-<breeds>-from turtleset
create-<breeds>-from turtleset [ commands ]
create-<breeds>-with turtleset
create-<breeds>-with turtleset [ commands ]
create-link-to turtle
create-link-to turtle [ commands ]
create-link-from turtle
create-link-from turtle [ commands ]
create-link-with turtle
create-link-with turtle [ commands ]
create-links-to turtleset
create-links-to turtleset [ commands ]
create-links-from turtleset
create-links-from turtleset [ commands ]
create-links-with turtleset
create-links-with turtleset [ commands ]
```

Ces primitives permettent de créer des liens entre les tortues d'une race ou entre les tortues sans races.

`create-link-with` crée un lien non orienté entre l'appelant et l'*agent*.

`create-link-to` crée un lien orienté de l'appelant vers l'*agent*.

`create-link-from` crée un lien orienté de l'*`agent`* vers l'appelant.

Quand la forme plurielle du nom de la race (*<breeds>*) est utilisée, la primitive doit recevoir en entrée un ensemble d'agents *`turtleset`* à la place d'un agent et des liens sont créés entre l'appelant et tous les agents de l'ensemble d'agents.

Le bloc de commandes optionnel `[ commands ]` est l'ensemble des commandes que chaque nouveau lien formé doit exécuter. (D'abord tous les liens sont créés, puis ils exécutent les uns après les autres et dans un ordre aléatoire les commandes qui leur ont été passées.)

Un noeud ne peut être lié à lui-même. De plus, vous ne pouvez pas avoir plus d'un lien non orienté de la même race entre deux noeuds, tout comme vous ne pouvez pas avoir plus d'un seul lien orienté de même race allant dans la même direction entre deux noeuds.

Si vous essayez de créer un lien là où un lien (de même race) existe déjà, rien ne se passe. Si vous tentez de créer un lien d'une tortue vers elle-même, vous obtenez une erreur d'exécution.

```
to setup
  crt 5
  ;; la tortue 1 crée des liens avec toutes les autres tortues
  ;; le lien entre la tortue et elle-même est ignoré
  ask turtle 0 [ create-links-with other turtles ]
  show count links ;; affiche 4
  ;; ne fait rien puisque le lien existe déjà
  ask turtle 0 [ create-link-with turtle 1 ]
  show count links ;; affiche 4 puisque le lien à créer existe déjà
  ask turtle 2 [ create-link-with turtle 1 ]
  show count links ;; affiche 5
end

;; définir les races de liens
directed-link-breed [red-links red-link]
undirected-link-breed [blue-links blue-link]

to setup
  crt 5
  ;; créer des liens dans les deux directions entre la tortue 0
  ;; et toutes les autres tortues
  ask turtle 0 [ create-red-links-to turtles ]
  ask turtle 0 [ create-red-links-from turtles ]
  show count links ;; affiche 8
  ;; créer maintenant des liens non orientés entre la tortue 0
  ;; et les autres tortues
  ask turtle 0 [ create-blue-links-with turtles ]
  show count links ;; affiche 12
end
```

## create-turtles
## crt
## create-*<breeds>*

```
create-turtles number
create-turtles number [ commands ]
create-<breeds> number
create-<breeds> number [ commands ]
```

Crée *`number`* nouvelles tortues. Les nouvelles tortues sont placées au centre du monde (coordonnées 0 0), ont une orientation aléatoire (nombre entier de 0 à 360) et une couleur choisie au hasard parmi les 14 couleurs primaires de NetLogo.

Si la forme `create-<breeds>` est utilisée, toutes les nouvelles tortues appartiennent à la race spécifiée.

Si une liste de commandes `[ commands ]` est fournie, les nouvelles tortues exécutent immédiatement les commandes données entre crochets. Cette forme est très utile pour donner aux nouvelles tortues une même couleur, un même cap, ou quoi que ce soit d'autre. (Les nouvelles tortues sont toutes créées d'un coup, puis exécutent chacune, les unes après les autres et dans un ordre aléatoire, les commandes transmises.)

```
crt 100 [ fd 10 ] ;; crée un ensemble de tortues réparties aléatoirement
breed [canaries canary]
breed [snakes snake]

to setup
  clear-all
  create-canaries 50 [ set color yellow ]
  create-snakes 50 [ set color green ]
end
```

**Note** : pendant que les commandes sont exécutées, aucun autre agent n'a l'autorisation d'exécuter quelque code que ce soit (comme avec la commande `without-interruption`). Cette précaution garantit que si `ask-concurrent` est utilisé, les nouvelles tortues ne puissent pas interagir avec n'importe quel autre agent avant qu'elles n'aient toutes été complètement initialisées.

Voir aussi `hatch` et `sprout`

## create-temporary-plot-pen

`create-temporary-plot-pen` *`string`*

Un nouveau crayon de traceur de courbes temporaire portant le nom spécifié par *`string`* est créé dans le traceur courant et désigné comme crayon courant.

Peu de modèles devront utiliser cette primitive car tous les crayons temporaires disparaissent quand les primitives `clear-plot` ou `clear-all-plots` sont appelées. La méthode normale pour créer un crayon est de faire un crayon permanent dans le fenêtre d'édition du traceur "Plot".

Si un crayon temporaire portant le même nom existe déjà dans le traceur courant, aucun nouveau crayon n'est créé et le crayon existant devient le crayon par défaut. Si un crayon permanent portant le même nom existe déjà dans le traceur courant, vous obtenez une erreur d'exécution.

Le nouveau crayon temporaire est créé avec les réglages initiaux suivants :

- `Pen : down` ; `Color : black` ; `Mode : 0` (line mode) ; `Interval : 1`

Voir aussi : `clear-plot`, `clear-all-plots` et `set-current-plot-pen`

# D

## date-and-time

```
date-and-time
```

Retourne une chaîne contenant la date et l'heure du système. Le format est montré ci-dessous. Tous les champs ont une largeur fixe, ce qui fait qu'ils occupent toujours la même place dans la chaîne. La précision potentielle de l'horloge est la milliseconde. (En pratique, l'obtention de cette précision peut varier d'un système à l'autre, dépendant notamment des performances de la Machine Java Virtuelle sous-jacente.)

```
show date-and-time
=> "01:19:36.685 PM 19-Sep-2002"
```

## die

```
die
```


Fait mourir la tortue ou le lien. Ils ne font dès lors plus partie de la simulation.

```
if xcor > 20 [ die ]
;; toutes les tortues avec xcor > 20 meurent
ask links with [color = blue] [ die ]
;; tous les liens de couleur bleue meurent
```

Voir aussi `clear-turtles` et `clear-links`

## diffuse

```
diffuse patch-variable number
```


Demande à chaque patch de distribuer à part égale la valeur (*number* * 100) pour cent de la valeur de la variable *patch-variable* à ses huit patches voisins. *number* peut prendre toute valeur de 0 à 1. Quelle que soit la topologie utilisée, la somme de *patch-variable* sera conservée de par le monde. (Si un patch a moins de huit voisins, chaque voisin reçoit un huitième de la somme à partager et le patch « distributeur » garde ce qu'il n'a pas pu donner.)

Notez que c'est une commande réservée à l'observateur, même si vous aimeriez que ce soit une commande patch. (La raison est que cette commande doit agir sur tous les patches d'un coup alors que les commandes patch agissent sur les patches individuels.)

```
diffuse chemical 0.5
;; chaque patch diffuse 50% de sa variable chemical
;; à ses 8 patches voisins. Donc, chaque patch
;; reçoit 1/8 de 50% de la variable chemical
;; de chaque patch voisin.
```

## diffuse4

```
diffuse4 patch-variable number
```


Fonctionne comme la primitive `diffuse`, mais ne transmet qu'à ses quatre patches voisins (au nord, au sud, à l'est et à l'ouest), pas aux patches voisins situés sur les diagonales.

```
diffuse4 chemical 0.5
;; chaque patch diffuse 50% de sa variable chemical
;; à ses 4 patches voisins. Donc, chaque patch
;; reçoit 1/4 de 50% de la variable chemical
;; de chaque patch voisin.
```

## directed-link-breed

```
directed-link-breed [ <link-breeds> <link-breed> ]
```

Ce mot-clé, tout comme les mots-clés `globals` et `breeds`, ne peut être utilisé qu'au début du panneau "Procedures", avant toute définition de procédures. Il définit une race de liens orientés. Les liens d'une race particulière doivent être soit tous orientés, soit tous non orientés. Le premier membre de la liste, *<link-breeds>*, spécifie le nom de l'ensemble d'agents associé avec le lien de race. Le second membre, *<link-breed>* définit le nom d'un seul individu de cette race. Les liens orientés peuvent être créés en utilisant les primitives `create-link(s)-to` et `create-link(s)-from`, mais pas la primitive `create-link(s)-with`.

Tout lien de la race de liens donnée :

- ✔ fait partie de l'ensemble d'agents spécifié par le nom de la race de liens,
- ✔ possède sa variable préprogrammée `breed` initialisée avec le nom de cet ensemble d'agents,
- ✔ est orienté ou non orienté selon ce qui est déclaré par le mot-clé.

Le plus souvent, l'ensemble d'agents est utilisé avec la commande `ask` pour ne donner des commandes qu'aux liens d'une race particulière.

```
directed-link-breed [streets street]
directed-link-breed [highways highway]

to setup
  clear-all
  crt 2
  ;; créer un lien de la tortue 0 à la tortue 1
  ask turtle 0 [ create-street-to turtle 1 ]
  ;; créer un lien de la tortue 1 à la tortue 0
  ask turtle 0 [ create-highway-from turtle 1 ]
end

ask turtle 0 [ show one-of in-links ]
;; affiche (street 0 1)
ask turtle 0 [ show one-of out-links ]
;; affiche (highway 1 0)
```

Voir aussi `breed` et `undirected-link-breed`

## display

```
display
```

Provoque une mise à jour immédiate de la Vue. (Exception : si l'utilisateur utilise le curseur de vitesse "speed" pour accélérer la simulation, la mise à jour peut être sautée.)

Cette primitive annule aussi l'effet de la commande `no-display` de manière à ce que si les mises à jour de la Vue avaient été désactivées, elles soient à nouveau activées.

```
no-display
ask turtles [ jump 10 set color blue set size 5 ]
display
;; les tortues se déplacent, changent de couleurs et grandissent sans qu'aucun
;; de ces états intermédiaires ne soient visibles par l'utilisateur, seul l'état
;; final sera affiché une fois toutes les modifications effectuées
```

Même si `no-display` n'a pas été utilisé, `display` peut quand même être utile car, normalement, NetLogo est libre de sauter certaines mises à jours de la Vue, parce que moins il y a de mises à jours, plus la simulation tourne rapidement. Cette commande permet de forcer une mise à jour de manière à ce que quel que soit le changement intervenu dans le monde, il puisse être montré à l'utilisateur.

```
ask turtles [ set color red ]
display
ask turtles [ set color blue]
;; les tortues deviennent rouges, puis bleues; l'utilisation de
;; "display" force les tortues rouges à apparaître brièvement
```

Notez que les commandes `display` et `no-display` opèrent indépendamment de l'état de la case à cocher "view updates" de la barre d'outils du panneau "Interface" qui permet de « geler » la vue.

Voir aussi `no-display`

## distance

`distance` *`agent`*

Retourne la distance qu'il y a entre l'agent appelant et la tortue ou au patch spécifié par *`agent`*.

La distance jusqu'à un patch ou à partir d'un patch est toujours mesurée à partir du centre du patch. Les tortues et les patches utilisent la distance « enroulée » (autour des bords du monde) si l'enroulement est autorisé par la topologie et si la distance « enroulée » est plus courte.

```
ask turtles [ show max-one-of turtles [distance myself] ]
;; chaque tortue affiche la tortue qui est le plus loin d'elle
```

## distancexy

`distancexy` *`xcor ycor`*

Retourne la distance entre l'agent courant et le point de coordonnées ( *`xcor`*, *`ycor`* ).

La distance à partir d'un patch est toujours mesurée à partir du centre du patch. Les tortues et les patches utilisent la distance « enroulée » (autour des bords du monde) si l'enroulement est autorisé par la topologie et si la distance « enroulée » est plus courte.

```
if (distancexy 0 0) > 10
  [ set color green ]
;; toutes les tortues situées à plus de 10 unités
;; du centre du monde deviennent vertes
```

## downhill
## downhill4

```
downhill patch-variable
downhill4 patch-variable
```

Déplace la tortue sur le patch voisin dont la variable *variable-patch* a la plus petite valeur. Si aucun patch voisin n'a sa *variable-patch* plus petite que celle du patch courant, la tortue reste où elle est. Si plusieurs patches voisins ont la même plus petite valeur, la tortue en choisit un au hasard. Les valeurs non numériques sont ignorées.

Pour la comparaison des valeurs, la primitive `downhill` prend en compte les huit patches voisins alors que la primitive `downhill4` ne prend en compte que les quatre voisins ayant un côté adjacent à l'un des siens.

Cette primitive est équivalente au code suivant (partant du principe que les variables ont des valeurs numériques) :

```
move-to patch-here  ;; va au centre du patch
let p min-one-of neighbors [patch-variable]  ;; ou neighbors4
if [patch-variable] of p < patch-variable
  [ face p
    move-to p
  ]
```

Notez que la tortue se retrouve toujours au centre du patch et que son orientation (cap) est un multiple de 45 (avec `downhill`) ou de 90 (avec `downhill4`).

Voir aussi `uphill` et `uphill4`

## dx
## dy

```
dx
dy
```

Retourne l'incrément en x (reporter *dx*) ou l'incrément en y (reporter *dy*). Cet incrément est la valeur dont la coordonnée `xcor` ou `ycor` de la tortue changera si la tortue doit avancer d'un pas en suivant son cap actuel.

**Note** : `dx` est simplement le sinus du cap de la tortue et `dy` en est simplement le cosinus. (Si vous vous attendiez au contraire, c'est parce qu'en NetLogo, une orientation de 0 correspond au Nord et une orientation de 90 à l'Est, ce qui est l'inverse de la manière dont les angles sont habituellement définis en géométrie.)

**Note** : dans les versions précédentes de NetLogo, ces primitives étaient utilisées dans bien des situations pour lesquelles la nouvelle primitive `patch-ahead` est nettement plus appropriée.

# E

## empty?

```
empty? list
empty? string
```

Retourne `true` (vrai) si la liste *`list`* ou la chaîne de caractères *`string`* est vide, sinon retourne `false` (faux).

**Note** : la liste vide est notée `[]`. La chaîne vide est notée `""`.

## end

```
end
```

Ce mot-clé **doit** être utilisé pour terminer la définition d'une procédure. De plus, il doit être seul sur sa ligne de code.

Voir aussi `to` et `to-report`

## end1

```
end1
```

`end1` est une variable de lien préprogrammée. Elle désigne la première extrémité (le premier noeud ou la première tortue) d'un lien. Pour les liens orientés, il s'agit toujours de la source, pour les liens non-orientés, il s'agit toujours de la tortue qui a le plus petit numéro d'identification (who-number). Vous ne pouvez ni spécifier ni modifier la valeur de `end1`.

```
crt 2
ask turtle 0
[ create-link-to turtle 1 ]
ask links
[ show end1 ] ;; affiche turtle 0
```

## end2

```
end2
```

`end2` est une variable de lien préprogrammée. Elle désigne la seconde extrémité (le second noeud ou la seconde tortue) d'un lien. Pour les liens orientés, il s'agit toujours de la destination, pour les liens non-orientés, il s'agit toujours de la tortue qui a le plus grand numéro d'identification (who-number). Vous ne pouvez ni spécifier ni modifier la valeur de `end2`.

```
crt 2
ask turtle 1
[ create-link-with turtle 0 ]
ask links
[ show end2 ] ;; affiche turtle 1
```

### error-message

```
error-message
```

Retourne une chaîne de caractères décrivant l'erreur qui a été supprimée par la primitive `carefully`.

Le reporter `error-message` ne peut être utilisé que dans le second bloc de la commande `carefully`.

Voir aussi `carefully`

### every

```
every number [ commands ]
```

Exécute la liste des commandes `[commands]` données seulement s'il s'est écoulé plus de *number* secondes depuis la dernière fois que cet agent les a exécutées dans ce contexte. Sinon, les commandes de la liste sont ignorées.

La primitive `every` n'exécute pas d'elle-même les commandes de la liste de manière répétée. Vous devez placer `every` dans une boucle ou dans un bouton "forever" (pour-toujours) si vous voulez que les *commands* soient exécutées plusieurs fois de suite. Donc, `every` ne contrôle que la fréquence d'exécution de ses commandes.

L'expression « dans ce contexte » utilisée ci-dessus signifie « au cours de la même demande `ask` » (ou de la pression sur un bouton ou pour la commande entrée dans le Centre de commande). Il ne sert donc à rien d'écrire `ask turtles [ every 0.5 [ ... ] ]`, parce que quand l'exécution de la commande `ask` se termine, toutes les tortues suppriment leur compteur pour *every*. L'utilisation correcte de cette commande est montrée ci-dessous :

```
every 0.5 [ ask turtles [ fd 1 ] ]
;; les tortues avancent d'un pas deux fois par seconde
every 2 [ set index index + 1 ]
;; toutes les deux secondes, index est incrémenté
```

Voir aussi `wait`

### exp

```
exp number
```

Retourne la valeur de e élevé à la puissance *number*.

**Note** : cette primitive fait la même chose que le code `e ^ nombre`.

### export-view
### export-interface
### export-output
### export-plot
### export-all-plots
### export-world

```
export-view filename
export-interface filename
export-output filename
export-plot plotname filename
export-all-plots filename
export-world filename
```

**export-view** sauvegarde l'image affichée dans la Vue courante dans un fichier image externe (sur le disque) dont le nom est spécifié par la chaîne *filename*. Le fichier image sauvegardé étant au format PNG (**P**ortable **N**etwork **G**raphics), il est recommandé de fournir un nom de fichier se terminant par `.png`.

**export-interface** fonctionne de la même manière, mais ici c'est l'image de tout le panneau "Interface" qui est sauvegardée.

**export-output** sauvegarde le contenu de la zone de sortie du modèle dans un fichier externe dont le nom est spécifié par la chaîne *filename*. (Si le modèle n'a pas de zone de sortie séparée, c'est le contenu de la zone sortie du Centre de commande qui est sauvegardé.)

**export-plot** sauvegarde les valeurs x et y (les coordonnées) de tous les points dessinés par tous les crayons du traceur désigné par la chaîne *plotname* dans le fichier externe spécifié par la chaîne *filename*. Si un crayon est en mode barre (mode 0) et que la valeur y du point est plus grande que 0, ce sont les coordonnées du coin supérieur gauche de la barre qui sont exportées. Si la valeur y est inférieure à 0, ce sont les coordonnées du coin inférieur gauche de la barre qui sont exportées.

**export-all-plots** sauvegarde le contenu de tous les traceurs du modèle dans un fichier externe spécifié par la chaîne *filename*. Le format de chaque traceur est identique à celui généré par la commande `export-plot`.

**export-world** sauvegarde les valeurs de toutes les variables, c'est-à-dire les valeurs des variables préprogrammées, celles des variables définies dans le code du modèle (par le programmeur), celles des variables de l'observateur, celles des variables de toutes les tortues et celles des variables de tous les patches, l'image laissée par les tortues (ou chargée) dans la couche de dessin, le contenu de la zone de sortie (si elle existe), le contenu de tous les traceurs de courbes et l'état du générateur de nombres aléatoires dans un fichier externe dont la nom est spécifié par la chaîne *filename*. (Le fichier résultant peut être chargé dans NetLogo au moyen de la primitive `import-world`.) Par contre, `export-world` ne sauvegarde pas l'état des fichiers ouverts.

Les fichiers créés par `export-plot`, `export-all-plots` et `export-world` sont des fichiers au format texte à « valeurs séparées par des virgules », format appelé CSV (**C**ommas **S**eparated **V**alues), et d'extension de nom `.csv`. Les fichiers CSV peuvent être lus par la plupart des tableurs et des bases de données ainsi que par tout éditeur de texte.

Si un ficher de même nom existe déjà, son contenu est écrasé et remplacé par le nouveau.

Si vous voulez exporter dans un fichier se trouvant à un autre endroit que le fichier du modèle, vous devez mettre dans *filename* la totalité du chemin du fichier que vous voulez exporter. (Utilisez la barre oblique, `/`, comme séparateur de dossiers.)

Notez que les fonctionnalités de ces primitives sont aussi directement accessibles dans le menu "File" de NetLogo.

```
export-world "fire.csv"
;; exporte l'état du modèle dans le fichier fire.csv
;; situé dans le dossier NetLogo
export-plot "Temperature" "c:/My Documents/plot.csv"
;; exporte le contenu du traceur nommé
;; "Temperature" dans le fichier plot.csv situé
;; dans le dossier C:\My Documents
export-all-plots "c:/My Documents/plots.csv"
;; exporte tous les traceurs dans le fichier plots.csv situé dans le dossier C:\My Documents
```

## extensions

`extensions [name ...]`

Permet au modèle d'utiliser les primitives définies dans les extensions spécifiées dans la liste `[name ...]` donnée en entrée.

Voir le Guide des extensions pour des informations supplémentaires concernant ce sujet.

## extract-hsb

```
extract-hsb color
```

Retourne une liste de trois valeurs, prises dans l'intervalle 0 à 255, représentant la teinte (**h**ue), la **s**aturation et la **b**rillance de la couleur NetLogo désignée par son numéro de couleur *color* (de 0 à 140, 140 non compris) ou par son nom (pour les 16 teintes de base).

```
show extract-hsb red
=> [2.198 206.372 215]
show extract-hsb cyan
=> [127.5 145.714 196]
```

Voir aussi approximate-hsb, approximate-rgb et extract-rgb

## extract-rgb

```
extract-rgb color
```

Retourne une liste de trois valeurs, prises dans l'intervalle 0 à 255, représentant les niveau de **r**ouge, de vert (**g**reen) et de **b**leu de la couleur NetLogo désignée par son numéro de couleur *color* (de 0 à 140, 140 non compris) ou par son nom (pour les 16 teintes de base).

```
show extract-rgb red
=> [215 50 41]
show extract-rgb cyan
=> [84 196 196]
```

Voir aussi approximate-rgb, approximate-hsb et extract-hsb

# F

## face

```
face agent
```


Fait pivoter l'appelant (donc modifie son cap) de manière à ce qu'il soit face à (que son cap pointe vers) la cible spécifiée par l'entrée *agent*.

Si l'enroulement (wrapping) est autorisé par la topologie et que la distance « enroulée » (autour des bords du monde) est plus courte, `face` utilisera le cap suivant le chemin « enroulé ».

Si l'appelant et l'*agent* ont exactement les mêmes coordonnées , le cap de l'appelant n'est pas modifié.

## facexy

```
facexy number-x number-y
```


Fait pivoter l'appelant (donc modifie son cap) de manière à ce qu'il soit face à (que son cap pointe vers) la cible spécifiée par ses coordonnées (*number-x*, *number-y*).

Si l'enroulement (wrapping) est autorisé par la topologie et que la distance « enroulée » (autour des bords du monde) est plus courte, `facexy` utilisera le cap suivant le chemin « enroulé ».

Si l'appelant est au point de coordonnées (*number-x*, *number-y*), le cap de l'appelant n'est pas modifié.

## file-at-end?

```
file-at-end?
```

Retourne `true` (vrai) quand il n'y a plus de caractères à lire dans le fichier courant (qui a été préalablement ouvert par la primitive `file-open`). Sinon retourne `false` (faux).

```
file-open "mon-fichier.txt"
print file-at-end?
=> false ;; il y a encore des caractères à lire
print file-read-line
=> This is the last line in file
print file-at-end
=> true ;; il n'y a plus de caractères à lire,
        ;; nous avons atteint la fin du fichier
```

Voir aussi `file-open`, `file-close-all`

## file-close

```
file-close
```

Ferme un fichier préalablement ouvert avec `file-open`.

Notez que cette commande et la commande `file-close-all` sont les deux seuls moyens de revenir au début d'un fichier ouvert et de changer le mode d'utilisation de ce fichier.

Si aucun fichier n'est ouvert, cette primitive ne fait rien.

Voir aussi `file-close-all`, `file-open`

## file-close-all

```
file-close-all
```

Ferme tous les fichiers (s'il y en a) préalablement ouverts avec `file-open`.

Notez que cette commande et la commande `file-close` sont les deux seuls moyens de revenir au début d'un fichier ouvert et de changer le mode d'utilisation de ce fichier.

Voir aussi `file-close`, `file-open`

## file-delete

```
file-delete string
```

Supprime le fichier dont le nom est spécifié par la chaîne de caractères *`string`*.

L'entrée *`string`* doit désigner un fichier existant avec autorisation d'écriture pour l'utilisateur. De plus, ce fichier ne doit pas être ouvert. Utilisez la commande `file-close` pour fermer un fichier ouvert avant de pouvoir le supprimer.

Notez que la chaîne de caractères *`string`* peut être soit un simple nom de fichier, soit un nom de chemin de fichier absolu. Si c'est un nom de fichier, la commande recherche ce fichier dans le répertoire courant. Le répertoire courant peut être changé avec la commande `set-current-directory`. Par défaut, le répertoire courant est le répertoire dans lequel se trouve le modèle courant.

## file-exists?

```
file-exists? string
```

Retourne `true` (vrai) si l'entrée *`string`* est le nom d'un fichier existant dans le système. Sinon, retourne `false` (faux).

Notez que la chaîne de caractères *`string`* peut être soit un simple nom de fichier, soit un nom de chemin de fichier absolu. Si c'est un nom de fichier, la commande recherche ce fichier dans le répertoire courant. Le répertoire courant peut être changé avec la commande `set-current-directory`. Par défaut, le répertoire courant est le répertoire dans lequel se trouve le modèle courant.

## file-flush

```
file-flush
```

Force NetLogo d'écrire sur le disque toutes les modifications de données faites en mémoire. Quand vous utilisez `file-write` ou d'autres commandes de sortie, les valeurs en mémoire peuvent ne pas être immédiatement enregistrées sur le disque. Cette commande améliore les performances des commandes de sortie de fichiers. La fermeture d'un fichier garantit que toutes les modifications sont écrites sur le disque.

Vous avez parfois besoin d'écrire des données sur le disque sans fermer le fichier. Par exemple, vous pourriez utiliser un fichier pour communiquer avec un autre programme se trouvant sur votre machine et voulez que cet autre programme puisse immédiatement lire les sorties de NetLogo.

## file-open

```
file-open string
```

Cette commande interprète la chaîne *`string`* comme étant le nom de chemin d'un fichier et ouvre ce fichier. Vous pouvez ensuite utiliser les reporters `file-read`, `file-read-line` et `file-read-characters` pour lire dans ce fichier, ou `file-write`, `file-print`, `file-type` ou `file-show` pour écrire dans ce fichier.

Notez que vous ne pouvez ouvrir un fichier qu'en lecture ou qu'en écriture, jamais pour les deux opérations à la fois. La prochaine primitive d'entrée/sortie de fichier que vous utiliserez après cette commande dictera le mode dans lequel le fichier sera ouvert. Pour changer de mode, vous devez d'abord fermer le fichier avec la commande `file-close`.

De plus, il faut que le fichier existe déjà pour pouvoir être ouvert en mode lecture.

Quand un fichier est ouvert en mode écriture, toutes les nouvelles données sont ajoutées à la fin du fichier original. S'il n'y a pas de fichier original, NetLogo crée un nouveau fichier correspondant à l'entrée *`string`*. (Vous devez avoir l'autorisation d'écriture dans le dossier de destination.) Si vous ne voulez pas ajouter les données à la suite des données existantes mais les utiliser pour remplacer (écraser) le contenu actuel du fichier, utilisez d'abord `file-delete` pour supprimer le fichier, peut-être à l'intérieur d'une commande `carefully` si vous n'êtes pas certain de son existence.

Notez que la chaîne de caractères *`string`* peut être soit un simple nom de fichier, soit un nom de chemin de fichier absolu. Si c'est un nom de fichier, la commande recherche ce fichier dans le répertoire courant. Le répertoire courant peut être changé avec la commande `set-current-directory`. Par défaut, le répertoire courant est le répertoire dans lequel se trouve le modèle courant.

```
file-open "my-file-in.txt"
print file-read-line
=> First line in file ;; le fichier est en mode lecture
file-open "C:\\NetLogo\\my-file-out.txt"
;; sous Windows
file-print "Hello World" ;; le fichier est en mode écriture
```

Voir aussi `file-close`

## file-print

`file-print` *`value`*

Écrit, dans un fichier ouvert, les données *`value`* suivies d'un retour de chariot.

Le nom de l'agent appelant n'est pas écrit dans le fichier avant *`value`*, contrairement à ce qui se passe avec la primitive `file-show`.

Notez que cette commande est l'équivalent, pour les entrées/sorties de fichiers, de la primitive `print`, et que la commande `file-open` doit être appelée avant que `file-print` puisse être utilisée.

Voir aussi `file-show`, `file-type` et `file-write`

## file-read

`file-read`

Ce reporter lit la constante suivante dans le fichier ouvert et l'interprète comme si elle avait été écrite dans le Centre de commande. Elle retourne la valeur résultante. Ce résultat peut être un nombre, une liste, une chaîne, un booléen ou la valeur spéciale `nobody`.

Le caractère "espace" sépare les constantes. Chaque appel de `file-read` saute les espaces avant et après.

Notez que les chaînes doivent être entourées de guillemets doubles. Utilisez la commande `file-write` pour avoir une écriture automatique de ces guillemets.

Notez aussi qu'il faut d'abord ouvrir le fichier avec la commande `file-open` avant de pouvoir lire dans ce fichier, et qu'il doit encore rester des données à lire dans le fichier. Utilisez le reporter `file-at-end?` pour savoir si la fin du fichier a été atteinte.

```
file-open "mon-fichier.data"
print file-read + 5
;; Si la valeur suivante est le nombre 1
=> 6
print length file-read
;; Si la valeur suivante est la liste [1 2 3 4]
=> 4
```

Voir aussi `file-open` et `file-write`

## file-read-characters

`file-read-characters` *`number`*

Lit dans le fichier ouvert le nombre de caractères spécifié par *`number`* et les retourne sous forme de chaîne de caractères. S'il reste moins de caractères à lire que le nombre spécifié dans *`number`*, la chaîne retournée contient les caractères restants.

Cette commande retourne tous les caractères, y compris les caractères "nouvelle ligne" et "espace".

Notez aussi qu'il faut d'abord ouvrir le fichier avec la commande `file-open` avant de pouvoir lire dans ce fichier, et qu'il doit encore rester des données à lire dans le fichier. Utilisez le reporter `file-at-end?` pour savoir si la fin du fichier a été atteinte.

```
file-open "mon-fichier.txt"
print file-read-characters 5
;; Si la ligne courante du fichier est "Hello World"
=> Hello
```

Voir aussi `file-open`

## file-read-line

`file-read-line`

Lit la ligne suivante dans le fichier ouvert et la retourne sous forme de chaîne. Elle reconnaît la fin de la ligne grâce au caractère "retour de chariot", au caractère "fin de fichier" ou à ces deux caractères dans une même ligne. Elle ne retourne pas ces caractères de fin de ligne.

Notez aussi qu'il faut d'abord ouvrir le fichier avec la commande `file-open` avant de pouvoir lire dans ce fichier, et qu'il doit encore rester des données à lire dans le fichier. Utilisez le reporter `file-at-end?` pour savoir si la fin du fichier a été atteinte.

```
file-open "my-file.txt"
print file-read-line
=> Hello World
```

Voir aussi `file-open`

## file-show

`file-show` *`value`*

Écrit les données contenues dans l'entrée *`value`* dans un fichier ouvert en les faisant précéder du nom de l'agent appelant et en les faisant suivre du caractère "retour de chariot". (L'agent appelant est inclus afin de vous aider à savoir quel agent a produit quelle ligne du fichier.) De plus, toutes les chaînes sont automatiquement entourées de guillemets, comme avec la primitive `file-write`.

Cette commande est l'équivalent pour les entrées/sorties de fichiers de la commande `show` et il faut d'abord ouvrir le fichier avec la commande `file-open` avant de pouvoir écrire dans ce fichier.

Voir aussi `file-print`, `file-type` et `file-write`

## file-type

```
file-type value
```

Écrit les données contenues dans l'entrée *value* dans un fichier ouvert, **sans** les faire suivre du caractère "retour de chariot" (contrairement à ce qui se passe avec les primitives `file-print` et `file-show`). L'absence du "retour de chariot" permet d'écrire plusieurs valeurs sur la même ligne du ficher.

Le nom de l'agent appelant n'est **pas** écrit avant les données de *value*, contrairement à ce qui se passe avec la primitive `file-show`.

Cette commande est l'équivalent, pour les entrées/sorties de fichiers, de `type` et la commande `file-open` doit être appelée avant que `file-type` puisse être utilisée.

Voir aussi `file-print`, `file-show` et `file-write`

## file-write

```
file-write value
```

Cette commande écrit les données contenues dans l'entrée *value*, qui peuvent être un nombre, une chaîne de caractères, une liste, un booléen ou la valeur spéciale `nobody` dans un fichier ouvert, sans les faire suivre d'un "retour de chariot" (contrairement à ce qui se passe avec les primitives `file-print` et `file-show`).

Le nom de l'agent appelant n'est **pas** écrit avant les données de *value*, contrairement à ce qui se passe avec la primitive `file-show`. Les données écrites dans le fichier contiennent aussi les guillemets autour des chaînes de caractères et sont précédées d'un caractère "espace". Ces données sont écrites de manière à ce que la primitive `file-read` puisse les interpréter.

Cette commande est l'équivalent pour les entrées/sorties de fichiers de `write` et la primitive `file-open` doit être appelé avant que `file-write` puisse être utilisée.

```
file-open "locations.txt"
ask turtles
  [ file-write xcor file-write ycor ]
```

Voir aussi `file-print`, `file-show` et `file-type`

## filter

```
filter [reporter] list
```

Retourne une liste ne contenant que les éléments de la liste en entrée *list* pour lesquels le *reporter* booléen vaut `true` (vrai) — en d'autres mots, les éléments de la liste *list* qui satisfont à la condition donnée.

Dans *reporter*, il faut utilisez `?` pour faire référence à l'élément courant de la liste *list*.

```
show filter [? < 3] [1 3 2]
=> [1 2]
show filter [first ? != "t"] ["hi" "there" "everyone"]
=> ["hi" "everyone"]
```

Voir aussi `map`, `reduce` et `?`

## first

```
first list
first string
```

Utilisée avec une liste, retourne le premier élément (de rang 0) de la liste *list* donnée en entrée.

Utilisée avec une chaîne de caractères, cette primitive retourne une chaîne mono-caractère ne contenant que le premier caractère de la chaîne *string* donnée en entrée.

## floor

```
floor number
```

Retourne le plus grand nombre entier plus petit ou égal à l'entrée *number*.

```
show floor 4.5
=> 4
show floor -4.5
=> -5
```

## follow

```
follow turtle
```


Semblable à la primitive `ride`, mais dans la "Vue 3D" le point de vue de l'observateur se trouve derrière et au-dessus de la tortue désignée par son identificateur *turtle*. Il la suit.

Voir aussi `follow-me`, `ride`, `reset-perspective`, `watch` et `subject`

## follow-me

```
follow-me
```


Demande à l'observateur du suivre la tortue appelante.

Voir aussi `follow`

## foreach

```
foreach list [ commands ]
(foreach list1 ... [ commands ])
```

Quand elle ne reçoit qu'une seule liste, cette primitive exécute les *commands* placées dans la liste de la deuxième entrée pour chaque élément de la première entrée *list*. Dans les *commands*, il faut utiliser `?` pour faire référence à l'élément courant de la liste *list*.

```
foreach [1.1 2.2 2.6] [ show (word ? " -> " round ?) ]
=> 1.1 -> 1
=> 2.2 -> 2
=> 2.6 -> 3
```

Quand elle reçoit plusieurs listes, la primitive `foreach` exécute les *commands* reçues pour chaque groupe d'éléments de chaque liste *list1*, *list2*, etc., passée en entrée avant la liste des commandes. Ces commandes sont donc exécutées une première fois pour le premier élément de chaque liste, une seconde fois pour le deuxième élément de chaque liste, et ainsi de suite. Toutes les listes doivent avoir la même lon-

gueur. Dans les *commands*, il faut utilise les primitives `?1` à `?n` (où n est le numéro d'ordre du dernier élément de chaque liste) pour faire référence à l'élément courant de chaque *list*.

Voici quelques exemples pour illustrer cette commande :

```
(foreach [1 2 3] [2 4 6]
  [ show word "la somme est: " (?1 + ?2) ])
=> "la somme est: 3"
=> "la somme est: 6"
=> "la somme est: 9"
(foreach list (turtle 1) (turtle 2) [3 4] [ ask ?1 [ fd ?2 ] ])
;; la tortue 1 avance de 3 patches
;; la tortue 2 avance de 4 patches
```

Voir aussi `map` et `?`

## forward
## fd

`forward` *number*

La tortue avance du nombre de pas spécifié dans l'entrée *number*, un pas à la fois. (Si *number* est négatif, la tortue recule.)

La commande `fd 10` est équivalente à `repeat 10 [ jump 1 ]` et la commande `fd 10.5` est équivalente à `repeat 10 [ jump 1 ] jump 0.5`.

Si la tortue ne peut avancer du nombre de pas demandé parce que la topologie du monde NetLogo ne le permet pas, la tortue fait autant de pas de 1 unité qu'elle peut puis s'arrête.

Voir aussi `jump` et `can-move?`

## fput

`fput` *item list*

Ajoute l'élément *item* au début de la liste *list* et retourne la nouvelle liste résultant de cette manipulation.

```
;; admettons que ma-liste soit [5 7 10]
set ma-liste fput 2 ma-liste
;; ma-liste est maintenant [2 5 7 10]
```

# G

## globals

```
globals [var1 ...]
```

Ce mot-clé, tout comme les mots-clés `breed`, `<races>-own`, `patches-own` et `turtles-own`, ne peut être utilisé qu'au début du code d'un programme, avant les définitions des procédures. Il définit une nouvelle variable globale. Les variables globales sont « globales » parce qu'elles sont accessibles par tous les agents et peuvent être utilisées n'importe où dans le modèle.

Le plus souvent, `globals` est utilisé pour définir des variables ou des constantes qui doivent pouvoir être utilisées dans plusieurs parties du programme. Il demande une entrée sous forme de liste dont les éléments sont les variables ou les constantes à définir.

# H

## hatch
## hatch-*<breeds>*

```
hatch number [ commands ]
hatch-<breeds> number [ commands ]
```


La tortue concernée crée le nombre de nouvelles tortues spécifié dans *number*. Chaque nouvelle tortue est identique et se trouve au même endroit que son parent. Les nouvelles tortues exécutent ensuite les commandes de la liste *commands*. Vous pouvez utiliser ces commandes pour donner aux nouvelles tortues des couleurs, des orientations, des emplacements différents ou quoi que ce soit d'autre. Les nouvelles tortues sont d'abord toutes créées puis elles exécutent, chacune à leur tour (choisi au hasard) les commandes qui leur sont passées.

Si la variante `hatch-<breeds>` est utilisée, les nouvelles tortues qui sont créées appartiennent à la race *breeds* spécifiée. Sinon, les nouvelles tortues sont de la même race que leur parent.

**Note** : pendant que les commandes spécifiées dans la liste sont exécutées, aucun autre agent n'a l'autorisation d'exécuter quelque code que ce soit (à l'image de ce qui se passe avec la commande `without-interruption`). Cette précaution garantit que si la primitive `ask-concurrent` est utilisée, les nouvelles tortues ne puissent pas interagir avec n'importe quel autre agent avant qu'elles n'aient toutes été complètement initialisées.

```
hatch 1 [ lt 45 fd 1 ]
;; cette tortue crée une nouvelle tortue
;; qui tourne à gauche de 45° et avance d'un pas
hatch-sheep 1 [ set color black ]
;; cette tortue crée une nouvelle tortue de race
;; sheep et de couleur noire
```

Voir aussi `create-turtles` et `sprout`

## heading

```
heading
```


`heading` est une variable tortue préprogrammée. Elle spécifie la direction (le cap) de la tortue. Sa valeur est toujours un nombre plus grand ou égal à zéro et plus petit que 360. 0 indique le nord, 90 l'est, etc, comme en géographie. Vous pouvez modifier cette variable pour faire tourner (pivoter) la tortue sur elle-même.

Voir aussi `right`, `left`, `dx` et `dy`

Exemple:

```
set heading 45 ;; la tortue fait maintenant face au nord-est
set heading heading + 10 ;; a le même effet que "rt 10"
```

## hidden?

```
hidden?
```


`hidden?` est une variable tortue ou une variable lien préprogrammée. Elle contient une valeur booléenne, `true` (vrai) ou `false` (faux), indiquant si la tortue ou le lien est actuellement caché (c'est-à-dire invisible). Vous pouvez modifier la valeur de cette variable pour faire disparaître ou apparaître la tortue ou le lien.

Voir aussi `hide-turtle`, `show-turtle`, `hide-link` et `show-link`

Exemple :

```
set hidden? not hidden?
;; si la tortue est visible, elle se cache,
;; et si la tortue était cachée, elle réapparaît
```

## hide-link

```
hide-link
```


Rend le lien invisible.

**Note** : cette commande a le même effet que de donner la valeur `true` (vrai) à la variable lien `hidden?`.

Voir aussi `show-link`.

## hide-turtle
## ht

```
hide-turtle
```


La tortue devient invisible.

**Note**: cette commande a le même effet que de donner la valeur `true` (vrai) à la variable tortue `hidden?`.

Voir aussi `show-turtle`.

## histogram

```
histogram list
```

Construit un histogramme à partir des valeurs contenues dans la liste de données *`list`*.

L'histogramme montre la distribution (la fréquence) des valeurs de la liste passée en paramètre. Les hauteurs des barres de l'histogramme représentent le nombre de valeurs pour chaque sous-ensemble de valeurs.

Toute valeur non-numérique de la liste est ignorée.

L'histogramme est dessiné dans le traceur de courbes courant en utilisant le crayon et la couleur courants. Utilisez la commande `set-plot-x-range` pour contrôler la plage des valeurs à prendre en compte pour le dessin de l'histogramme et spécifiez l'intervalle, soit directement avec `set-plot-pen-interval`, soit indirectement avec `set-histogram-num-bars`, pour déterminer combien de barres doivent être dessinées pour représenter les valeurs.

Avant de commencer le dessin de l'histogramme, le traceur efface tout point préalablement dessiné par le crayon courant.

Assurez-vous que le crayon courant soit bien en mode barre (mode 1) si vous voulez que l'histogramme soit dessiné avec des barres.

Pour des raisons de représentation graphique de l'histogramme, la plage des valeurs X du traceur ne contient pas la plus grande valeur X. Les valeurs égales au plus grand X tombent hors de la plage des valeurs de l'histogramme.

```
histogram [color] of turtles
;; dessine un histogramme montrant le nombre
;; de tortues de chaque couleur
```

## home

`home`

Les tortues concernées se déplacent à l'origine du monde (0,0). Équivalent à `setxy 0 0`.

## hsb

`hsb` *`hue saturation brightness`*

Ce reporter retourne une liste formée de trois nombres spécifiant les valeurs RGB de la couleur qui lui a été donnée dans le format HSB. Les valeurs données en entrée *`hue`* (teinte), *`saturation`*, *`brightness`* (brillance) sont des nombres entiers dans la plage 0-255. La liste RGB retournée contient aussi trois entiers appartenant à la même plage.

Voir aussi `rgb`

## hubnet-broadcast

`hubnet-broadcast` *`tag-name value`*

Cette primitive envoie les données de *`value`* du NetLogo fonctionnant en tant que serveur à la variable *`tag-name`* (dans le cas de "Calculator HubNet") ou à l'élément d'interface portant le nom *`tag-name`* (dans le cas de "Computer HubNet") de tous les clients.

Voir le Guide de programmation HubNet pour les détails.

## hubnet-broadcast-view

`hubnet-broadcast-view`

Cette primitive envoie l'état courant de la vue 2D du modèle NetLogo tournant sur le serveur à tous les clients "Computer HubNet". Ne fait rien avec "Calculator HubNet".

**Note** : c'est une primitive expérimentale dont le comportement pourrait changer dans une version future.

Voir le Guide de programmation HubNet pour les détails.

## hubnet-enter-message?

`hubnet-enter-message?`

Retourne `true` (vrai) si un nouveau client "Computer HubNet" vient d'entrer dans la simulation. Sinon retourne `false` (faux). La primitive `hubnet-message-source` fournit le nom d'utilisateur du client qui vient de se connecter.

Voir le Guide de programmation HubNet pour les détails.

### hubnet-exit-message?

```
hubnet-exit-message?
```

Retourne `true` (vrai) si un client "Computer HubNet" vient de quitter la simulation, sinon retourne `false` (faux). La primitive `hubnet-message-source` fournit le nom du client qui vient de se déconnecter.

Voir le Guide de programmation HubNet pour les détails.

### hubnet-fetch-message

```
hubnet-fetch-message
```

S'il y a (encore) de nouvelles données qui ont été envoyées par les clients, cette primitive récupère le prochain paquet de données de manière à ce que ces données puissent être utilisées par `hubnet-message`, `hubnet-message-source` et `hubnet-message-tag`. Une erreur est générée s'il n'y a plus de nouvelles données venant des clients.

Voir le Guide de programmation HubNet pour les détails.

### hubnet-message

```
hubnet-message
```

Retourne le message récupéré par le reporter `hubnet-fetch-message`.

Voir le Guide de programmation HubNet pour les détails.

### hubnet-message-source

```
hubnet-message-source
```

Retourne le nom du client qui a envoyé le message, message récupéré par la primitive `hubnet-fetch-message`.

Voir le Guide de programmation HubNet pour les détails.

### hubnet-message-tag

```
hubnet-message-tag
```

Retourne le tag (marqueur, identificateur) associé aux données récupérées par la primitive `hubnet-fetch-message`. Avec "Calculator HubNet", ce reporter retourne un des noms de variables spécifiés par la primitive `hubnet-set-client-interface`. Avec "Computer HubNet", ce reporter retourne le nom d'affichage de l'un des éléments de l'interface du client.

Voir le Guide de programmation HubNet pour les détails.

### hubnet-message-waiting?

```
hubnet-message-waiting?
```

Cette primitive regarde s'il y a un nouveau message envoyé par un client. Retourne `true` (vrai) s'il y en a un et `false` (faux) s'il n'y en a pas.

Voir le Guide de programmation HubNet pour les détails.

## hubnet-reset

```
hubnet-reset
```

Met en route le système "HubNet". "HubNet" doit être en fonction pour pouvoir utiliser l'une des autres primitives "HubNet", à l'exception de `hubnet-set-client-interface`.

Voir le Guide de programmation HubNet pour les détails.

## hubnet-send

```
hubnet-send string tag-name value
hubnet-send list-of-strings tag-name value
```

Pour "Calculator HubNet", cette primitive fonctionne exactement de la même manière que la primitive `hubnet-broadcast`. (Nous prévoyons de changer ce comportement dans une version future de NetLogo.)

Pour "Computer HubNet", elle fonctionne de la manière suivante :

Avec une chaîne de caractères comme première entrée, elle envoie les données *value* du serveur NetLogo à l'élément nommé *tag-name* placé sur l'interface du client dont le nom d'utilisateur est spécifié par *string*.

Avec une liste de chaînes comme première entrée, cette primitive envoie les données *value* du serveur NetLogo à l'élément nommé *tag-name* placé sur l'interface de tous les clients dont les noms d'utilisateur se trouvent dans l'entrée *list-of-strings*.

L'envoi d'un message avec `hubnet-send` à un client qui n'existe pas génère un message `hubnet-exit-message`.

Voir le Guide de programmation HubNet pour les détails.

## hubnet-send-view

```
hubnet-send-view string
hubnet-send-view list-of-strings
```

Cette primitive ne fait rien avec "Calculator HubNet".

Avec "Computer HubNet", elle fonctionne de la manière suivante :

Avec une chaîne de caractères comme première entrée, elle envoie l'état courant (l'image) de la Vue 2D du serveur NetLogo au client "Computer HubNet" dont le nom d'utilisateur est spécifié par *string*. Cette action permet d'actualiser la Vue 2D du client afin que l'image qu'elle affiche soit identique à celle du serveur.

Avec une liste de chaînes comme première entrée, elle envoie l'état courant (l'image) de la Vue 2D du serveur NetLogo à tous les clients "Computer HubNet" dont les noms d'utilisateur se trouvent dans l'entrée *list-of-strings*. Cette action permet d'actualiser les Vues 2D des clients afin que l'image qu'elles affichent soit identiques à celle du serveur.

L'envoi de la vue 2D avec `hubnet-send-view` à un client qui n'existe pas génère un message `hubnet-exit-message`.

**Note**: c'est une primitive expérimentale dont le comportement pourrait changer dans une version future.

Voir le Guide de programmation HubNet pour les détails.

### hubnet-set-client-interface

`hubnet-set-client-interface` *`client-type client-info`*

Si l'entrée *`client-type`* est `"COMPUTER"`, *`client-info`* doit être une liste vide avec "Computer HubNet".

```
hubnet-set-client-interface "COMPUTER"[]
```

Des versions futures de "HubNet" supporteronT d'autres types de clients. Même pour "Computer HubNet", la signification du second paramètre de cette commande pourrait changer par la suite.

Voir le Guide de programmation HubNet pour les détails.

# I

## if

```
if condition [ commands ]
```

Cette primitive fait exécuter les commandes placées dans la liste [*commands*] si la *condition* testée est vraie. Le reporter *condition* doit retourner une valeur booléenne `true` ou `false` (vrai ou faux).

Le reporter peut retourner une valeur différente pour des agents différents, ce qui fait que certains agents exécutent les commandes et d'autres non.

```
if xcor > 0[ set color blue ]
;; les tortues placées dans la moitié gauche
;; du monde sont colorées en bleu
```

Voir aussi `ifelse` et `ifelse-value`

## ifelse

```
ifelse reporter [ commands1 ] [ commands2 ]
```

La primitive `ifelse` fait exécuter les commandes listées dans [*commands1*] et ignore les commandes listées dans [*commands2*] si la condition *reporter* retourne `true` (vrai) et ignore les commandes listées dans [*commands1*] mais fait exécuter les commandes listées dans [*commands2*] si la condition *reporter* retourne `false` (faux).

Le reporter *condition* doit retourner une valeur booléenne `true` ou `false` (vrai ou faux).

Le reporter peut retourner une valeur différente pour des agents différents, ce qui fait que certains agents exécutent les commandes de la liste [*commands1*] alors que d'autres exécutent celles de la liste [*commands2*].

```
ask patches
  [ ifelse pxcor > 0
      [ set pcolor blue ]
      [ set pcolor red ] ]
;; la moitié droite du monde devient bleue et
;; la moitié gauche devient rouge
```

Voir aussi `if` et `ifelse-value`

## ifelse-value

```
ifelse-value reporter [reporter1] [reporter2]
```

La primitive `ifelse-value` retourne la valeur fournie par *reporter1* et ignore celle fournie par *reporter2* si la condition *reporter* est vraie (`true`), mais retourne la valeur fournie par *reporter2* et ignore celle fournie par *reporter1* si la condition *reporter* est fausse (`false`).

La condition *reporter* doit retourner une valeur booléenne `true` ou `false` (vrai ou faux).

Cette primitive peut être utilisée quand une expression conditionnelle est nécessaire dans un contexte de reporter où des commandes (telles que `ifelse`) ne sont pas autorisées.

```
ask patches
[
  set pcolor ifelse-value (pxcor > 0) [blue] [red]
]
```

```
;; la moitié gauche du monde devient rouge et
;; la moitié droite bleue
show n-values 10 [ifelse-value (? < 5) [0] [1]]
=> [0 0 0 0 0 1 1 1 1 1]
show reduce [ifelse-value (?1 > ?2) [?1] [?2]]
  [1 3 2 5 3 8 3 2 1]
=> 8
```

Voir aussi `if` et `ifelse`

## import-drawing

```
import-drawing filename
```


Cette primitive lit un fichier image *filename* et l'affiche dans la couche dessin de la Vue (celle sur laquelle les tortues peuvent dessiner) en ajustant sa taille à celle du monde NetLogo tout en conservant les proportions (largeur/hauteur) de l'image originale. L'image est centrée dans la Vue. L'ancien dessin n'est pas effacé en premier.

Les agents ne peuvent « ausculter » l'image, ce qui fait qu'ils ne peuvent interagir avec ou manipuler les images importées par `import-drawing`. Si des agents doivent pouvoir « ausculter » une image, il faut utiliser `import-pcolors` ou `import-pcolors-rgb`.

Les formats de fichiers image supportés sont les suivants : BMP, JPG, GIF et PNG. Si le format de l'image supporte la transparence (alpha), cette information est aussi importée.

## import-pcolors

```
import-pcolors filename
```


La primitive `import-pcolors` lit le fichier image nommé *filename*, ajuste sa taille à celle de la grille de patches tout en conservant les proportions (largeur/hauteur) de l'image originale et transfère les couleurs résultantes aux pixels des patches. L'image est centrée sur la grille des patches. Les nouvelles couleurs des patches peuvent être faussées puisque l'espace des couleurs NetLogo ne contient pas toutes les couleurs possibles. (Voir le chapitre concernant les couleurs du Guide de la programmation.) Le travail de cette primitive peut prendre du temps pour certaines images, en particulier quand le nombre de patches est élevé, que l'image importée est grande et qu'elle comporte de nombreuses couleurs.

Puisque la primitive `import-pcolors` modifie la valeur de la variable `pcolor` des patches, les agents peuvent « ausculter » l'image, autrement dit, lire la valeur de cette variable et réagir en conséquence. Ce qui est utile si des agents doivent analyser, manipuler ou interagir d'une quelconque manière avec l'image. Si l'affichage d'un arrière-plan statique et sans distorsions de couleurs suffit, utilisez plutôt la primitive `import-drawing`.

Les formats de fichiers image supportés sont les suivants : BMP, JPG, GIF, and PNG. Si le format de l'image supporte la transparence (alpha), tous les pixels totalement transparents sont ignorés. (Les pixels partiellement transparents sont considérés comme opaques.)

## import-pcolors-rgb

```
import-pcolors-rgb filename
```


Cette primitive lit le fichier image nommé *filename*, ajuste la taille de l'image à celle de la grille de patches tout en conservant les proportions (largeur/hauteur) de l'image originale et transfère les couleurs résultantes aux pixels des patches. L'image est centrée sur la grille des patches. Contrairement à ce qui se passe

avec la primitive `import-pcolors`, ce sont les couleurs exactes de l'image originale qui sont retenues. La variable `pcolor` de tous les patches contiendra alors une liste de valeurs RGB plutôt qu'un nombre désignant une couleur NetLogo (approximative).

Les formats de fichiers image supportés sont les suivants : BMP, JPG, GIF, and PNG. Si le format de l'image supporte la transparence (alpha), tous les pixels totalement transparents sont ignorés. (Les pixels partiellement transparents sont considérés comme opaques.)

## import-world

`import-world` *`filename`*

`import-world` lit les valeurs de toutes les variables d'un modèle, aussi bien celles des variables préprogrammées que celles définies par l'utilisateur, y compris toutes les variables de type observateur, tortue et patch dans le fichier externe désigné par *`filename`*. Ce fichier doit être au format utilisé par la primitive `export-world`.

Notez que la fonctionnalité de cette primitive peut aussi être obtenue directement dans le menu "File" de NetLogo.

Pour éviter des erreurs lors d'une importation de données avec `import-world`, exécutez les opérations suivantes dans l'ordre indiqué :

1. Ouvrez le modèle avec lequel vous avez créé le fichier exporté.
2. Pressez le bouton d'initialisation, généralement nommé "Setup", pour mettre le modèle dans un état qui lui permette de faire tourner la simulation.
3. Importez le fichier de données avec `import-world`.
4. Ré-ouvrez tout fichier que le modèle avait ouvert avec la commande `file-open`.
5. Si vous le voulez, pressez le bouton "Go" pour continuer la simulation à partir de l'endroit où vous l'aviez stoppée.

Si vous voulez importer un fichier résidant dans un autre dossier que celui dans lequel se trouve le modèle, vous devez transmettre à la commande tout le chemin du fichier (chemin absolu) que vous voulez importer. Voir `export-world` pour un exemple.

## in-cone

*`agentset`* `in-cone` *`distance angle`*

Ce reporter permet de doter une tortue d'un « cône de vision » vers l'avant. Le cône est défini par les deux entrées *`distance`* de vision (rayon) et *`angle`* de vision, cette dernière valeur représentant l'ouverture du cône. L'angle de vision peut prendre des valeurs de 0 à 360 et est centré sur le cap courant de la tortue. (Si l'angle est de 360, alors `in-cone` est équivalent à `in-radius`.)

Le reporter `in-cone` retourne un ensemble d'agents *`agentset`* ne contenant que les agents se trouvant dans le cône et appartenant au même ensemble d'agents que l'agent appelant. (L'agent appelant peut aussi en faire partie.)

La distance à un patch est mesurée à partir du centre de patch.

```
ask turtles
[ ask patches in-cone 3 60
  [ set pcolor red ]
]
;; chaque tortue fait une «tache» de patches rouges dans
;; un cône de 60 degrés et jusqu'à 3 unités devant elle
```

## in-<breed>-neighbor?<br>in-link-neighbor?

```
in-<breed>-neighbor? agent
in-link-neighbor? turtle
```

Retourne `true` (vrai) s'il y a un lien orienté partant de la tortue spécifiée *`turtle`* et allant à l'agent appelant.

```
crt 2
ask turtle 0
[
  create-link-to turtle 1
  show in-link-neighbor? turtle 1 ;; affiche false
  show out-link-neighbor? turtle 1 ;; affiche true
]
ask turtle 1
[
   show in-link-neighbor? turtle 0 ;; affiche true
   show out-link-neighbor? turtle 0 ;; affiche false
]
```

## in-<breed>-neighbors<br>in-link-neighbors

```
in-<breed>-neighbors
in-link-neighbors
```

Retourne l'ensemble d'agents formé de toutes les tortues qui ont des liens orientés venant l'appelant.

```
crt 4
ask turtle 0 [ create-links-to other turtles ]
ask turtle 1 [ ask in-link-neighbors [ set color blue ] ]
;; la tortue 0 devient bleue
```

## in-<breed>-from<br>in-link-from

```
in-<breed>-from turtle
in-link-from turtle
```

Retourne le lien partant de la *`tortue`* pour aller à l'agent appelant. Retourne `nobody` s'il n'y a pas un tel lien.

```
crt 2
ask turtle 0 [ create-link-to turtle 1 ]
ask turtle 1 [ show in-link-from turtle 0 ] ;; affiche link 0 1
ask turtle 0 [ show in-link-from turtle 1 ] ;; affiche nobody
```

## __includes

```
__includes [ filename ... ]
```

Cette primitive (expérimentale) provoque l'inclusion dans le modèle courant des fichiers source NetLogo externes (portant le suffixe `.nls`) listés dans l'entrée `[`*`filename`*`...]`. Les fichiers d'inclusions peuvent contenir des races, des variables et des définitions de procédures. `__includes` ne peut être utilisé qu'une seule fois par fichier.

## in-radius

*agentset* in-radius *number*

Retourne un ensemble d'agents formé des agents, appartenant au même ensemble d'agents que l'agent appelant, dont la distance à l'appelant est inférieure ou égale à *number*. (L'agent appelant peut aussi en faire partie.)

La distance jusqu'à un patch ou la distance à partir d'un patch est mesurée à partir du centre du patch.

```
ask turtles
[ ask patches in-radius 3
    [ set pcolor red ]
]
;; chaque tortue fait une tache rouge autour d'elle.
```

## inspect

inspect *agent*

Ouvre un moniteur d'agent pour l'*agent* donné (une tortue ou un patch).

```
inspect patch 2 4
;; un moniteur d'agent s'ouvre pour ce patch
inspect one-of moutons
;; un moniteur d'agent s'ouvre pour une tortue
;; choisie au hasard dans la race des "moutons"
```

## int

int *number*

Retourne la partie entière de *nombre*, la partie fractionnaire étant simplement supprimée.

```
show int 4.7
=> 4
show int -3.5
=> -3
```

### is-agent?
### is-agentset?
### is-boolean?
### is-*<breed>*?
### is-directed-link?
### is-link?
### is-link-set?
### is-list?
### is-number?
### is-patch?
### is-patch-set?
### is-string?
### is-turtle?
### is-turtle-set?
### is-undirected-link?

```
is-agent? value
is-agentset? value
is-boolean? value
is-<breed>? value
is-directed-link? value
is-link? value
is-link-set? value
is-list? value
is-number? value
is-patch? value
is-patch-set? value
is-string? value
is-turtle? value
is-turtle-set? value
is-directed-link? value
```

Ces reporters retournent `true` (vrai) si les données de l'entrée *`value`* sont du type indiqué par le nom du reporter, sinon retourne `false` (false).

### item

```
item index list
item index string
```

Appelé avec une liste, ce reporter retourne la valeur de l'élément de numéro *`index`* appartenant à la liste *`list`* donnée.

Appelé avec une chaîne de caractères, ce reporter retourne le caractère de numéro *`index`* de la chaîne de caractères *`string`* donnée en entrée.

Notez que les numéros des éléments (*`index`*) commencent à 0 et non à 1 (Le premier élément porte le numéro 0, le deuxième porte le numéro 1, et ainsi de suite.)

```
;; suppose mylist is [2 4 6 8 10]
show item 2 mylist
=> 6
show item 3 "my-shoe"
=> "s"
```

# J

## jump

```
jump number
```


La tortue avance d'un seul coup du nombre d'unités spécifié par l'entrée *number* (plutôt que par pas d'une unité à la fois comme avec la commande `forward` ou son abréviation `fd`).

Si la tortue ne peut avancer du nombre d'unités demandé parce que la topologie du monde NetLogo ne le permet pas, la tortue ne se déplace pas du tout.

Voir aussi `forward` et `can-move?`

# L

## label

```
label
```


C'est une variable tortue ou lien préprogrammée. Elle peut contenir une valeur de n'importe quel type. Dans la Vue, la tortue ou le lien est accompagné d'une étiquette affichant (sous forme de texte) la valeur de cette variable. Cette variable permet d'ajouter, de modifier ou d'enlever une étiquette de tortue ou de lien.

Exemple :

```
ask turtles [ set label who ]
;; toutes les tortues ont maintenant une étiquette
;; avec leur numéro d'identification
ask turtles [ set label "" ]
;; toutes les tortues n'ont maintenant plus d'étiquette
```

Voir aussi `label-color`, `plabel` et `plabel-color`

## label-color

```
label-color
```


C'est une variable tortue ou lien préprogrammée. Elle contient un nombre plus grand ou égal à zéro et plus petit que 140 qui spécifie la couleur de l'étiquette de la tortue ou du lien (si l'étiquette est visible). Vous pouvez spécifier la valeur de cette variable pour modifier la couleur de l'étiquette des tortues ou des liens.

Exemple :

```
ask turtles [ set label-color red ]
;; toutes les tortues ont maintenant des étiquettes rouges
```

Voir aussi `label`, `plabel` et `plabel-color`

## last

```
last list
last string
```

Appelé avec une liste, ce reporter retourne le dernier élément de la liste *`list`* passée en entrée.

Appelé avec une chaîne de caractères, ce reporter retourne une chaîne d'un seul caractère ne contenant que le dernier caractère de la chaîne *`string`* passée en entrée.

## layout-circle

```
layout-circle agentset radius
layout-circle list-of-turtles radius
```

Dispose les tortues spécifiées par la première entrée sur un cercle de rayon *`radius`*, cercle centré sur le patch placé au centre du monde. (Si le monde a une dimension paire, le centre du cercle est arrondi vers le bas vers le patch le plus proche.) Les tortues pointent vers l'extérieur (le long des rayons).

Si la première entrée est un ensemble d'agents *`agentset`*, les tortues sont disposées dans un ordre aléatoire.

Si la première entrée est une liste *list-of-turtles*, les tortues sont arrangées, en suivant l'ordre donné par la liste, dans le sens des aiguilles d'une montre en commençant au sommet du cercle. (Tout ce qui, dans la liste, n'est pas tortue est ignoré.)

```
;; en ordre aléatoire
layout-circle turtles 10
;; en suivant l'ordre des numéro d'identification
layout-circle sort turtles 10
;; en fonction de la taille
layout-circle sort-by [[size] of ?1 < [size] of ?2] turtles 10
```

## __layout-magspring

`__layout-magspring` *turtle-set link-set spring-constant spring-length repulsion-constant magnetic-field-strength magnetic-field-type bidirectional?*

Ressemblant beaucoup à la primitive `layout-spring`, `__layout-magspring` possède un niveau de complexité plus élevé. Les tortues de l'ensemble de tortues *turtle-set* s'attirent et se repoussent en fonction des liens, spécifiés dans *link-set*, qui les unissent, mais il y a en plus un champ magnétique sur les lignes duquel les liens tentent de s'aligner.

L'entrée *link-set* est l'ensemble des liens qui exercent une force sur les tortues qu'ils relient. Les tortues connectées par des liens de l'ensemble *link-set* mais n'appartenant pas à l'ensemble *turtle-set* sont traitées comme des points d'ancrage. S'il n'y a pas de tortues avec des positions fixes (ancrées), le réseau tout entier se refermera probablement sur lui-même.

*spring-constant* spécifie la rigidité du ressort (constante de rappel du ressort). (Voir `layout-spring`).

*spring-length* spécifie la longueur des ressorts au repos ou leur longueur naturelle. (Voir `layout-spring`).

*repulsion-constant* spécifie la grandeur de la force de répulsion exercée entre les noeuds. (Voir `layout-spring`).

L'entrée *magnetic-field-strength* spécifie la force du champ magnétique. (Les valeurs raisonnables vont de 0 à 1, mais 0.05 est une bonne valeur par défaut.)

L'entrée *magnetic-field-type* est un nombre de 0 à 10 spécifiant l'orientation du champ magnétique. Les diverses possibilités sont énumérées dans le tableau ci-dessous :

| *magnetic-field-type* | Description |
|---|---|
| NONE = 0 | Si aucun champ n'est utilisé, la commande fonctionne comme la primitive `layout-spring`. |
| NORTH = 1 | Le champ magnétique est orienté vers le Nord. |
| NORTHEAST = 2 | Le champ magnétique est orienté vers le Nord-Est. |
| EAST = 3 | Le champ magnétique est orienté vers l'Est. |
| SOUTHEAST= 4 | Le champ magnétique est orienté vers le Sud-Est. |
| SOUTH = 5 | Le champ magnétique est orienté vers le Sud. |
| SOUTHWEST= 6 | Le champ magnétique est orienté vers le Sud-Ouest. |
| WEST = 7 | Le champ magnétique est orienté vers l'Ouest. |
| NORTHWEST = 8 | Le champ magnétique est orienté vers le Nord-Ouest. |
| POLAR = 9 | Le champ magnétique rayonne dans toutes les directions à partir de l'origine (comme les méridiens à partir du pôle). |
| CONCENTRIC = 10 | Le champ magnétique tourne dans sens des aiguilles d'une montre en cercles concentriques autour de l'origine (comme les parallèles autour du pôle). |

Si l'entrée *bidirectional?* vaut `true` (vrai), alors les liens essaient de s'aligner sur le champ magnétique en poussant les tortues qu'ils relient aussi bien dans la direction du champ que dans la direction opposée. Dans le cas contraire, les liens ne poussent que dans une direction.

```
to make-a-tree
  set-default-shape turtles "circle"
  crt 5
  ask turtle 0
  [
    create-link-with turtle 1
    create-link-with turtle 2
  ]
  ask turtle 1
  [
    create-link-with turtle 3
    create-link-with turtle 4
  ]
  ;; disposition avec une champ magnétique SUD assez fort
  repeat 50
  [
    __layout-magspring
    turtles with [who != 0] links 0.3 4 1 .50 5 false
  ]
end
```

## layout-radial

`layout-radial` *turtle-set link-set root-agent*

Place les tortues de l'ensemble *turtle-set* connectées par les liens de l'ensemble de liens *link-set* sur un arbre radial centré autour de l'agent-racine *root-agent* qui est d'abord déplacé au centre du monde NetLogo affiché dans la Vue.

Seuls les liens de l'ensemble *link-set* sont pris en compte pour déterminer la forme du réseau. Si des liens sont connectés à des tortues n'appartenant pas à l'ensemble *turtle-set*, ces tortues restent où elles sont.

Cette primitive fonctionnera même si le réseau contient des cycles et n'a pas une structure en arbre, mais le résultat ne sera pas toujours bien joli.

```
to make-a-tree
  set-default-shape turtles "circle"
  crt 6
  ask turtle 0
  [
    create-link-with turtle 1
    create-link-with turtle 2
    create-link-with turtle 3
  ]
  ask turtle 1
  [
    create-link-with turtle 4
    create-link-with turtle 5
  ]
  ;; construit un arbre radial centré sur la tortue 0
  layout-radial turtles links (turtle 0)
end
```

## layout-spring

`layout-spring` *`turtle-set link-set spring-constant spring-length repulsion-constant`*

Place les tortues de l'ensemble *`turtle-set`* comme si les liens de l'ensemble *`link-set`* qui les relient étaient des ressorts et que les tortues se repoussaient les unes les autres. Les tortues connectées par les liens de l'ensemble *`link-set`* mais n'appartenant pas à l'ensemble *`turtle-set`* sont traitées comme des points d'ancrage et ne sont pas déplacées.

L'entrée *`spring-constant`* définit la mesure de la « raideur » du ressort (appelée en physique constante de rappel du ressort ou constante d'élasticité). Il s'agit en fait de la « résistance » du ressort à toute modification de sa longueur. *`spring-constant`* est la force qu'exercerait le ressort si sa longueur était modifiée d'une unité.

L'entrée *`spring-length`* définit la longueur du ressort « force-zéro », autrement dit la longueur du ressort au repos ou sa longueur « naturelle ». C'est la longueur que tous les ressorts tentent d'atteindre soit en tirant, soit en poussant les noeuds auxquels ils sont attachés.

L'entrée *`repulsion-constant`* définit la mesure de la force de répulsion entre deux noeuds. C'est la force que deux noeuds séparés d'une distance d'une unité exercent l'un sur l'autre.

La force de répulsion tente d'éloigner le plus possible les noeuds les uns des autres de manière à éviter l'entassement et l'effet de ressort tente de les retenir « à peu près » à une certaine distance des noeuds auxquels ils sont connectés. Le résultat est un arrangement des noeuds de tout le réseau de manière à ce que cet arrangement puisse mettre en évidence les relations entre les noeuds tout en étant le moins possible entassé et en offrant un aspect visuel plaisant.

L'algorithme de construction du réseau est basé sur celui de Fruchterman-Reingold. Des informations supplémentaires peuvent être obtenues à cette adresse : http://citeseer.ist.psu.edu/fruchterman91graph.html

```
to make-a-triangle
  set-default-shape turtles "circle"
  crt 3
  ask turtle 0
 [
    create-links-with other turtles
  ]
  ask turtle 1
  [
    create-link-with turtle 2
  ]
  repeat 30 [ layout-spring turtles links 0.2 5 1 ]
  ;; place les noeuds dans un triangle
end
```

## layout-tutte

`layout-tutte` *`turtle-set link-set radius`*

Les tortues connectées par les liens de l'ensemble *`link-set`* mais n'appartenant pas à l'ensemble *`turtle-set`* sont disposées sur un cercle de rayon spécifié par l'entrée *`radius`*. Cet ensemble d'agents doit contenir au moins trois tortues qui serviront de points d'ancrage.

Les tortues de l'ensemble *`turtle-set`* sont alors placées de la manière suivante : chaque tortue est placée au « centre de gravité » (barycentre) du polygone formé par les tortues voisines qui lui sont liées. (Ce « centre de gravité » est en quelque sorte la moyenne bidimentionnelle des coordonnées de tortues voisines liées.)

La raison d'être du cercle « d'agents ancrés » est d'empêcher que toutes les tortues s'agglutinent en un seul point.

Après quelques itérations du processus, la position des tortues (l'aspect du réseau) se stabilise.

Ce type de réseau a été nommé d'après le mathématicien William Thomas Tutte qui l'a proposé comme méthode de dessin de graphe.

```
to make-a-tree
  set-default-shape turtles "circle"
  crt 6
  ask turtle 0
  [
    create-link-with turtle 1
    create-link-with turtle 2
    create-link-with turtle 3
  ]
  ask turtle 1
  [
    create-link-with turtle 4
    create-link-with turtle 5
  ]

  ;; place toutes les tortues qui n'ont qu'une voisine
  ;; sur le périmètre d'un cercle, puis place les tortues
  ;; restantes dans ce cercle, réparties entre leurs voisines.
  repeat 10
  [
    layout-tutte turtles (turtles with [count link-neighbors = 1]) 12
  ]
end
```

## left
## lt

```
left number
```


Fait pivoter la tortue vers la gauche de *number* degrés. Si *number* est négatif, la tortue pivote à droite.

## length

```
length list
length string
```

Retourne le nombre d'éléments de la liste *list* donnée, ou le nombre de caractères de la chaîne *string* donnée.

## let

```
let variable value
```

Crée une nouvelle variable locale dont le nom est spécifié par l'entrée *variable* et lui assigne la valeur spécifiée dans l'entrée *value*. Une variable locale n'existe (n'est visible) que dans le bloc de commandes délimité par des crochets dans lequel elle a été définie.

Si vous voulez modifier cette valeur par la suite, il faut utiliser la commande `set`.

Exemple :

```
let prey one-of sheep-here
if prey != nobody
[
  ask prey [ die ]
]
```

## link

```
link end1 end2
link <breed> end1 end2
```

Retourne le lien qui connecte les tortues marquant ses deux extrémités, tortues spécifiées par leur numéro d'identification *end1* et *end1*. Si un tel lien n'existe pas, retourne `nobody`. Pour faire référence à des liens de race, vous devez utiliser la forme singulier du nom de la race.

```
ask link 0 1 [ set color green ]
;; le lien sans race connectant la tortue 1 à la tortue 2 devient vert
ask directed-link 0 1 [ set color red ]
;; le lien orienté connectant la tortue 1 à la tortue 2 devient rouge
```

Voir aussi `patch-at`

## link-heading

```
link-heading
```


Retourne le cap (l'orientation) en degrés (au moins 0 et inférieur à 360) du lien, en partant de la tortue *end1* vers la tortue *end2*. Génère une erreur d'exécution si les deux extrémités sont au même endroit.

```
ask link 0 1 [ print link-heading ]
;; affiche [[towards other-end] of end1] of link 0 1
```

Voir aussi `link-length`

## link-length

```
link-length
```


Retourne la distance séparant les deux extrémités du lien.

```
ask link 0 1 [ print link-length ]
;; affiche [[distance other-end] of end1] of link 0 1
```

Voir aussi `link-heading`

## link-set

```
link-set value
(link-set value1 value2 ...)
```

Retourne un ensemble d'agents contenant tous les liens se trouvant n'importe où dans n'importe laquelle des entrées. Les entrées peuvent être des liens individuels, des ensembles de liens, `nobody` ou des listes (ou des listes imbriquées) contenant n'importe quels objets cités ci-dessus.

```
link-set self

link-set [my-links] of nodes with [color = red]
```

Voir aussi `turtle-set` et `patch-set`

## link-shapes

```
link-shapes
```

Retourne une liste de chaînes de caractères contenant les noms de toutes les formes de liens du modèle.

De nouvelles formes (shapes) peuvent être créées ou importées d'autres modèles en utilisant L'éditeur de formes de liens.

```
show link-shapes
=> ["default"]
```

## links

```
links
```

Retourne un ensemble d'agents constitué de tous les liens.

```
show count links

;; affiche le nombre de liens
```

## links-own
## *<link-breeds>*-own

```
links-own [var1 ...]
<link-breeds>-own [var1 ...]
```

Le mot-clé `links-own`, tout comme les mots-clés `globals`, `breed`, `<races>-own`, `turtles-own` et `patches-own`, ne peut être défini qu'au début du programme, avant toute définition de procédures. Il permet de définir les variables appartenant à chaque lien qui sera créé par la suite.

Si vous spécifiez une race de liens à la place de *`links`* , seuls les liens de cette race disposeront des variables créées. (La même variable peut être partagée par plus d'une race.)

```
undirected-link-breed [sidewalks sidewalk]
directed-link-breed [streets street]
links-own [traffic]   ;;  s'applique à toutes les races
sidewalks-own [pedestrians]
streets-own [cars bikes]
```

## list

```
list value1 value2
(list value1 ...)
```

Retourne une liste formée des éléments *`value`* donnés. Les éléments peuvent être de n'importe quel type, retournés par n'importe quel type de reporter.

```
show list (random 10) (random 10)
=> [4 9]  ;; ou une liste similaire
show (list 5)
=> [5]
show (list (random 10) 1 2 3 (random 10))
=> [4 1 2 3 9]  ;; ou une liste similaire
```

## ln

```
ln number
```

Retourne le logarithme naturel de *number*, c'est-à-dire le logarithme en base e (soit 2.71828...).

Voir aussi e et log

## log

```
log number base
```

Retourne le logarithme de *number* en base *base*.

```
show log 64 2
=> 6
```

Voir aussi ln.

## loop

```
loop [ commands ]
```

Exécute les commandes de la liste [*commands*] en boucle (encore et encore) ou jusqu'à ce que la procédure en cours soit interrompue par la commande stop ou par la commande report.

**Note** : Dans la plupart des cas, il vaut mieux utiliser un bouton "forever" pour répéter « interminablement » les mêmes commandes. L'avantage d'un tel bouton est que l'utilisateur peut le cliquer pour interrompre l'exécution de la boucle.

## lput

```
lput value list
```

Ajoute les données de l'entrée *value* à la fin de la liste *liste* et retourne la nouvelle liste ainsi créée.

```
;; supposons ma-liste valant [2 7 10 "Bob"]
set ma-liste lput 42 ma-liste
;; ma-liste vaut maintenant [2 7 10 "Bob" 42]
```

# M

## map

```
map [reporter] list
(map [reporter] list1 ...)
```

Avec une seule liste en entrée, `map` fait tourner le *`reporter`* donné avec chaque élément de la liste *`list`* et retourne ensuite une liste formée des valeurs retournées par les appels successifs du reporter.

Dans *`reporter`*, utilisez `?` pour faire référence à l'élément courant de la liste *`list`*.

```
show map [round ?] [1.1 2.2 2.7]
=> [1 2 3]
show map [? * ?] [1 2 3]
=> [1 4 9]
```

Avec plusieurs listes en entrée, le *`reporter`* donné est appelé pour chaque groupe d'éléments de chaque liste. Ainsi, il est appelé une fois pour le premier élément de chaque liste, une fois pour le deuxième élément de chaque liste, et ainsi de suite. Toutes les listes doivent avoir le même nombre d'éléments.

Dans *`reporter`*, utilisez `?1` à `?n` pour faire référence à l'élément courant de chaque liste.

Quelques exemples pour y voir plus clair :

```
show (map [?1 + ?2] [1 2 3] [2 4 6])
=> [3 6 9]
show (map [?1 + ?2 = ?3] [1 2 3] [2 4 6] [3 5 9])
=> [true false true]
```

Voir aussi `foreach` et `?`

## max

```
max list
```

Retourne l'élément de type nombre de la liste *`liste`* donnée en entrée ayant la plus grande valeur. Ignore les autres type d'éléments.

```
show max [xcor] of turtles
;; affiche la coordonnée x de la tortue qui est
;; la plus éloignée à droite de l'origine du monde
```

## max-n-of

```
max-n-of number agentset [reporter]
```

Retourne un ensemble d'agents contenant *`number`* agents de l'ensemble d'agents *`agentset`* ayant les plus grandes valeurs pour le *`reporter`*. L'ensemble d'agents est construit en trouvant tous les agents ayant la plus haute valeur de *`reporter`*. S'il n'y a pas *`number`* agents ayant cette valeur, alors les agents ayant la seconde plus haute valeur sont pris, et ainsi de suite. À la fin, s'il y a une ou des égalités qui conduisent à un ensemble d'agents trop grand, l'égalité est brisée aléatoirement.

```
;; supposons un monde de 11 x 11 patches
show max-n-of 5 patches [pxcor]
;; affiche 5 patches avec pxcor = max-pxcor
show max-n-of 5 patches with [pycor = 0] [pxcor]
```

```
;; affiche un ensemble d'agents contenant:
;; (patch 1 0) (patch 2 0) (patch 3 0) (patch 4 0) (patch 5 0)
```

Voir aussi `max-one-of` et `with-max`

## max-one-of

```
max-one-of agentset [reporter]
```

Retourne l'agent de l'ensemble d'agent *agentset* qui a la plus grande valeur pour le *reporter* donné. S'il y a égalité, cette commande retourne un agent pris au hasard parmi ceux ayant cette plus grande valeur. Si vous voulez tous les agents ayant la plus grande valeur, utilisez à la place la commande `with-max`.

```
show max-one-of patches [count turtles-here]
;; affiche le patch portant le plus grand nombre de tortues
```

Voir aussi `max-n-of` et `with-max`

## max-pxcor
## max-pycor

```
max-pxcor
max-pycor
```

Ces reporters retournent les plus grandes coordonnées x et y (respectivement) pour les patches, ce qui détermine la taille du monde.

Contrairement aux anciennes versions de NetLogo, l'origine n'a plus besoin d'être au centre du monde. Toutefois, les coordonnées x et y maximales doivent être plus grandes ou égales à zéro.

**Note** : la taille du monde ne peut être spécifiée que dans la boîte de dialogue "Model Settings" du panneau "Interface" car les valeurs retournée par ces reporters ne peuvent être fixées par programmation.

```
crt 100 [ setxy random-float max-pxcor
                random-float max-pycor ]
;; répartit 100 tortues aléatoirement
;; dans le premier quadrant
```

Voir aussi `min-pxcor`, `min-pycor`, `world-width` et `world-height`

## mean

```
mean list
```

Retourne la moyenne statistique de tous les éléments numériques de la liste *list*. Les éléments non numériques sont ignorés. Cette moyenne est obtenue en faisant la somme des valeurs des éléments numériques divisée par le nombre total d'éléments.

```
show mean [xcor] of turtles
;; affiche la moyenne des coordonnées x de toutes les tortues
```

## median

```
median list
```

Retourne la médiane statistique des éléments numériques de la liste *liste*. Les éléments non numériques sont ignorés. La médiane est l'élément qui se trouverait au milieu si tous les éléments étaient arrangés dans l'ordre. (Si deux éléments occupent le milieu, la médiane est la moyenne de ces deux éléments.)

```
show median [xcor] of turtles
;; affiche la médiane des coordonnées x de toutes les tortues
```

## member?

```
member? value list
member? string1 string2
member? agent agentset
```

Avec une liste en entrée, `member?` retourne `true` (vrai) si la valeur *value* apparaît dans la liste *list* donnée, sinon retourne `false` (faux).

Avec des chaînes de caractères en entrée, `member?` retourne `true` (vrai) si la chaîne de caractères *string1* apparaît quelque part dans la chaîne *string2* sous forme de sous-chaîne, sinon retourne `false` (faux).

Avec un ensemble d'agents en entrée, `member?` retourne `true` (vrai) si *agent* fait partie de l'ensemble d'agents *agentset*, sinon retourne `false` (faux).

```
show member? 2 [1 2 3]
=> true
show member? 4 [1 2 3]
=> false
show member? "bat" "abate"
=> true
show member? turtle 0 turtles
=> true
show member? turtle 0 patches
=> false
```

Voir aussi position

## min

```
min list
```

Retourne la plus petite valeur numérique de la liste *list*. Les autres types d'éléments sont ignorés.

```
show min [xcor] of turtles
;; affiche la plus petite coordonnée x parmi toutes les tortues
```

## min-n-of

```
min-n-of number agentset [reporter]
```

Retourne un ensemble d'agents contenant *number* agents de l'ensemble d'agents *agentset* ayant les plus petites valeurs pour le *reporter*. L'ensemble d'agents est construit en trouvant tous les agents ayant la plus petite valeur de *reporter*. S'il n'y a pas *number* agents ayant cette valeur, alors les agents ayant la seconde plus petite valeur sont pris, et ainsi de suite. À la fin, s'il y a une ou des égalités qui conduisent à un ensemble d'agents trop grand, l'égalité est brisée aléatoirement.

```
;; supposons un monde de 11 x 11
show min-n-of 5 patches [pxcor]
;; affiche 5 patches avec pxcor = min-pxcor
show min-n-of 5 patches with [pycor = 0] [pxcor]
;; affiche un ensemble d'agents contenant:
;; (patch -5 0) (patch -4 0) (patch -3 0) (patch -2 0) (patch -1 0)
```

Voir aussi `min-one-of` et `with-min`

## min-one-of

```
min-one-of agentset [reporter]
```

Retourne l'agent de l'ensemble d'agent *agentset* qui a la plus petite valeur pour le *reporter* donné. S'il y a égalité, cette commande retourne un agent pris au hasard parmi ceux ayant cette plus petite valeur. Si vous voulez tous les agents ayant la plus petite valeur, utilisez à la place la commande `with-min`.

```
show min-one-of turtles [xcor + ycor]
;; retourne la première tortue dont la somme
;; des coordonnées est la plus petite
```

Voir aussi `with-min` et `min-n-of`

## min-pxcor
## min-pycor

```
min-pxcor
min-pycor
```

Ces reporters retournent les plus petites coordonnées x et y (respectivement) pour les patches, ce qui détermine la taille du monde.

Contrairement aux anciennes versions de NetLogo, l'origine n'a plus besoin d'être au centre du monde. Toutefois, les coordonnées x et y minimales doivent être plus petites ou égales à zéro.

**Note** : la taille du monde ne peut être spécifiée que dans la boîte de dialogue "Model Settings" du panneau "Interface" car les valeurs retournée par ces reporters ne peuvent être fixées par programmation.

```
crt 100 [ setxy random-float min-pxcor random-float min-pycor ]
;; répartit 100 tortues aléatoirement dans le troisième quadrant
```

Voir aussi `max-pxcor`, `max-pycor`, `world-width` et `world-height`

## mod

```
number1 mod number2
```

Retourne *number1* modulo *number2*, c'est-à-dire le reste de la division de la première entrée *number1* par la seconde entrée *number2*. La primitive `mod` est équivalente au code NetLogo suivant :

```
number1 - (floor (number1 / number2)) * number2
```

Notez que `mod` est un opérateur « infixé », c'est-à-dire qu'il doit prendre place entre les deux entrées.

```
show 62 mod 5
=> 2
show -8 mod 3
=> 1
```

**Remarque** : `mod` et `remainder` se comportent de la même manière pour les nombres positifs, mais différemment pour les nombres négatifs.

Voir aussi `remainder`

## modes

```
modes list
```

Retourne une liste contenant l'élément ou les éléments apparaissant le plus souvent dans la liste donnée.

L'entrée *list* peut contenir n'importe quelle valeur admise par NetLogo. Si l'entrée est une liste vide, la commande retourne une liste vide.

```
show modes [1 2 2 3 4]
=> [2]
show modes [1 2 2 3 3 4]
=> [2 3]
show modes [ [1 2 [3]] [1 2 [3]] [2 3 4] ]
=> [[1 2 [3]]
show modes [pxcor] of turtles
;; affiche les colonnes de patches qui portent le plus de tortues
```

## mouse-down?

```
mouse-down?
```

Retourne `true` (vrai) si le bouton de la souris est pressé, sinon retourne `false` (faux).

**Note** : si le pointeur de la souris est hors de la Vue courante, `mouse-down?` retourne toujours `false`.

## mouse-inside?

```
mouse-inside?
```

Retourne `true` (vrai) si le pointeur de la souris est dans la Vue courante, sinon retourne `false` (faux).

## mouse-patch

```
mouse-patch
```

Retourne les coordonnées d'un patch si le pointeur de la souris en pointe un, sinon retourne `nobody`.

## mouse-xcor
## mouse-ycor

```
mouse-xcor
mouse-ycor
```

Retourne la coordonnée x, respectivement y, de la souris dans la vue 2D. La valeur est du type coordonnée de tortue, elle peut donc ne pas être un nombre entier. Si vous voulez des coordonnées de type nombre entier (comme pour les coordonnées des patches), utilisez `round mouse-xcor` et `round mouse-ycor`.

**Note** : si la souris est hors de la vue 2D, ce reporter retourne la valeur de sa dernière position dans la Vue.

```
;; pour que la tortue « dessine » un patch rouge :
if mouse-down? [
  ask patch mouse-xcor mouse-ycor [ set pcolor red ]
]
```

## move-to

```
move-to agent
```


La tortue donne à ses coordonnées x et y les mêmes valeurs que celles des coordonnées de l'*agent* spécifié, ce qui la place au même endroit que cet agent. (Si l'agent est un patch, la tortue est déplacée et mise au centre du patch.)

```
move-to turtle 5
;; la tortue se place au même endroit que la tortue 5
move-to one-of patches
;; la tortue se place au centre d'un patch pris au hasard
move-to max-one-of turtles [size]
;; la tortue se place à l'endroit où se trouve la plus grande tortue
```

Notez que l'orientation (le cap) de la tortue n'est pas modifiée. Vous devez d'abord utiliser la commande `face` pour orienter la tortue dans la direction de son mouvement.

Voir aussi `setxy`

## movie-cancel

```
movie-cancel
```

Annule l'animation vidéo courante.

## movie-close

```
movie-close
```

Stoppe l'enregistrement de l'animation vidéo courante.

## movie-grab-view
## movie-grab-interface

```
movie-grab-view
movie-grab-interface
```

Ajoute une image de la Vue courante (`movie-grab-view`) ou du panneau "Interface" (`movie-grab-interface`) à l'animation courante.

```
;; construit une animation de 20 images de la Vue courante
setup
movie-start "out.mov"
repeat 20 [ movie-grab-view go ]
movie-close
```

## movie-set-frame-rate

```
movie-set-frame-rate frame-rate
```

Spécifie la vitesse de l'animation. La vitesse *frame-rate* est mesurée un images (frames) par seconde. (Si vous ne spécifiez pas explicitement une vitesse, la valeur par défaut de 15 images par seconde est utilisée.)

Doit être appelée après `movie-start`, mais avant `movie-grab-view` ou `movie-grab-interface`.

Voir aussi `movie-status`

## movie-start

```
movie-start filename
```

Crée une nouvelle animation. L'entrée *filename* spécifie le nom du nouveau fichier QuickTime dans lequel l'animation sera enregistrée, c'est pourquoi il devra se terminer par l'extension «.mov».

Voir aussi movie-grab-view, movie-grab-interface, movie-cancel, movie-status, movie-set-frame-rate et movie-close

## movie-status

```
movie-status
```

Retourne une chaîne décrivant l'animation courante.

```
print movie-status
=> No movie.
movie-start
print movie-status
=> 0 frames; frame rate = 15
movie-grab-view
print movie-status
1 frames; frame rate = 15; size = 315x315.
```

## my-<breeds>
## my-links

```
my-<breeds>
my-links
```

Retourne un ensemble d'agents contenant tous les liens non orientés connectés à l'agent appelant.

```
crt 5
ask turtle 0
[
  create-links-with other turtles
  show my-links
  ;; affiche l'ensemble d'agents contenant tous les liens
  ;; (puisque tous les liens créés l'ont été à partir de la tortue 0)
]
ask turtle 1
[
  show my-links
  ;; affiche un ensemble d'agents contenant le lien 0 1
]
```

## my-in-<breeds>
## my-in-links

```
my-in-<breeds>
my-in-links
```

Retourne un ensemble d'agents contenant tous les liens orientés venant des autres noeuds vers l'agent appelant.

```
crt 5
ask turtle 0
[ create-links-to other turtles
  show my-in-links
  ;; affiche en ensemble d'agents vide
]
ask turtle 1
[ show my-in-links
  ;; affiche en ensemble d'agents contenant le lien 0 1
]
```

## my-out-<breeds>
## my-out-links

```
my-out-<breeds>
my-out-links
```

Retourne un ensemble d'agents contenant tous les liens orientés partant de l'agent appelant en direction d'autres noeuds.

```
crt 5
ask turtle 0
[ create-links-to other turtles
  show my-out-links
  ;; affiche un ensemble d'agent contenant tous les liens
]
ask turtle 1
[ show my-out-links
  ;; affiche en ensemble d'agents vide
]
```

## myself

```
myself
```

`self` et `myself` sont très différents. `self` signifie simplement « moi » alors que `myself` signifie « la tortue ou le patch qui m'a demandé de faire ce que je suis justement en train de faire ».

Quand il a été demandé à un agent d'exécuter un certain code, l'utilisation de `myself` dans ce code retourne l'agent (tortue ou patch) qui a fait la demande.

`myself` est le plus souvent utilisé avec `of` pour lire la valeur d'une variable de l'agent appelant ou donner une valeur cette variable.

`myself` peut être utilisé non seulement dans les blocs de la commande `ask`, mais aussi dans les commandes `hatch`, `sprout`, `of`, `with`, `all?`, `with-min`, `with-max`, `min-one-of`, `max-one-of`, `min-n-of` et `max-n-of`.

```
ask turtles
[
  ask patches in-radius 3
  [
    set pcolor [color] of myself
  ]
]
;; chaque tortue fait une tache colorée autour d'elle
```

Voir l'exemple de code "Myself Example" pour d'autres exemples.

Voir aussi `self`

# N

## n-of

```
n-of size agentset
n-of size list
```

Avec un ensemble d'agents en entrée, `n-of` retourne un ensemble d'agents contenant le nombre d'agents *`size`* choisis aléatoirement dans l'ensemble *`agentset`* fourni, sans répétitions.

Avec une liste en entrée, `n-of` retourne une liste contenant *`size`* éléments choisis aléatoirement dans la liste *`list`* fournie, sans répétitions. Les éléments de la liste retournée sont dans le même ordre que dans la liste fournie en entrée. (Si vous les voulez dans un ordre aléatoire, utilisez la primitive `shuffle` sur le résultat.)

Il y a une erreur si *`size`* est plus grand que la taille de l'entrée *`agentset`* ou de l'entrée *`list`*.

```
ask n-of 50 patches [ set pcolor green ]
;; 50 patches choisis aléatoirement deviennent verts
```

Voir aussi `one-of`

## n-values

```
n-values size [reporter]
```

Retourne une liste de longueur *`size`* contenant les valeurs calculées par une exécution répétée de *`reporter`*.

Dans l'entrée *`reporter`*, il faut utilisez `?` pour faire référence au numéro de l'élément qui est actuellement calculé, en commençant par zéro.

```
show n-values 5 [1]
=> [1 1 1 1 1]
show n-values 5 [?]
=> [0 1 2 3 4]
show n-values 3 [turtle ?]
=> [(turtle 0) (turtle 1) (turtle 2)]
show n-values 5 [? * ?]
=> [0 1 4 9 16]
```

Voir aussi `reduce`, `filter` et `?`

## neighbors
## neighbors4

```
neighbors
neighbors4
```

Retourne un ensemble d'agents contenant les 8 patches (avec `neighbors`) ou les 4 patches (avec `neighbors4`) voisins.

```
show sum [count turtles-here] of neighbors
;; affiche le nombre total de tortues se trouvant sur les
;; 8 patches placés autour de la tortue ou du patch appelant
show count turtles-on neighbors
;; une manière plus courte pour dire la même chose
ask neighbors4 [ set pcolor red ]
;; colore en rouge les 4 patches ayant un côté adjacent à l'agent appelant
```

## <breed>-neighbors
## link-neighbors

```
<breed>-neighbors
link-neighbors
```

Retourne un ensemble d'agents contenant toutes les tortues trouvées à l'autre extrémité des liens non orientés connectés à la tortue appelante.

```
crt 3
ask turtle 0
[
  create-links-with other turtles
  ask link-neighbors [ set color red ]
  ;; les tortues 1 et 2 deviennent rouges
]
ask turtle 1
[
  ask link-neighbors [ set color blue ]
  ;; la tortue 0 devient bleue
]
```

## <breed>-neighbor?
## link-neighbor?

```
<breed>-neighbor? turtle
link-neighbor? turtle
```

Retourne `true` (vrai) s'il y a un lien non orienté entre la tortue *`turtle`* et l'agent appelant.

```
crt 2
ask turtle 0
[
 create-link-with turtle 1
 show link-neighbor? turtle 1
 ;; affiche true
]
ask turtle 1
[
  show link-neighbor? turtle 0
  ;; affiche true
]
```

## netlogo-applet?

```
netlogo-applet?
```

Retourne `true` (vrai) si le modèle est exécuté en tant qu'applet (donc dans une page HTLM).

## netlogo-version

```
netlogo-version
```

Retourne une chaîne contenant le numéro de la version de NetLogo que vous êtes en train d'utiliser.

```
show netlogo-version
=> "4.0.3"
```

## new-seed

```
new-seed
```

Retourne un nombre convenant à l'initialisation (l'amorçage) du générateur de nombres aléatoires.

Les nombres retournés par `new-seed` sont basés sur la date et l'heure courantes du système, exprimées en millisecondes. Ils appartiennent à l'ensemble des entiers NetLogo (allant de -9007199254740992 à +9007199254740992).

`new-seed` ne retourne jamais deux fois de suite le même nombre. (NetLogo y arrive en attendant une milliseconde si l'amorce pour la milliseconde en cours a déjà été utilisée.)

Voir aussi `random-seed`

## no-display

```
no-display
```

Désactive toutes les mises à jour de la Vue courante jusqu'à l'utilisation de la commande `display`. Cette commande a deux utilisations principales.

Premièrement, vous pouvez contrôler quand l'utilisateur voit les mise à jours de la Vue. Vous pouvez vouloir modifier plein de choses sur la Vue (derrière le dos de l'utilisateur), puis les lui faire voir toutes d'un seul coup.

Deuxièmement, votre simulation tournera plus rapidement si la mise à jour de la vue est désactivée, donc, si vous êtes pressés, cette commande permettra d'obtenir des résultats plus rapidement. (Notez que, normalement, il n'est pas nécessaire d'utiliser `no-display` puisque l'utilisateur peut aussi activer ou désactiver les mises à jour de la Vue dans la case à cocher "view updates" du panneau "Interface" pour obtenir le même effet.)

Notez que les commandes `display` et `no-display` agissent indépendamment de la case à cocher qui gèle l'affichage.

Voir aussi `display`

## nobody

```
nobody
```

`nobody` (personne) est une valeur spéciale que retournent certaines primitives telles que `turtle`, `one-of`, `max-one-of`, etc. pour indiquer qu'aucun agent n'a été trouvé. De même, quand une tortue meurt (`die`), elle devient égale à `nobody`.

**Note** : Les ensembles d'agents vides ne sont pas égaux à `nobody`. Si vous voulez savoir si un ensemble d'agents est vide, utilisez `any?`. Vous ne recevez `nobody` en retour que dans les situations où vous n'attendez qu'un seul agent et pas un ensemble d'agents.

```
set other one-of other turtles-here
if other != nobody
  [ ask other [ set color red ] ]
```

## no-links

```
no-links
```

Retourne un ensemble d'agents de type lien vide.

## no-patches

```
no-patches
```

Retourne un ensemble d'agents de type patch vide.

## not

```
not boolean
```

Retourne `true` (vrai) si l'entrée *`boolean`* vaut `false` (faux), sinon retourne `false`.

```
if not any? turtles [ crt 10 ]
```

## no-turtles

```
no-turtles
```

Retourne un ensemble d'agents de type turtles vide.

# O

## of

```
[reporter] of agent
[reporter] of agentset
```

Avec un *agent* en entrée, `of` retourne la valeur de *reporter* pour cet agent (tortue ou patch).

```
show [pxcor] of patch 3 5
;; affiche 3
show [pxcor] of one-of patches
;; affiche la valeur de la variable pxcor d'un patch pris au hasard
show [who * who] of turtle 5
=> 25
show [count turtles in-radius 3] of patch 0 0
;; affiche le nombre de tortues situées dans un rayon de trois patches autour de l'origine
```

Avec un ensemble d'agents *agentset* en entrée, la primitive `of` retourne une liste contenant la valeur de *reporter* pour chaque agent de l'ensemble d'agents (dans un ordre aléatoire).

```
crt 4
show sort [who] of turtles
=> [0 1 2 3]
show sort [who * who] of turtles
=> [0 1 4 9]
```

## one-of

```
one-of agentset
one-of list
```

Avec un ensemble d'agents *agentset* en entrée, retourne un agent pris au hasard dans l'ensemble. Si l'ensemble d'agents est vide, retourne `nobody`.

Avec une liste en entrée, retourne un élément de *list* pris au hasard. Une liste vide provoque une erreur.

```
ask one-of patches [ set pcolor green ]
;; un patch pris au hasard devient vert
ask patches with [any? turtles-here]
[ show one-of turtles-here ]
;; pour chaque patch contenant des tortues,affiche l'une de ces tortues
;; supposons ma-liste valant [1 2 3 4 5 6]
show one-of ma-liste
;; affiche une valeur choisie au hasard dans ma-liste
```

Voir aussi `n-of`

## or

```
boolean1 or boolean2
```

Retourne `true` (vrai) si soit *boolean1*, soit *boolean2* est vrai, ou si les deux le sont.

Notez que si *boolean1* est vrai, alors les commandes de *boolean2* ne sont pas exécutées (puisqu'elles n'affectent pas le résultat final).

```
if (pxcor > 0) or (pycor > 0) [ set pcolor red ]
;; les patches deviennent rouges sauf ceux du quadrant inférieur gauche
```

## other

`other` *agentset*

`other` retourne un ensemble d'agents qui est le même que celui reçu en entrée mais ne contenant plus l'agent appelant.

```
show count turtles-here
=> 10
show count other turtles-here
=> 9
```

## other-end

`other-end`

Si `other-end` est exécuté par une tortue, retourne la tortue qui se trouve à l'autre extrémité du lien appelant.

Si `other-end` est exécuté par un lien, retourne la tortue qui est à extrémité du lien et qui n'est pas la tortue appelante.

Ces définitions sont difficiles à comprendre dans l'abstrait, mais les exemples suivants devraient aider à les comprendre :

```
ask turtle 0 [ create-link-with turtle 1 ]
ask turtle 0 [ ask link 0 1 [ show other-end ] ] ;; affiche turtle 1
ask turtle 1 [ ask link 0 1 [ show other-end ] ] ;; affiche turtle 0
ask link 0 1 [ ask turtle 0 [ show other-end ] ] ;; affiche turtle 1
```

Comme ces exemples (espérons-le) le montrent, l'« autre » extrémité est l'extrémité qui n'est ni appelante ni appelée.

## out-<breed>-neighbor? out-link-neighbor?

`out-<breed>-neighbor?` *turtle*
`out-link-neighbor?` *turtle*

Retourne `true` (vrai) s'il y a un lien orienté allant de l'agent appelant à la tortue *turtle*.

```
crt 2
ask turtle 0
[
  create-link-to turtle 1
  show in-link-neighbor? turtle 1 ;; affiche false
  show out-link-neighbor? turtle 1 ;; affiche true
]
ask turtle 1
[
  show in-link-neighbor? turtle 0 ;; affiche true
  show out-link-neighbor? turtle 0 ;; affiche false
]
```

## out-<breed>-neighbors
## out-link-neighbors

```
out-<breed>-neighbors
out-link-neighbors
```


Retourne un ensemble d'agents contenant toutes les tortues qui ont un lien orienté venant de l'agent appelant.

```
crt 4
ask turtle 0
[
 create-links-to other turtles
 ask out-link-neighbors [ set color pink ]
  ;; les tortues 1-3 deviennent roses
]
ask turtle 1
[
  ask out-link-neighbors [ set color orange ]
  ;; aucune tortue ne change de couleur puisque
  ;; la tortue 1 n'a qu'un lien venant vers elle
  ;; mais aucun lien partant d'elle
]
```

## out-<breed>-to
## out-link-to

```
out-<breed>-to turtle
out-link-to turtle
```


Retourne le lien allant de l'agent appelant à la tortue *turtle*. S'il n'y a pas de lien, retourne `nobody`.

```
crt 2
ask turtle 0
[
  create-link-to turtle 1
  show out-link-to turtle 1
  ;; affiche link 0 1
]
ask turtle 1
[
  show out-link-to turtle 0
  ;; affiche nobody
]
```

## output-print
## output-show
## output-type
## output-write

```
output-print value
output-show value
output-type value
output-write value
```

Ces commandes sont les mêmes que les commandes `print`, `show`, `type` et `write` sauf qu'ici, les données de l'entrée *value* sont affichées dans la zone de sortie "Output" du modèle plutôt que dans le Centre de commande. (Si le modèle ne possède pas de zone de sortie séparée, c'est le Centre de commande qui est utilisé.)

# P

## patch

```
patch xcor ycor
```

La primitive `patch` retourne le patch contenant le point défini par les coordonnées *xcor* et *ycor* reçues en entrée. (Les coordonnées sont des coordonnées absolues se rapportant à l'origine du monde NetLogo; elles ne sont pas calculées par rapport à l'agent appelant comme avec `patch-at`.)

Si *xcor* et *ycor* sont des nombres entiers, le point est situé au centre du patch. Si *xcor* ou *ycor* ne sont pas des nombres entiers, il y a arrondi vers l'entier le plus proche pour déterminer quel patch contient le point.

Si l'enroulement est autorisé par la topologie, les coordonnées données seront « enroulées » jusqu'à ce que le point se trouve à l'intérieur des limites du monde. Si l'enroulement n'est pas autorisé et que les coordonnées sortent du monde, la primitive retourne `nobody`.

```
ask patch 3 -4 [ set pcolor green ]
;; le patch ayant un pxcor de 3 et un pycor de -4 devient vert
show patch 1.2 3.7
;; affiche (patch 1 4), notez les arrondis
show patch 18 19
;; supposons min-pxcor et min-pycor valant -17 et max-pxcor et max-pycor valant 17 : dans une
;; topologie enroulante, affiche (patch -17 -16), dans une topologie non enroulante, affiche nobody
```

Voir aussi `patch-at`

## patch-ahead

```
patch-ahead distance
```

Retourne le patch qui se trouve à la *distance* donnée « devant » la tortue appelante, c'est-à-dire sur le cap de la tortue. Retourne `nobody` si le patch n'existe pas parce qu'il se situerait hors du monde.

```
ask patch-ahead 1 [ set pcolor green ]
;; colore le patch se trouvant une unité devant la tortue appelante en vert;
;; notez que ce peut être le même patch que celui sur lequel se trouve la tortue
```

Voir aussi `patch-at`, `patch-left-and-ahead`, `patch-right-and-ahead` et `patch-at-heading-and-distance`

## patch-at

```
patch-at dx dy
```

Retourne le patch situé aux distances (*dx*, *dy*) de l'agent appelant, c'est-à-dire le patch contenant le point situé à *dx* unités à l'est (vers la droite) et à *dy* unités au nord de l'agent appelant.

Retourne `nobody` s'il n'y a pas de pas un tel patch car le point se trouve à l'extérieur des limites d'un monde non « enroulé ».

```
ask patch-at 1 -1 [ set pcolor green ]
;; si l'agent appelant est une tortue ou un patch, colore le patch situé au sud-est en vert
```

Voir aussi `patch`, `patch-ahead`, `patch-left-and-ahead`, `patch-right-and-ahead` et `patch-at-heading-and-distance`

## patch-at-heading-and-distance

```
patch-at-heading-and-distance heading distance
```


`patch-at-heading-and-distance` retourne le patch qui se trouve le long du cap absolu *`heading`* donné (par rapport au Nord) et à la distance *`distance`* donnée de la tortue ou du patch appelant. (Contrairement à ce qui se passe avec les primitives `patch-left-and-ahead` et `patch-right-and-ahead`, le cap de la tortue appelante n'est pas pris en compte.) Retourne `nobody` si le patch n'existe pas car il se trouverait hors des limites du monde.

```
ask patch-at-heading-and-distance -90 1 [ set pcolor green ]
;; colore en vert le patch se trouvant à une distance d'une
;; unité à l'ouest (à gauche) du patch appelant
```

Voir aussi `patch`, `patch-at`, `patch-left-and-ahead` et `patch-right-and-ahead`

## patch-here

```
patch-here
```


`patch-here` retourne le patch sur lequel se trouve la tortue.

Notez que ce reporter n'est pas disponible pour un patch car un patch ne peut tout simplement pas s'adresser à lui-même.

## patch-left-and-ahead
## patch-right-and-ahead

```
patch-left-and-ahead angle distance
patch-right-and-ahead angle distance
```


Retourne le patch se trouvant à la distance *`distance`* donnée de la tortue appelante et le long de la direction donnée par le cap de la tortue corrigé à gauche (`patch-left-and-ahead`) ou à droite (`patch-right-and-ahead`) de la valeur de l'angle donné `angle`. Retourne `nobody` si le patch n'existe pas car il se trouverait hors des limites du monde.

Si vous voulez trouvez un patch le long d'un cap absolu plutôt que le long d'un cap relatif au cap de la tortue appelante, utilisez la primitive `patch-at-heading-and-distance`.

```
ask patch-right-and-ahead 30 1 [ set pcolor green ]
;; la tortue appelante «cherche» un patch situé à 30 degrés
;; à droite de son cap actuel et à une distance de 1 unité
;; et le colore vert. Notez que ce peut être le même patch
;; que celui sur lequel se trouve la tortue
```

Voir aussi `patch`, `patch-at` et `patch-at-heading-and-distance`

## patch-set

```
patch-set value1
(patch-set value1 value2 ...)
```

Retourne un ensemble d'agents contenant tous les patches cités n'importe où dans n'importe quelle entrée. Les entrées peuvent être des patches individuels, des ensembles d'agents de type patch, `nobody`, ou des listes (ou des listes imbriquées) contenant des objets cités ci-avant.

```
patch-set self
patch-set patch-here
(patch-set self neighbors)
(patch-set patch-here neighbors)
(patch-set patch 0 0 patch 1 3 patch 4 -2)
(patch-set patch-at -1 1 patch-at 0 1 patch-at 1 1)
patch-set [patch-here] of turtles
patch-set [neighbors] of turtles
```

Voir aussi `turtle-set` et `link-set`

## patches

```
patches
```

Retourne l'ensemble d'agents contenant tous les patches.

## patches-own

```
patches-own [var1 ...]
```

Ce mot-clé, tout comme les mots-clés `globals`, `breed`, `<race>-own` et `turtles-own`, ne peut être utilisé qu'au début d'un programme, avant toute définition de procédures. Il définit les variables que tous les patches peuvent utiliser.

Tous les patches auront alors les variables ainsi définies et pourront les utiliser.

Toutes les variables de type patch sont aussi directement accessibles par toutes les tortues se trouvant sur le patch.

Voir aussi `globals`, `turtles-own`, `breed` et `<breeds>-own`

## pcolor

```
pcolor
```

`pcolor` est une variable patch pré-programmée. Elle définit la couleur du patch. Vous pouvez (et devez) changer la valeur de cette variable pour modifier la couleur du patch.

Toutes les variables patch sont directement accessibles par toute tortue se trouvant sur le patch. Les couleurs sont représentées soit par une couleur NetLogo (un seul nombre) soit par une couleur RGB (une liste de 3 nombres). Voir les détails dans le chapitre Couleurs du Guide de la programmation.

Voir aussi `color`

## pen-down
## pd
## pen-erase
## pe
## pen-up
## pu

```
pen-down
pen-erase
pen-up
```


Ces commandes permettent de changer le mode de dessin de la tortue. `pen-down` ou `pd` (baisser-crayon) dessine une ligne, `pen-erase` ou `pe` efface une ligne et `pen-up` ou `pu` relève le crayon de sorte que la tortue ne dessine plus. Les lignes sont toujours dessinées sur le calque se trouvant au-dessus des patches et au-dessous des tortues. Pour modifier la couleur du crayon de la tortue, il faut modifier la couleur de la tortue avec `set color`.

**Note 1** : Quand le crayon d'une tortue est baissé, toutes les commandes de mouvements entraînent le dessin d'une ligne quand la tortue se déplace, y compris les commandes `jump`, `setxy` et `move-to`.

**Note 2** : On obtient le même résultat en donnant les valeurs `down`, `erase` ou `up` à la variable tortue `pen-mode`.

**Note 3** : Avec le système Windows, dessiner puis effacer une ligne peut ne pas effacer tous les pixels.

## pen-mode

```
pen-mode
```


`pen-mode` est une variable tortue préprogrammée. Elle spécifie l'état du crayon de la tortue. Vous pouvez donner à cette la variable la valeur `draw` pour dessiner des lignes, la valeur `erase` pour effacer des lignes ou la valeur `up` pour faire cesser l'une de ces deux actions.

## pen-size

```
pen-size
```


`pen-size` est une variable tortue préprogrammée. Elle spécifie la largeur de la ligne en pixels, autrement dit la largeur du crayon (ou de la gomme) avec lequel la tortue dessine (ou efface).

## plabel

```
plabel
```


`plabel` est une variable patch préprogrammée. Elle peut contenir un valeur de n'importe quel type. Le patch apparaît dans la Vue avec une étiquette contenant la valeur donnée sous forme de texte. Vous pouvez spécifier cette variable pour ajouter, modifier ou supprimer l'étiquette du patch.

Toutes les variables patch sont directement accessibles par toute tortue se trouvant sur le patch.

Voir aussi `plabel-color`, `label` et `label-color`

## plabel-color

```
plabel-color
```

`plabel-color` est une variable patch préprogrammée. Elle contient un nombre plus grand ou égal à zéro et plus petit que 140. Ce nombre définit la couleur de l'étiquette attachée au patch (si ce dernier en a une). Vous pouvez (et devez) modifier la valeur de cette variable pour changer la couleur de l'étiquette du patch.

Toutes les variables patch sont directement accessibles par toute tortue se trouvant sur le patch.

Voir aussi `plabel`, `label` et `label-color`

## plot

```
plot number
```

Incrémente l'abscisse (valeur x) du crayon du traceur de courbes de la valeur `plot-pen-interval`, puis dessine un point d'abscisse x dont la valeur vient d'être mise à jour et d'ordonnée dont la valeur est spécifiée par *`number`*. (La première fois que cette commande est utilisée par un traceur, le point dessiné a une abscisse de 0.)

## plot-name

```
plot-name
```

Retourne le nom du traceur de courbes courant (une chaîne de caractères).

## plot-pen-exists?

```
plot-pen-exists? string
```

Retourne `true` (vrai) si un crayon de traceur portant le nom spécifié dans la chaîne de caractères *`string`* est défini (existe) dans le traceur courant, sinon retourne `false` (faux).

## plot-pen-down
## plot-pen-up

```
plot-pen-down
plot-pen-up
```

Abaisse (`plot-pen-down`) ou relève (`plot-pen-up`) le crayon de traceur courant afin qu'il dessine (ou ne dessine pas). (Par défaut, tous les crayons sont initialement baissés.)

## plot-pen-reset

```
plot-pen-reset
```

Efface tout ce que le crayon actif du traceur a dessiné, déplace ce crayon à l'origine (0,0) et l'abaisse. Si le crayon est un crayon permanent, la couleur et le mode sont réinitialisés à leurs valeurs par défaut telles qu'elles ont été définies dans la boîte de dialogue "Plot" du panneau "Interface".

### plotxy

```
plotxy number1 number2
```

Déplace le crayon actif du traceur au point de coordonnées *number1 number2*. Si le crayon est baissé, une ligne, une barre ou un point est dessiné (en fonction du mode du crayon).

### plot-x-min
### plot-x-max
### plot-y-min
### plot-y-max

```
plot-x-min
plot-x-max
plot-y-min
plot-y-max
```

Retourne la valeur minimale (`plot-x-min`) ou maximale (`plot-x-max`) de l'axe x ou la valeur minimale (`plot-y-min`) ou maximale (`plot-y-max`) de l'axe y du traceur courant.

Ces valeurs peuvent être spécifiées avec les commandes `set-plot-x-range` et `set-plot-y-range`. (Leurs valeurs par défaut sont celles définies dans la boîte de dialogue "Plot" du panneau "Interface".)

### position

```
position item list
position string1 string2
```

Avec une liste, `position` retourne la position de la première occurrence de l'élément *item* se trouvant dans la liste *list*, ou `false` (faux) si cet élément ne fait pas partie de la liste.

Avec des chaînes, `position` retourne la position du premier caractère appartenant à la première occurrence de la chaîne *string1* qui apparaît en tant que sous-chaîne de la chaîne *string2* ou `false` (faux) si elle n'apparaît pas dans la chaîne.

**Note** : Les positions sont numérotées à partir de 0, pas de 1.

```
;; supposons ma-liste valant [2 7 4 7 "Bob"]
show position 7 ma-liste
=> 1
show position 10 ma-liste
=> false
show position "in" "string"
=> 3
```

Voir aussi `member?`

### precision

```
precision number places
```

Retourne le nombre donné *number* arrondi à *places* décimales.

Si *places* est négatif, l'arrondi se fait à gauche du point décimal.

```
show precision 1.23456789 3
=> 1.235
show precision 3834 -3
=> 4000
```

## print

```
print value
```

Affiche (écrit) les données de l'entrée *value* dans le Centre de Commande, suivies d'un retour du chariot .

L'agent appelant n'est pas affiché avant les données de *value*, contrairement à ce qui se passe avec show.

Voir aussi show, type, write et output-print

## pxcor
## pycor

```
pxcor
pycor
```

Ce sont des variables patch préprogrammées. Elles contiennent les coordonnées x (pxcor) et y (pycor) du patch. Ces coordonnées sont toujours des entiers. Vous ne pouvez pas modifier ces variables car les patches ne se déplacent pas.

pxcor est toujours plus grand ou égal à min-pxcor et plus petit ou égal à max-pxcor; il en va de même pour pycor avec min-pycor et max-pycor.

Toutes les variables patch sont directement accessibles par toute tortue se trouvant sur le patch.

Voir aussi xcor et ycor

# R

## random

```
random number
```

Le reporter `random` retourne un nombre aléatoire entier pris dans un ensemble de nombres défini par l'entrée *`number`*.

Si *`number`* est positif, le nombre aléatoire retourné est un entier plus grand ou égal à 0 mais strictement plus petit que *`number`*.

Si *`number`* est négatif, le nombre aléatoire retourné est un entier plus petit ou égal à 0 mais strictement plus grand que *`number`*.

Si *`number`* vaut zéro, le résultat est toujours 0.

**Note** : Dans les versions de NetLogo antérieures à 2.0, cette primitive retourne un nombre en virgule flottante si *`number`* n'est pas un entier. Ce qui n'est plus le cas. Si vous voulez un nombre aléatoire en virgule flottante, vous devez utiliser `random-float`.

```
show random 3
;; affiche 0, 1 ou 2
show random -3
;; affiche 0, -1 ou -2
show random 3.5
;; affiche 0, 1, 2 ou 3
```

Voir aussi `random-float`

## random-float

```
random-float number
```

Le reporter `random-float` retourne un nombre aléatoire en virgule flottante pris dans un ensemble de nombres défini par l'entrée *`number`*.

Si *`number`* est positif, le nombre aléatoire retourné est un nombre en virgule flottante plus grand ou égal à 0 mais strictement plus petit que *`number`*.

Si *`number`* est négatif, le nombre aléatoire retourné est un nombre en virgule flottante plus petit ou égal à 0 mais strictement plus grand que *`number`*.

Si *`number`* vaut zéro, le résultat est toujours 0.

```
show random-float 3
;; affiche un nombre valant au moins 0 mais plus petit que 3,
;; par exemple 2.589444906014774
show random-float 2.5
;; affiche un nombre valant au moins 0 mais plus petit que 2.5,
;; par exemple 1.0897423196760796
```

## random-exponential
## random-gamma
## random-normal
## random-poisson

```
random-exponential mean
random-gamma alpha lambda
random-normal mean standard-deviation
random-poisson mean
```

Ces primitives retournent un nombre aléatoire pris dans un ensemble de nombres dont la distribution est fonction de l'entrée *mean*.

`random-exponential` retourne un nombre aléatoire en virgule flottante choisi en fonction d'une distribution exponentielle.

`random-gamma` retourne un nombre aléatoire en virgule flottante choisi en fonction d'une distribution gamma, distribution contrôlée par les entrées en virgule flottante *alpha* et *lambda*. Ces deux paramètres doivent être plus grands que zéro. (**Note** : pour des résultats avec une moyenne et une variance données, construisez vos entrées comme suit : `alpha = moyenne * moyenne / variance; lambda = 1 / (variance / moyenne)`.)

`random-normal` retourne un nombre aléatoire en virgule flottante choisi en fonction d'une distribution normale avec une déviation standard spécifiée par *standard-deviation*.

`random-poisson` retourne un nombre aléatoire entier choisi en fonction d'une distribution de Poisson.

```
show random-exponential 2
;; affiche un nombre aléatoire en virgule flottante choisi en
;; fonction d'une distribution exponentielle avec une moyenne de 2
show random-normal 10.1 5.2
;; affiche un nombre aléatoire en virgule flottante choisi en
;; fonction d'une distribution normale avec une moyenne de 10.1
;; et une déviation standard de 5.2
show random-poisson 3.4
;; affiche une nombre aléatoire entier choisi en fonction d'une
;; distribution de Poisson avec une moyenne de 3.4
```

## random-pxcor
## random-pycor

```
random-pxcor
random-pycor
```

Retourne une nombre entier aléatoire appartenant à l'ensemble limité par les valeurs `min-pxcor` et `max-pxcor` (pour `random-pxcor`) ou à l'ensemble limité par les valeurs `min-pycor` et `max-pycor` (pour `random-pycor`). Les limites font partie des ensembles pris en compte.

```
ask turtles
[
  ;; déplace chaque tortue au centre d'un patch choisi aléatoirement
  setxy random-pxcor random-pycor
]
```

Voir aussi `random-xcor` et `random-ycor`

## random-seed

```
random-seed number
```

Initialise le générateur de nombres pseudo-aléatoires avec la partie entière de l'entrée *number*. L'amorce peut être un nombre entier appartenant l'ensemble des nombres entiers supportés par NetLogo (allant de -9007199254740992 à +9007199254740992).

Voir la section Nombres aléatoires du Guide de programmation pour plus de détails.

```
random-seed 47823
show random 100
=> 57
show random 100
=> 91
random-seed 47823
show random 100
=> 57
show random 100
=> 91
```

## random-xcor
## random-ycor

```
random-xcor
random-ycor
```

Retourne un nombre aléatoire en virgule flottante appartenant à l'ensemble des coordonnées autorisées de la tortue pour l'axe spécifié : axe x pour `random-xcor` et axe y pour `random-ycor`.

Les coordonnées de la tortue appartiennent à l'ensemble limité par `min-pxcor` - 0.5 (inclusivement) et `max-pxcor` + 0.5 (exclusivement) horizontalement et à l'ensemble limité par `min-pycor` - 0.5 (inclusivement) et `max-pycor` + 0.5 (exclusivement) verticalement.

```
ask turtles
[
  ;; déplace chaque tortue sur un point choisi aléatoirement
  ;; dans tout l'espace du monde NetLogo disponible
  setxy random-xcor random-ycor
]
```

Voir aussi `random-pxcor` et `random-pycor`

## read-from-string

```
read-from-string string
```

Interprète la chaîne de caractères *string* donnée comme si elle avait été tapée dans le Centre de commande et retourne la valeur résultant des opérations demandées. Le résultat peut être un nombre, une liste, une chaîne de caractères, une valeur booléenne ou encore la valeur spéciale `nobody`.

Utilisée conjointement avec la primitive `user-input`, cette commande permet de convertir l'entrée faite par l'utilisateur en une forme utilisable par NetLogo.

```
show read-from-string "3" + read-from-string "5"
=> 8
show length read-from-string "[1 2 3]"
=> 3
crt read-from-string user-input "Combien de tortues faut-il créer?"
;; NetLogo crée le nombre de tortues introduit par l'utilisateur
```

## reduce

`reduce [reporter] list`

Réduit la liste *liste* fournie en entrée de gauche à droite en utilisant les opérations définies par *reporter* jusqu'à la transformer en une seule valeur. Cela signifie par exemple que `reduce [?1 + ?2] [1 2 3 4]` est équivalent à `(((1 + 2) + 3) + 4)`. Si *list* ne comprend qu'un seul élément, c'est cet élément qui est retourné. La réduction d'une liste vide conduit à une erreur.

Dans *reporter*, il faut utiliser `?1` et `?2` pour faire référence aux deux éléments de la liste qui doivent être combinés.

Étant donné qu'il peut être difficile de comprendre intuitivement ce que fait `reduce`, voici quelques exemples simples qui, bien que n'étant en soi pas utiles, peuvent vous faire mieux comprendre ce que fait cette primitive :

```
show reduce [?1 + ?2] [1 2 3]
=> 6
show reduce [?1 - ?2] [1 2 3]
=> -4
show reduce [?2 - ?1] [1 2 3]
=> 2
show reduce [?1] [1 2 3]
=> 1
show reduce [?2] [1 2 3]
=> 3
show reduce [sentence ?1 ?2] [[1 2] [3 [4]] 5]
=> [1 2 3 [4] 5]
show reduce [fput ?2 ?1] (fput [] [1 2 3 4 5])
=> [5 4 3 2 1]
```

Voici quelques exemples plus utiles :

```
;; trouver la plus longue chaîne dans une liste
to-report longest-string [strings]
  report reduce
  [ifelse-value (length ?1 >= length ?2) [?1] [?2]]
  strings
end

show longest-string ["hi" "there" "!"]
=> "there"

;; compter le nombre d'occurences d'un élément dans une liste
to-report occurrences [x the-list]
  report reduce
  [ifelse-value (?2 = x) [?1 + 1] [?1]] (fput 0 the-list)
end

show occurrences 1 [1 2 1 3 1 2 3 1 1 4 5 1]
=> 6

;; évaluer le polynôme, avec les coefficients donnés, pour x
to-report evaluate-polynomial [coefficients x]
  report reduce [(x * ?1) + ?2] coefficients
end

;; évaluer 3x^2 + 2x + 1 pour x = 4
show evaluate-polynomial [3 2 1] 4
=> 57
```

## remainder

```
remainder number1 number2>
```

Retourne le reste de la division de la première entrée *number1* par la deuxième entrée *number2*. C'est l'équivalent du code NetLogo suivant :

```
number1 - (int (number1 / number2)) * number2
```

```
show remainder 62 5
=> 2
show remainder -8 3
=> -2
```

**Remarque** : `mod` et `remainder` se comportent de la même manière pour les nombres positifs, mais différemment pour les nombres négatifs.

Voir aussi `mod`

## remove

```
remove element list
remove string1 string2
```

Avec une liste comme deuxième entrée, `remove` retourne une copie de la liste *list* débarrassée de toutes les occurrences de le première entrée *element*.

Avec des chaînes de caractères en entrée, `remove` retourne une copie de la deuxième entrée *string2* débarrassée de toutes les sous-chaînes *string1* qui y apparaissent.

```
set ma-liste [2 7 4 7 "Bob"]
set ma-liste remove 7 ma-liste
;; ma-liste vaut maintenant [2 4 "Bob"]
show remove "to" "phototonique"
=> "phonique"
```

## remove-duplicates

```
remove-duplicates list
```

Retourne une copie de liste *list* débarrassée de tous les doublons de ses éléments. Seul le premier exemplaire de chaque doublon reste en place.

```
set ma-liste [2 7 4 7 "Bob" 7 "Bob"]
set ma-liste remove-duplicates ma-liste
;; ma-liste vaut maintenant [2 7 4 "Bob"]
```

## remove-item

```
remove-item index list
remove-item index string
```

Avec une liste comme deuxième entrée, `remove-item` retourne une copie de la liste *list* débarrassée de l'élément désigné par son numéro *index*.

Avec une chaîne de caractères comme deuxième entrée, `remove-item` retourne une copie de la chaîne *string* débarrassée du caractère se trouvant à la position *index*.

Notez que les éléments, ou les caractères, sont numérotés à partir de 0, pas de 1. (Le premier élément est l'élément d'index 0, le second, l'élément d'index 1, et ainsi de suite.)

```
set ma-liste [2 7 4 7 "Bob"]
set ma-liste remove-item 2 ma-liste
;; ma-liste vaut maintenant [2 7 7 "Bob"]
show remove-item 2 "string"
=> "sting"
```

## repeat

```
repeat number [ commandes ]
```

Exécute *number* fois les commandes de la liste [*commandes*].

```
pd repeat 36 [ fd 1 rt 10 ]
;; la tortue dessine un cercle
```

## replace-item

```
replace-item index list value
replace-item index string1 string2
```

Avec une liste comme deuxième entrée, `replace-item` remplace l'élément *index* de cette liste *list* par les données de l'entrée *value*. L'entrée *index* est le numéro de l'élément devant être remplacé, en commençant par 0 (donc, le 6e élément d'une liste porte le numéro 5). Notez que `replace-item` doit être utilisé avec `set` pour modifier une liste.

Il en va de même avec des chaînes, mais ici, c'est le caractère de numéro *index* de la chaîne *string1* qui est supprimé et remplacé par le contenu de la chaîne *string2*.

```
show replace-item 2 [2 7 4 5] 15
=> [2 7 15 5]
show replace-item 1 "cat" "are"
=> "caret"
```

## report

```
report valeur
```

Fait immédiatement sortir de la procédure `to-report` courante et retourne les données *valeur* en tant que résultat de la procédure. Les primitives `report` et `to-report` sont toujours utilisées l'une avec l'autre.

Voir la description de `to-report` pour savoir comment les utiliser.

## reset-perspective
## rp

```
reset-perspective
```

Après l'appel de cette procédure, l'observateur cesse de regarder (`watch`), de suivre (`follow`) ou de chevaucher (`ride`) toute tortue (ou patch). (S'il n'était pas en train de regarder, suivre ou chevaucher un agent, il ne se passe rien.) Dans la vue 3D, l'observateur retourne aussi à sa position par défaut (au-dessus de l'origine, regardant directement vers le bas).

Voir aussi `follow`, `ride` et `watch`

## reset-ticks

```
reset-ticks
```

Remets le compteur de cycles (tick counter) à zéro.

Voir aussi `tick`, `ticks` et `tick-advance`

## reset-timer

```
reset-timer
```

Remet le chronomètre (timer) à zéro seconde.

Noter que chronomètre et compteur de cycles sont deux choses différentes. Le chronomètre mesure le temps écoulé réel en secondes alors que le compteur de cycles mesure le temps écoulé de la simulation du modèle en cycles (`ticks`).

Voir aussi `timer`

## reverse

```
reverse list
reverse string
```

Retourne une copie «renversée» de la liste *`list`* ou de la chaîne de caractères *`string`* donnée en entrée.

```
show ma-liste
;; ma-liste vaut [2 7 4 "Bob"]
set ma-liste reverse ma-liste
;; ma-liste vaut maintenant ["Bob" 4 7 2]
show reverse "lived"
=> "devil"
```

## rgb

```
rgb red green blue
```

Ce reporter retourne une liste formée de trois nombres spécifiant les valeurs RGB de la couleur. Les valeurs données en entrée *`red`* (rouge), *`green`* (vert) et *`blue`* (bleu) sont des nombres entiers dans la plage 0-255.

Voir aussi `hsb`

## ride

```
ride turtle
```

Place le point de vue (l'observateur, l'utilisateur) sur la tortue *`turtle`*.

Chaque fois que la tortue « chevauchée» se déplace, l'observateur se déplace aussi avec elle. Il en résulte que dans la Vue 2D, la tortue reste toujours au centre de la Vue, c'est le monde qui se déplace autour d'elle. Dans la vue 3D, c'est comme si l'observateur voyait à travers les yeux de la tortue. Si la tortue meurt (`die`), le point de vue `perspective` redevient le point de vue par défaut.

Voir aussi `reset-perspective`, `watch`, `follow` et `subject`

## ride-me

```
ride-me
```


Demande à l'observateur de chevaucher (`ride`) la tortue appelante.

Voir aussi `ride`

## right
## rt

```
right number
```

Fait pivoter la tortue vers la droite de *`number`* degrés. Si *`number`* est négatif, la tortue pivote à gauche.

## round

```
round number
```

Retourne l'entier qui est le plus proche du nombre *`number`* passé en entrée. Si la partie décimale de *`number`* est exactement `.5`, le nombre est arrondi dans le sens **positif**.

Notez que l'arrondi dans le sens positif de NetLogo ne se fait pas toujours de la même manière que dans d'autres programmes. (En particulier, il ne correspond pas au comportement de StarLogoT qui arrondit toujours les nombres se terminant par .5 vers l'entier pair le plus proche.) La raison d'être du comportement de l'arrondi de NetLogo est qu'il doit correspondre à la manière dont les coordonnées des tortues sont en relation avec les coordonnées des patches. Par exemple, si la coordonnée `xcor` d'une tortue est -4.5, cette tortue se situe à la frontière entre le patch dont la coordonnée `pxcor` vaut -4 et celui dont la coordonnée `pxcor` vaut -5, mais la tortue doit être sur un patch ou sur l'autre (elle ne peut être ni sur deux patches à la fois, ni entre deux patches), ce qui fait que dans ce cas, la tortue est considérée comme étant sur le patch de coordonnée `pxcor` -4 puisque NetLogo arrondit vers les nombres positifs.

```
show round 4.2
=> 4
show round 4.5
=> 5
show round -4.5
=> -4
```

## run

```
run string
```

L'agent concerné interprète la chaîne de caractères *`string`* comme une séquence d'une ou de plusieurs commandes NetLogo et les exécute.

Le code est exécuté dans le contexte courant de l'agent, ce qui signifie qu'il a accès aux valeurs des variables locales telles que `myself`, etc.

Le code doit d'abord être compilé, ce qui prend du temps. Toutefois, les portions de code compilées sont mises en cache par NetLogo, ce qui fait que l'utilisation répétée de `run` sur la même chaîne *`string`* est beaucoup plus rapide que son utilisation sur des portions de code différentes.

Notez que vous **ne pouvez pas** utiliser `run` pour définir ou redéfinir des procédures.

Notez aussi que faire exécuter du code au moyen de `run` ou de `runresult` peut être plusieurs fois plus lent qu'exécuter le même code directement.

Voir aussi `runresult`

### runresult

```
runresult string
```

L'agent concerné interprète la chaîne de caractères *string* comme étant une séquence d'une ou de plusieurs commandes NetLogo d'un reporter, les exécute et retourne le résultat obtenu.

Le code est exécuté dans le contexte courant de l'agent, ce qui signifie qu'il a accès aux valeurs des variables locales telles que `myself`, etc.

Le code doit d'abord être compilé, ce qui prend du temps. Toutefois, les portions de code compilées sont mises en cache par NetLogo, ce qui fait que l'utilisation répétée de `runresult` sur la même chaîne *string* est beaucoup plus rapide que son utilisation sur des portions de code différentes.

Notez que vous **ne pouvez pas** utiliser `runresult` pour définir ou redéfinir des procédures.

Notez aussi que faire exécuter du code au moyen de `run` ou de `runresult` peut être plusieurs fois plus lent qu'exécuter le même code directement.

Voir aussi `run`

# S

## scale-color

```
scale-color color number range1 range2
```

Retourne une nuance de la couleur *color* proportionnelle à la valeur *number* choisie dans l'intervalle *range1* *range2*.

Si la limite définie par *range1* est plus petite que celle définie par *range2*, alors plus *number* est grand plus pâle est la nuance de *color*. Mais si *range2* est plus petit que *range1*, l'échelle des couleurs est inversée.

Si *number* est plus petit que *range1*, c'est la nuance la plus sombre de *color* qui est choisie.

Si *number* est plus grand que *range2*, c'est la nuance la plus pâle de *color* qui est choisie.

**Note** : pour *color*, la nuance (shade) n'a pas d'importance, c'est-à-dire que `green` et `green + 2` sont équivalents et que le même spectre de couleurs sera utilisé.

```
ask turtles [ set color scale-color red age 0 50 ]
;; colore chaque tortue avec une nuance de rouge
;; proportionnelle à la valeur de sa variable age
```

## self

```
self
```

Retourne la tortue ou le patch courant (actif, c'est-à-dire celui ou celle qui a donné cette commande).

`self` et `myself` sont très différents. `self` signifie simplement « moi » alors que `myself` signifie « la tortue ou le patch qui m'a demandé de faire ce que je suis en train de faire ».

Voir aussi `myself`

## ; (point-virgule)

```
; comments
```

Le reste de la ligne de code est ignoré après un point-virgule. Ce qui est très utile pour ajouter des commentaires à votre code — du texte qui explique le code aux lecteurs humains. Des points-virgules supplémentaires peuvent être ajoutés pour augmenter l'effet visuel des commentaires (comme dans les exemples de ce dictionnaire).

Le menu "Edit" de NetLogo possède des commandes permettant de commenter ou de dé-commenter des portions entières de code (dans le panneau "Procedures").

## sentence
## se

```
sentence value1 value2
(sentence value1 ...)
```

Construit une liste avec les entrées *value1*, *value2*, etc. reçues. Si une entrée est une liste, ses éléments sont incorporés directement dans la liste résultante plutôt que d'y être incorporés sous forme de sous-liste. Les exemples suivants illustrent son fonctionnement :

```
show sentence 1 2
=> [1 2]
show sentence [1 2] 3
=> [1 2 3]
show sentence 1 [2 3]
=> [1 2 3]
show sentence [1 2] [3 4]
=> [1 2 3 4]
show sentence [[1 2]] [[3 4]]
=> [[1 2] [3 4]]
show (sentence [1 2] 3 [4 5] (3 + 3) 7)
=> [1 2 3 4 5 6 7]
```

## set

`set` *`variable value`*

Donne à la variable *`variable`* la valeur spécifiée dans l'entrée *`value`*.

La *`variable`* peut être l'un de objets suivants :

- une variable globale déclarée en utilisant `globals` ;
- la variable globale associée à un curseur, un commutateur, un sélecteur ou une boîte d'entrée (dans le panneau "Interface") ;
- une variable appartenant à l'agent appelant ;
- une variable appartenant au patch sur lequel se trouve la tortue si l'agent appelant est une tortue ;
- une variable locale créée au moyen de la commande `let` ;
- une entrée de (une valeur reçue par) la procédure courante ;
- une variable locale spéciale (`?`, `?1`, `?2`...).

## set-current-directory

`set-current-directory` *`string`*

Indique à NetLogo le répertoire (dossier) courant qui devra être utilisé par les primitives `file-delete`, `file-exists?` et `file-open`.

Le répertoire courant n'est pas utilisé si les commandes ci-dessus reçoivent un chemin de fichier absolu. Par défaut, le répertoire courant est le répertoire `home` de l'utilisateur pour les nouveaux modèles, il est changé pour le répertoire dans lequel se trouve le modèle dès que ce modèle est chargé.

Notez que dans les chemins de fichiers Windows, le backslash doit être doublé d'un second backslash dans les chaînes, comme dans "`C:\\`".

Le changement de répertoire courant est temporaire et n'est pas sauvegardé avec le modèle.

**Note** : dans les applets, cette commande n'a pas d'effet puisque les applets ne sont autorisés à lire que les fichiers se trouvant dans le même répertoire que celui dans lequel se trouve le modèle.

```
set-current-directory "C:\\NetLogo"
;; supposé qu'il s'agisse d'un système Windows
file-open "my-file.txt"
;; ouvre le fichier "C:\\NetLogo\\my-file.txt"
```

## set-current-plot

`set-current-plot` *`plotname`*

Active le traceur de courbes nommé *`plotname`* (une chaîne de caractères) pour en faire le traceur courant. Les commandes de traçage de courbes subséquentes seront dirigées vers et exécutées par le traceur courant.

## set-current-plot-pen

```
set-current-plot-pen penname
```

Active le crayon portant le nom *penname* (une chaîne de caractères) pour en faire crayon courant du traceur courant. Si un tel crayon n'existe pas, il en résulte une erreur d'exécution.

## set-default-shape

```
set-default-shape turtles string
set-default-shape breed string
```

Spécifie une forme initiale par défaut pour toutes les tortues ou pour une race de tortues particulière. Quand une tortue est créée, ou qu'elle change de race, sa forme prend celle de la forme par défaut.

Cette commande ne concerne pas les tortues existantes mais seulement les tortues créées à la suite de son appel.

Le type d'agent spécifié doit être soit l'ensemble des tortues en général (`turtles`) soit une race de tortues nommée `breed` préalablement définie à l'aide du mot-clé `breed`. La chaîne de caractères *string* doit être le nom d'une forme de tortue qui a déjà été définie.

Dans les nouveaux modèles, la forme par défaut pour toutes les tortues est `"default"`.

Notez que la spécification d'une forme par défaut ne vous empêche pas de modifier la forme d'une tortue par la suite. Les tortues ne doivent pas être « prisonnières » de la forme de leur race par défaut.

```
create-turtles 1    ;; la forme de la nouvelle tortue est "default"
create-cats 1       ;; la forme de la nouvelle tortue est "default"

set-default-shape turtles "circle"
create-turtles 1    ;; la forme de la nouvelle tortue est "circle"
create-cats 1       ;; la forme de la nouvelle tortue est "circle"

set-default-shape cats "cat"
set-default-shape dogs "dog"
create-cats 1       ;; la forme de la nouvelle tortue est "cat"
ask cats [ set breed dogs ]
;; tous les chats deviennent des chiens et changent automatiquement
;; de forme pour prendre celle d'un chien "dog"
```

Voir aussi `shape`

## set-histogram-num-bars

```
set-histogram-num-bars number
```

Spécifie le nombre de barres *number* que devra dessiner le crayon courant du traceur courant si la commande `histogram` est appelée. NetLogo en déduit l'intervalle de dessin du crayon de manière à ce que le nombre de barres demandé couvre l'étendue courante de l'abscisse du graphique.

Voir aussi `histogram`

### __set-line-thickness

```
__set-line-thickness number
```


Spécifie l'épaisseur des lignes et des bordures dans la forme de la tortue.

La valeur par défaut est 0. Ce qui produit toujours une ligne de 1 pixel de large.

Les valeurs différentes de zéro sont interprétées comme des épaisseurs en patches. Une épaisseur de 1, par exemple, produit des lignes aussi épaisses que la largeur d'un patch. (Il est fréquent d'utiliser des valeurs plus petites que l'unité, telles que 0.5 ou 0.2.)

Les lignes ont toujours une épaisseur d'au moins un pixel.

Cette commande est expérimentale et peut changer dans des versions ultérieures de NetLogo.

### set-plot-pen-color

```
set-plot-pen-color number
```

Donne la couleur NetLogo de numéro *number* au crayon courant du traceur.

### set-plot-pen-interval

```
set-plot-pen-interval number
```

Demande au crayon courant du traceur de se déplacer d'une distance de *number* dans la direction x à chaque utilisation de la commande `plot`. (L'intervalle du crayon du traceur influence aussi le comportement de la commande `histogram`.)

### set-plot-pen-mode

```
set-plot-pen-mode number
```

Spécifie le mode de dessin du crayon du traceur courant à l'aide de *number*. Les modes de dessin pour ce crayon sont :

- **0** (mode ligne) : le crayon dessine une droite joignant les points du graphique.
- **1** (mode barre) : le crayon dessine une barre de largeur `plot-pen-interval` à partir du point dessiné dans le coin supérieur (ou inférieur si une valeur négative doit être représentée) gauche de la barre.
- **2** (mode point) : le crayon dessine un point pour représenter la valeur. Les points ne sont pas reliées les uns aux autres.

Le mode par défaut pour les nouveaux crayons est `0` (mode ligne).

### set-plot-x-range
### set-plot-y-range

```
set-plot-x-range min max
set-plot-y-range min max
```

Spécifie les valeurs minimales et maximales pour l'axe x (`set-plot-x-range`) ou les valeurs minimales et maximales pour l'axe y (`set-plot-y-range`) du traceur de courbes courant.

Cette modification est temporaire et n'est pas sauvegardée avec le modèle. Quand un traceur est nettoyé (effacé), ces dimensions reprennent les valeurs par défaut définies dans la fenêtre d'édition du traceur "Plot".

## setxy

```
setxy x y
```


La tortue donne la valeur *x* à sa variable `xcor` et la valeur `y` à sa variable `ycor` et se rend au point défini par ces coordonnées.

Cette commande est équivalente à `set xcor x set ycor y`, sauf que l'opération se fait en un seul pas au lieu de deux.

Si *x* ou *y* sort des limites du monde, NetLogo génère une erreur d'exécution.

```
setxy 0 0
;; la tortue se déplace au centre du patch central
setxy random-xcor random-ycor
;; la tortue se déplace sur un point aléatoire
setxy random-pxcor random-pycor
;; la tortue se déplace sur le centre d'un patch aléatoire
```

Voir aussi `move-to`

## shade-of?

```
shade-of? color1 color2
```

Retourne `true` (vrai) si les deux couleurs spécifiées par les nombres *color1 color2* (nombres valides pour désigner les couleurs dans le système de couleurs NetLogo) sont des nuances d'une même teinte, sinon retourne `false` (faux).

```
show shade-of? blue red
=> false
show shade-of? blue (blue + 1)
=> true
show shade-of? gray white
=> true
```

## shape

```
shape
```


`shape` est une variable tortue ou lien préprogrammée. Elle contient une chaîne de caractères qui est le nom de la forme courante de la tortue ou du lien. Vous pouvez changer la valeur de cette variable pour modifier la forme de l'agent. Les nouvelles tortues et les nouveaux liens ont la forme `"default"` à moins qu'une forme différente n'ait été spécifiée par l'utilisation de la primitive `set-default-shape`.

Exemple :

```
ask turtles [ set shape "wolf" ]
;; à supposé que vous ayez créé une forme "wolf"
;; dans l'Editeur de formes de tortue
ask links [ set shape "link 1" ]
;; à supposé que vous ayez créé une forme "link 1"
;; dans l'Editeur de formes de lien.
```

Voir aussi `set-default-shape` et `shapes`

## shapes

```
shapes
```

Retourne une liste de chaînes de caractères contenant toutes les formes de tortue présentes dans le modèle.

De nouvelles formes peuvent être créées, ou importées de la bibliothèque des formes ou importées d'autres modèles à l'aide de l'Editeur de formes.

```
show shapes
=> ["default" "airplane" "arrow" "box" "bug" ...
ask turtles [ set shape one-of shapes ]
```

## show

```
show value
```

Affiche les données de l'entrée *value* dans le Centre de commande, précédées du nom de l'agent appelant et suivi d'un retour de chariot. (L'agent appelant est ajouté pour vous permettre de savoir quel agent a produit quelle ligne de données.) De plus, toutes les chaînes conservent leurs guillemets, comme avec `write`.

Voir aussi `print`, `type`, `write` et `output-show`

## show-turtle st

```
show-turtle
```

La tortue est à nouveau visible.

**Note** : Cette commande revient à donner la valeur `false` à la variable tortue `hidden?`.

Voir aussi `hide-turtle`

## show-link

```
show-link
```

Le lien est à nouveau visible.

**Note** : Cette commande revient à donner la valeur `false` à la variable lien `hidden?`.

Voir aussi `hide-link`

## shuffle

```
shuffle list
```

Retourne une nouvelle liste contenant les mêmes éléments que la liste *list* donnée en entrée, mais dans un ordre aléatoire, autrement dit, elle mélange les éléments de *list*.

```
show shuffle [1 2 3 4 5]
=> [5 2 4 1 3]
show shuffle [1 2 3 4 5]
=> [1 3 5 2 4]
```

## sin

```
sin number
```

Retourne le sinus de l'angle donné *number*. L'angle doit être exprimé en degrés.

```
show sin 270
=> -1
```

## size

```
size
```

`size` est une variable tortue préprogrammée. Elle contient un nombre qui représente la taille apparente de la tortue. La taille par défaut est 1, ce qui signifie que la taille de la tortue est la même que celle d'un patch. Vous pouvez modifier cette variable pour changer la taille de la tortue.

## sort

```
sort list-of-numbers
sort list-of-strings
sort agentset
```

Le reporter `sort` sert à trier les éléments de l'ensemble reçu en entrée.

Si l'entrée est une liste de nombres *list-of-numbers* ou de chaînes de caractères *list-of-strings*, le reporter retourne une nouvelle liste contenant les mêmes éléments que la liste fournie, mais rangés par ordre croissant (numérique ou alphabétique).

Tout élément d'une liste qui n'est pas un nombre ou une chaîne de caractères est ignoré. (Si la liste fournie ne contient ni nombres ni chaînes, la primitive retourne une liste vide.)

Si l'entrée est un ensemble d'agent *agentset* ou une liste d'agents, `sort` retourne une liste d'agents (jamais un ensemble d'agents). Si les agents sont des tortues, elles sont listées par numéro d'identification (who number) croissant. Si les agents sont des patches, ils sont listés de gauche à droite et de haut en bas.

```
show sort [3 1 4 2]
=> [1 2 3 4]
let n 0
foreach sort patches [
  ask ? [
    set plabel n
    set n n + 1
  ]
]
;; les patches sont étiquetés avec des numéros
;; de gauche à droite et de haut en bas
```

## sort-by

```
sort-by [reporter] list
sort-by [reporter] agentset
```

Le reporter `sort-by` trie les éléments de l'ensemble reçu en entrée en fonction d'un critère défini par la liste `[reporter]` et retourne la liste triée.

Si l'entrée est une liste, `sort-by` retourne une nouvelle liste contenant les mêmes éléments que la liste *list* fournie en entrée, triés dans un ordre défini par le *reporter* booléen (`true` ou `false`).

Dans *reporter*, il faut utiliser ?1 et ?2 pour faire référence aux deux éléments devant être comparés. *reporter* doit valoir true (vrai) si ?1 vient strictement avant ?2 selon l'ordre de tri, et false (faux) dans tous les autres cas.

Si l'entrée est un ensemble d'agents *agentset* ou une liste d'agents, retourne une liste d'agents (jamais un ensemble d'agents).

Le tri est stable, c'est-à-dire que l'ordre des éléments considérés égaux par le *reporter* n'est pas perturbé.

```
show sort-by [?1 < ?2] [3 1 4 2]
=> [1 2 3 4]
show sort-by [?1 > ?2] [3 1 4 2]
=> [4 3 2 1]
show sort-by [length ?1 < length ?2] ["Grumpy" "Doc" "Happy"]
=> ["Doc" "Happy" "Grumpy"]
foreach sort-by [[size] of ?1 < [size] of ?2] turtles
  [ ask ? [ do-something ] ]
;; les tortues exécutent "do-something" les unes après
;; les autres par ordre de taille croissante
```

## sprout
## sprout-<*breeds*>

```
sprout number [ commands ]
sprout-<breeds> number [ commands ]
```

Crée le nombre de nouvelles tortues spécifié par l'entrée *number* sur le patch courant. Les nouvelles tortues ont des caps aléatoires (nombre entier de degrés par rapport au Nord) et leur couleur est choisie aléatoirement parmi les 14 couleurs primaires de NetLogo. Les tortues exécutent immédiatement les commandes spécifiées dans la liste [*commands*], ce qui est utile pour donner aux nouvelles tortues des couleurs, des caps ou quoi que ce soit d'autre différents. (Les nouvelles tortues sont créées toutes d'un seul coup puis elles exécutent les commandes les unes après les autres dans un ordre aléatoire.)

Si la forme sprout-<*breeds*> est utilisée, les nouvelles tortues sont automatiquement créées en tant que membre de la race <*breeds*> spécifiée.

```
sprout 5
sprout-wolves 10
sprout 1 [ set color red ]
sprout-sheep 1 [ set color black ]
```

**Note** : Tant que les commandes données sont exécutées, aucun autre agent n'a l'autorisation d'exécuter quelque code que ce soit (comme c'est le cas avec la commande without-interruption). Ceci afin de garantir que si ask-concurrent a été utilisé, les nouvelles tortues ne puissent interagir avec n'importe quel autre agent avant qu'elles n'aient toutes été complètement initialisées.

Voir aussi create-turtles et hatch

## sqrt

```
sqrt number
```

Retourne la racine carrée de *number*.

## stamp

```
stamp
```


La tortue ou le lien concerné laisse une image de sa forme sur le calque de dessin à sa position actuelle.

**Note** : Les formes créée par `stamp` peuvent ne pas être identiques au pixel près d'un système d'exploitation à l'autre.

## stamp-erase

```
stamp-erase
```


La tortue ou le lien concerné efface tous les pixels du calque de dessin qui se trouvent sous sa forme.

**Note** : Les formes dues à l'effacement du dessin par `stamp-erase` peuvent ne pas être identiques au pixel près d'un système d'exploitation à l'autre.

## standard-deviation

```
standard-deviation list
```

Retourne la déviation statistique standard des nombres de la liste *`list`* fournie en entrée. Ignore les autres types d'éléments.

```
show standard-deviation [1 2 3 4 5 6]
=> 1.8708286933869707
show standard-deviation [energy] of turtles
;; affiche la déviation standard de la variable
;; "energy" pour toutes les tortues
```

## startup

```
startup
```


Procédure définie par l'utilisateur qui, si elle existe, est appelée automatiquement quand le modèle est chargé. (Si le contenu de la procédure est défini par l'utilisateur, son nom, `startup` est prédéfini dans NetLogo.)

```
to startup
  setup
end
```

## stop

```
stop
```

L'agent appelant sort immédiatement de la procédure, de la construction `ask` ou semblable à `ask` (`crt`, `hatch`, `sprout`, `without-interruption`) englobante en cours. Seule la procédure en cours s'arrête, et non toutes celles exécutées par cet agent.

```
if not any? turtles [ stop ]
;; sort (s'arrête) s'il n'y a plus de tortues
```

**Note** : `stop` peut être utilisé pour stopper un bouton "forever". Si le bouton "forever" appelle directement une procédure, alors le bouton se déclenche quand la procédure s'arrête. (Dans un bouton "forever" de tortue ou de patch, le bouton ne se déclenche pas tant que chaque tortue ou chaque patch ne s'est pas arrêté — une seule tortue ou un seul patch n'a pas le pouvoir de déclencher un tel bouton.)

## subject

```
subject
```

Retourne la tortue (ou le patch) que l'observateur est en train de regarder (`watch`), de suivre (`follow`) ou de chevaucher (`ride`). Retourne `nobody` s'il n'y a pas de tel patch ou de telle tortue.

Voir aussi `watch`, `follow` et `ride`

## sublist<br>substring

```
sublist list position1 position2
substring string position1 position2
```

Retourne une partie de la liste *`list`* (avec `sublist`) ou de la chaîne de caractères *`string`* (avec `substring`) donnée allant de *`position1`* (compris) à *`position2`* (non compris).

**Note**: Les positions sont numérotées en commençant par 0 et non pas par 1.

```
show sublist [99 88 77 66] 1 3
=> [88 77]
show substring "appartement" 2 6
=> "part"
```

## subtract-headings

```
subtract-headings heading1 heading2
```

Calcule et retourne la différence entre les deux caps donnés, c'est-à-dire le nombre de degrés correspondant au plus petit angle dont il faut faire pivoter la direction *`heading2`* pour la faire coïncider avec la direction *`heading1`*. Une réponse positive signifie une rotation dans le sens des aiguilles d'une montre; une réponse négative, une rotation dans le sens inverse des aiguilles d'une montre. Le résultat se trouve toujours dans l'intervalle -180 à 180, mais ne vaut jamais exactement -180.

Notez qu'une simple soustraction des deux caps en utilisant l'opérateur `-` (moins) ne donne pas toujours le bon résultat. Cette soustraction correspond toujours à une rotation dans le sens des aiguilles d'une montre de la direction *`heading2`* vers *`heading1`*, mais parfois la rotation dans le sens inverse des aiguilles d'une montre est plus petite. Par exemple, l'angle minimal correspondant à différence entre le cap 5 et le cap 355 est 10 degrés et non de 350 degrés.

```
show subtract-headings 80 60
=> 20
show subtract-headings 60 80
=> -20
show subtract-headings 5 355
=> 10
show subtract-headings 355 5

=> -10
show subtract-headings 180 0
=> 180
show subtract-headings 0 180
=> 180
```

## sum

`sum` *`list`*

Retourne la somme des éléments de la liste *`list`* fournie en entrée.

```
show sum [energy] of turtles
;; affiche le total de la variable
;; "energy" de toutes les tortues
```

# T

### tan

```
tan number
```

Retourne la tangente de l'angle donné. L'angle est un nombre *number* exprimé en degrés.

### thickness

```
thickness
```


`thickness` est une variable lien préprogrammée. Elle contient un nombre qui définit l'épaisseur apparente du lien en fonction de la taille des patches. L'épaisseur par défaut est 0, ce qui signifie que, quelle que soit la taille des patches, le lien aura toujours une épaisseur de 1 pixel. Vous pouvez modifier la valeur de cette variable pour changer l'épaisseur du lien.

### tick

```
tick
```


Fait avancer le compteur de cycles d'une unité.

Voir aussi `ticks`, `tick-advance` et `reset-ticks`

### tick-advance

```
tick-advance number
```


Fait avancer le compteur de cycles (tick counter) du nombre de pas spécifié par *number*. L'entrée peut être un nombre entier ou un nombre en virgule flottante. (Certains modèles divisent les cycles (ticks) plus finement que l'unité.) L'entrée ne peut contenir un nombre négatif

Voir aussi `tick`, `ticks` et `reset-ticks`

### ticks

```
ticks
```

Retourne la valeur courante du compteur de cycles (tick counter). Le résultat est toujours un nombre et n'est jamais négatif.

La plupart des modèles utilisent la commande `tick` pour faire avancer le compteur de cycles et dans ce cas, `ticks` retournera toujours un nombre entier. Si la commande `tick-advance` a été utilisée, alors `ticks` peut retourner un nombre en virgule flottante.

Voir aussi `tick`, `tick-advance` et `reset-ticks`

## tie

`tie`

Attache rigidement l'une à l'autre les deux extrémités *end1* et *end2* du lien concerné. Si le lien est un lien orienté, *end1* devient la *tortue racine* et *end2* la *tortue feuille*. Le mouvement de la *tortue racine* influence la position et le cap de la *tortue feuille*. Si le lien n'est pas orienté, l'attache est réciproque ce qui fait que les deux tortues peuvent être considérées à la fois comme *tortues racines* et *tortues feuilles*. Le déplacement ou le changement de cap de l'une des deux tortues modifie l'emplacement et le cap de l'autre tortue.

Quand la *tortue racine* se déplace, les *tortues feuilles* se déplacent de la même distance et dans la même direction. Le cap de la *tortue feuille* n'est pas influencé. Cette primitive fonctionne avec `forward` et `jump` et prend en compte les modifications des coordonnées `xcor` et/ou `ycor` de la *tortue racine*.

Quand la *tortue racine* pivote à gauche ou à droite, la *tortue feuille* lui tourne autour dans le même sens et du même angle. Le cap de la *tortue feuille* est aussi modifié de la même valeur.

Si le lien est détruit (`die`), la relation d'attache rigide `tie` est supprimée.

```
crt 2 [ fd 3 ]
;; créer un lien et attacher la tortue 1 à la tortue 0
ask turtle 0 [ create-link-to turtle 1 [ tie ] ]
```

Voir aussi `untie`

## tie-mode

`tie-mode`

`tie-mode` est une variable de lien préprogrammée. Elle contient une chaîne caractères qui est le nom du mode d'attache du lien. L'utilisation des commandes `tie` et `untie` modifie le mode du lien. Vous pouvez aussi donnez la valeur `"free"` à la variable `tie-mode` pour créer une attache non rigide entre les deux tortues. (Voir la section Attache du Guide de la programmation pour plus de détails.) Par défaut, les liens ne sont pas attachés.

Voir aussi `tie` et `untie`

## timer

`timer`

Retourne le nombre de secondes écoulées depuis le dernier appel de la commande `reset-timer` (ou depuis le démarrage de NetLogo). La précision potentielle du chronomètre est la milliseconde. (La possibilité d'obtenir réellement une telle précision peut varier d'un système à l'autre en fonction des performances de la Machine Java Virtuelle sous-jacente.)

Notez que le chronomètre (timer) est différent du compteur de cycles (tick counter). Le chronomètre mesure le temps écoulé en secondes alors que le compteur de cycles mesure le temps écoulé du modèle en cycles (ticks).

Voir aussi `reset-timer`

## to

```
to procedure-name
to procedure-name [input1 ...]
```

Doit être utilisé pour marquer le début de la définition d'une procédure de type commande (avec ou sans entrées `[input1 ...]`).

```
to setup
  clear-all
  crt 500
end

to circle [radius]
  crt 100 [ fd radius ]
end
```

## to-report

```
to-report procedure-name
to-report procedure-name [input1 ...]
```

Doit être utilisé pour marquer le début de la définition d'une procédure de type **reporter** (appelée une fonction dans les autres langages de programmation).

Le corps de la procédure doit contenir le mot-clé `report` pour retourner la valeur calculée par la procédure.

```
to-report moyenne [a b]
  report (a + b) / 2
end

to-report valeur-absolue [nombre]
  ifelse nombre >= 0
    [ report nombre ]
    [ report (- nombre) ]
end

to-report premiere-tortue?
  report who = 0  ;; retourne true ou false
end
```

Voir aussi `report`

## towards

```
towards agent
```

Retour le cap que doit prendre l'agent appelant pour se mettre dans le même direction que (regarder vers) l'agent spécifié dans l'entrée *agent*.

Si l'enroulement est autorisé par la topologie et que la distance « enroulée » (autour des bords du monde) est plus courte, `towards` utilise le chemin « enroulé » pour calculer le cap.

**Note** : demander la cap d'un agent vers lui-même ou vers un agent placé au même endroit génère une erreur d'exécution.

```
set heading towards turtle 1
;; semblable à "face turtle 1"
```

Voir aussi `face`

## towardsxy

```
towardsxy x y
```


Retourne le cap à prendre à partir de l'endroit où se trouve la tortue ou le patch appelant pour faire face au (regarder vers) le point de coordonnées (*x y*).

Si l'enroulement est autorisé par la topologie et que la distance « enroulée » (autour des bords du monde) est plus courte, `towardsxy` utilise le chemin « enroulé » pour calculer le cap.

**Note** : demander la cap d'un point sur lequel se trouve déjà l'agent appelant génère une erreur d'exécution.

Voir aussi `facexy`

## turtle

```
turtle number
<breed> number
```

Retourne la tortue dont le numéro d'identification (who number) est spécifié par l'entrée *number*, ou `nobody` si cette tortue n'existe pas. Pour les tortues appartenant à une race, vous pouvez aussi utiliser la forme singulier de la race *<breed>* pour y faire référence. Ce reporter permet de s'adresser à (donner des commandes ou d'interroger) une tortue particulière.

```
ask turtle 5 [ set color red ]
;; la tortue numéro 5 devient rouge
```

## turtle-set

```
turtle-set value1
(turtle-set value1 value2 ...)
```

Retourne un ensemble d'agents contenant toutes les tortues citées n'importe où dans les différentes entrées. Les entrées peuvent être des tortues individuelles, des ensembles d'agents contenant des tortues, `nobody` ou encore des listes (ou des listes imbriquées) contenant les entrées citées ci-dessus.

```
turtle-set self
(turtle-set self turtles-on neighbors)
(turtle-set turtle 0 turtle 2 turtle 9)
(turtle-set frogs mice)
```

Voir aussi `patch-set` et `link-set`

## turtles

```
turtles
```

Retourne un ensemble d'agents contenant toutes les tortues présentes dans le modèle au moment de son appel.

```
show count turtles
;; affiche le nombre de tortues
```

### turtles-at
### *<breeds>*-at

```
turtles-at dx dy
<breeds>-at dx dy
```

Retourne un ensemble d'agents contenant toutes les tortues se trouvant sur le patch situé à la distance horizontale *dx* et à la distance verticale *dy* de l'agent appelant. (Le résultat peut contenir l'agent appelant lui-même si l'agent appelant est une tortue.)

Si le nom *breeds* d'une race est utilisé à la place de `turtles`, seules les tortues de cette race feront partie de l'ensemble retourné.

```
create-turtles 5 [ setxy 2 3 ]
show count [turtles-at 1 1] of patch 1 2
=> 5
```

### turtles-here
### *<breed>*-here

```
turtles-here
<breeds>-here
```

Retourne un ensemble d'agents contenant toutes les tortues se trouvant sur le patch portant l'agent appelant (y compris l'agent appelant si ce dernier est une tortue).

```
crt 10
ask turtle 0 [ show count turtles-here ]
=> 10
```

Si le nom *breeds* d'une race est mis à la place de `turtles`, seules les tortues de cette race feront partie de l'ensemble retourné.

```
breed [cats cat]
breed [dogs dog]
create-cats 5
create-dogs 1
ask dogs [ show count cats-here ]
=> 5
```

### turtles-on
### *<breeds>*-on

```
turtles-on agent
turtles-on agentset
<breeds>-on agent
<breeds>-on agentset
```

Retourne un ensemble d'agents contenant toutes les tortues se trouvant sur le patch (*agent*) ou l'ensemble de patches (*agenset*) donné, ou se trouvant sur le même patch que la tortue (*agent*) ou l'ensemble de tortues (*agenset*) donné.

Si le nom *breeds* d'une race est mis à la place de `turtles`, seules des tortues appartenant à cette race feront partie de l'ensemble retourné.

```
ask turtles [
  if not any? turtles-on patch-ahead 1
    [ fd 1 ]
]
ask turtles [
  if not any? turtles-on neighbors [
    die-of-loneliness
  ]
]
```

## turtles-own<br>*<breeds>*-own

```
turtles-own [var1 ...]
<breeds>-own [var1 ...]
```

Le mot-clé `turtles-own`, tout comme les mots-clés `globals`, `breed`, `<races>-own` et `patches-own` ne peut être utilisé qu'au début du code, avant toute définition de procédures. Il permet de définir les variables qui appartiendront à chaque tortue.

Si vous spécifiez une race *breeds* à la place de `turtles`, seules les tortues appartenant à cette race posséderont les variables définies. (Une même variable peut être partagée par plusieurs races.)

```
breed [cats cat ]
breed [dogs dog]
breed [hamsters hamster]
turtles-own [eyes legs]   ;; valable pour toutes les races
cats-own [fur kittens]
hamsters-own [fur cage]
dogs-own [hair puppies]
```

Voir aussi `globals`, `patches-own`, `breed` et `<breeds>-own`

## type

```
type value
```

Affiche les données de l'entrée *value* dans le Centre de commande, non suivi d'un retour de chariot (contrairement à `print` et `show`). L'absence de retour de chariot permet d'écrire plusieurs paquets de données sur la même ligne.

L'agent appelant n'est pas affiché avant les données, contrairement à ce qui se passe avec `show`

```
type 3 type " " print 4
=> 3 4
```

Voir aussi `print`, `show`, `write` et `output-type`

# U

## undirected-link-breed

```
undirected-link-breed [<link-breeds> <link-breed>]
```

Ce mot-clé, tout comme les mots-clés `globals` et `breeds`, ne peut être utilisé qu'au début du code écrit dans le panneau "Procedures", avant toute définition de procédures. Il définit un lien non orienté de race. Les liens d'une race particulière sont toujours soit tous orientés, soit tous non orientés. La première entrée, *<link-breeds>*, spécifie l'ensemble d'agents associé au lien de race, la seconde, *<link-breed>* spécifie le nom d'un membre particulier de cette race.

Tout lien du lien de race donné :

- appartient à l'ensemble d'agents désigné par le nom du lien de race,
- a sa variable préprogrammée `breed` initialisée avec le nom de cet ensemble d'agents,
- est orienté ou non orienté comme spécifié par le mot-clé.

Le plus souvent, l'ensemble d'agents est utilisé avec `ask` pour ne donner des commandes qu'aux liens d'une race particulière.

```
undirected-link-breed [streets street]
undirected-link-breed [highways highway]
to setup
  clear-all
  crt 2
  ask turtle 0 [ create-street-with turtle 1 ]
  ask turtle 0 [ create-highway-with turtle 1 ]
end

ask turtle 0 [ show sort my-links ]
;; affiche [(street 0 1) (highway 0 1)]
```

Voir aussi `breed` et `directed-link-breed`

## untie

```
untie
```

Désolidarise (détache) l'extrémité *end2* de l'extrémité *end1* d'un lien (donne la valeur `"none"` à la variable `tie-mode` du lien) si ces agents étaient physiquement attachés. Si le lien est un lien non orienté, cette commande désolidarise ausssi aussi l'extrémité *end1* de l'extrémité *end2*. Elle ne supprime **pas** le lien entre les deux tortues.

Voir la section Attache du Guide de la programmation pour de plus amples informations.

Voir aussi `tie`

## uphill
## uphill4

```
uphill patch-variable
uphill4 patch-variable
```

Déplace la tortue sur le patch voisin dont la variable *patch-variable* à la plus grande valeur parmi tous les patches voisins. Si aucun patch voisin n'a une valeur supérieure pour cette variable à celle du patch courant

(le patch sur lequel se trouve la tortue), la tortue reste où elle est. Si plusieurs patches ont la même valeur maximale, la tortue en choisit un au hasard et s'y rend. Les valeurs non numériques sont ignorées.

`uphill` prend en considération les huit patches voisins, `uphill4` seulement les 4 patches voisins (ceux dont les côtés sont adjacents).

Cette commande est équivalente au morceau de code suivant (en supposant que les valeurs des variables sont numériques) :

```
move-to patch-here  ;; va a centre de patch
let p max-one-of neighbors [patch-variable]  ;; ou neighbors4
if [patch-variable] of p > patch-variable [
  face p
  move-to p
]
```

Notez que la tortue finit toujours au centre d'un patch avec un cap qui est un multiple de 45 (avec up`hill`) ou de 90 (avec up`hill4`).

Voir aussi `downhill` et `downhill4`

## user-directory

`user-directory`

Ouvre une boîte de dialogue permettant à l'utilisateur de sélectionner un répertoire (dossier) existant dans le système.

Retourne une chaîne de caractères contenant le chemin absolu du répertoire, ou `false` (faux) si l'utilisateur annule l'opération.

```
set-current-directory user-directory
;; suppose que l'utilisateur veuille choisir un répertoire
```

## user-file

`user-file`

Ouvre une boîte de dialogue permettant à l'utilisateur de sélectionner un fichier existant dans le système.

Retourne une chaîne de caractères contenant le chemin absolu du fichier, ou `false` (faux) si l'utilisateur annule l'opération.

```
file-open user-file
;; suppose que l'utilisateur veuille choisir un fichier
```

## user-new-file

`user-new-file`

Ouvre une boîte de dialogue permettant à l'utilisateur de choisir un emplacement (un répertoire) et de donner un nom pour le nouveau fichier à créer. Retourne une chaîne contenant le chemin absolu du fichier, ou `false` (faux) si l'utilisateur annule l'opération.

```
file-open user-new-file
;; suppose que l'utilisateur veuille créer un nouveau fichier
```

Notez que ce reporter ne crée pas véritablement le fichier : vous devez normalement créer le fichier avec `file-open`, comme dans l'exemple ci-dessus.

Si l'utilisateur choisit un fichier existant (ou écrit le nom d'un fichier existant), il lui sera demandé s'il veut le remplacer ou non par le nouveau, mais le reporter ne provoque jamais lui-même le remplacement d'un fichier existant. Pour le faire, vous devez utiliser la primitive `file-delete`.

### user-input

`user-input` *`value`*

Ouvre une boîte de dialogue affichant le message *`value`* et retourne la chaîne entrée par l'utilisateur dans le champ texte du dialogue.

L'entrée *`value`* peut être de n'importe quel type, mais il s'agit généralement d'une chaîne de caractères.

```
show user-input "Quel est votre nom ?"
```

### user-message

`user-message` *`value`*

Ouvre une boîte de dialogue affichant le message `value`.

L'entrée *`value`* peut être de n'importe quel type, mais il s'agit généralement d'une chaîne de caractères.

```
user-message (word "Il y a " count turtles " tortues.")
```

### user-one-of

`user-one-of` *`value list-of-choices`*

Ouvre une boîte de dialogue affichant le message spécifié par *`value`* et un menu permettant à l'utilisateur de sélectionner l'un des éléments de la liste *`list-of-choices`*.

Retourne l'élément de la liste *`list-of-choices`* sélectionné par l'utilisateur.

L'entrée *`value`* peut être de n'importe quel type, mais il s'agit généralement d'une chaîne de caractères.

```
if "oui" = user-one-of? "Initialiser le modèle?" ["oui" "non"]
  [ setup ]
```

### user-yes-or-no?

`user-yes-or-no?` *`value`*

Ouvre une boîte de dialogue et retourne la réponse `true` (vrai) ou `false` (faux) de l'utilisateur à la question *`value`* posée.

L'entrée *`value`* peut être de n'importe quel type, mais il s'agit généralement d'une chaîne de caractères.

```
if user-yes-or-no? "Initialiser le modèle?"
  [ setup ]
```

# V

## variance

```
variance list
```

Retourne la variance de l'échantillon représenté par la liste de nombres *list* donnée en entrée. Ignore les autres types d'éléments.

La variance est la somme des carrés des déviations des nombres par rapport à la moyenne, divisée par le nombre de nombres dans la liste moins 1.

```
show variance [2 7 4 3 5]
=> 3.7
```

# W

## wait

```
wait number
```

Attend le nombre de secondes *number* donné. (*nombre* ne doit pas obligatoirement être un entier, vous pouvez spécifier des fractions de secondes.) Notez que vous ne pouvez atteindre une précision absolue : l'agent n'attendra jamais moins que la durée spécifiée, mais il peut parfois attendre un tout petit peu plus.

```
repeat 10 [ fd 1 wait 0.5 ]
```

Voir aussi every

## watch

```
watch agent
```

Dirige la lumière d'un projecteur sur l'agent *agent* donné. Dans la vue 3D, l'observateur se tourne toujours pour faire face au sujet.

Voir aussi follow, subject, reset-perspective et watch-me

## watch-me

```
watch-me
```

Demande à l'observateur de regarder l'agent appelant.

Voir aussi watch

## while

```
while [ reporter ]
```

La primitive while permet de construire une boucle « tant que ». Elle teste d'abord la condition [*reporter*], exécute les commandes de l'entrée [*commands*] si la condition est remplie (vraie) et revient au début de la boucle pour faire un nouveau test. La boucle est exécutée « tant que » la condition est remplie. La boucle est quittée dès que la condition échoue, sans exécuter les commandes qu'elle contient.

Le *reporter* peut prendre différentes valeurs pour différents agents, ce qui fait que tous les agents n'exécutent pas les commandes le même nombre de fois.

```
while [any? other turtles-here]
  [ fd 1 ]
;; la tortue se déplace jusqu'à ce qu'elle trouve
;; un patch sur lequel il n'y a pas encore de tortue(s)
```

## who

```
who
```


`who` est une variable tortue préprogrammée. Elle contient le numéro d'identification « who number » de la tortue, un nombre entier plus grand ou égal à zéro. Vous ne pouvez pas initialiser ou modifier la valeur de cette variable : le numéro d'identification est donné à la tortue lors de sa création et ne change plus.

Les numéros d'identification commencent à 0. Le numéro d'identification d'une tortue décédée (`die`), donc théoriquement devenu disponible, n'est pas réaffecté à une nouvelle tortue tant que vous n'utilisez pas les commandes `clear-turtles` ou `clear-all`; ce n'est qu'à ce moment-là que la numérotation des tortues recommence à partir de 0.

Exemple :

```
show [who] of turtles with [color = red]
;; affiche la liste des numéros d'identification de toutes les
;; tortues rouges dans le Centre de commande, en ordre aléatoire
crt 100
  [ ifelse who < 50
  [ set color red ]
  [ set color blue ] ]
;; les tortues 0 à 49 deviennent rouges,
;; les tortues 50 à 99 deviennent bleues
```

Vous pouvez utiliser le reporter `turtle` pour atteindre une tortue ayant un numéro d'identification donné.

Voir aussi `turtle`

## with

```
agentset with [ reporter ]
```

Retourne un nouvel ensemble d'agents ne contenant que les agents de l'ensemble *`agentset`* ayant satisfait (retourné `true` (vrai)) à la condition `[`*`reporter`*`]` donnée. `with` demande deux entrées : une à gauche, un ensemble d'agents *`agentset`* (généralement `turtles` ou `patches`), l'autre à droite qui doit être un reporter booléen.

```
show count patches with [pcolor = red]
;; affiche le nombre de patches rouges
```

## <breed>-with<br>link-with

```
<breed>-with turtle
link-with turtle
```


Retourne le lien entre la tortue *`turtle`* et l'agent appelant. S'il n'y a pas de lien, retourne `nobody`.

```
crt 2
ask turtle 0 [
  create-link-with turtle 1
  show link-with turtle 1 ;; affiche link 0 1
]
```

## with-max

*agentset* with-max [*reporter*]

Retourne un nouvel ensemble d'agents ne contenant que les agents de l'ensemble *agentset* ayant la valeur maximale pour le reporter [*reporter*] donné. with-max demande deux entrées, celle de gauche doit être l'ensemble d'agents *agentset* contenant les agents à tester (généralement turtles ou patches), celle de droite doit être un reporter définissant les valeurs à tester.

```
show count patches with-max [pxcor]
;; affiche le nombre de patches sur le bord droit du monde
```

Voir aussi max-one-of et max-n-of

## with-min

*agentset* with-min [*reporter*]

Retourne un nouvel ensemble d'agents ne contenant que les agents de l'ensemble *agentset* ayant la valeur minimale pour le reporter [*reporter*] donné. with-min demande deux entrées, celle de gauche doit être l'ensemble d'agents *agentset* contenant les agents à tester (généralement turtles ou patches), celle de droite doit être un reporter définissant les valeurs à tester.

```
show count patches with-min [pycor]
;; affiche le nombre de patches au bord inférieur du monde
```

Voir aussi min-one-of et min-n-of

## with-local-randomness

with-local-randomness [ *commands* ]

Les commandes spécifiées dans l'entrée [*commands*] sont exécutées sans avoir aucune influence sur les événements aléatoires suivants. Cette primitive est utile pour effectuer des opérations supplémentaires (telle qu'un affichage de données) sans qu'elles modifient la suite de la simulation.

Exemple :

```
;; première exécution :
random-seed 50 setup repeat 10 [ go ]
;; seconde exécution :
random-seed 50 setup
with-local-randomness [ watch one-of turtles ]
repeat 10 [ go ]
```

Puisque one-of est utilisée dans le bloc without-local-randomness, les deux exécutions seront identiques.

Plus précisément, cette primitive fonctionne de la manière suivante : l'état du générateur de nombres aléatoires est sauvegardé avant l'exécution du bloc de commandes, puis restauré après coup. (Si vous voulez exécuter les commandes du bloc avec un nouvel état du générateur de nombres aléatoires plutôt qu'avec l'état qui sera restauré par la suite, commencez le bloc de commandes avec random-seed new-seed.)

L'exemple suivant montre que l'état du générateur de nombres aléatoires est le même avant et après l'exécution du bloc de commandes.

```
random-seed 10
with-local-randomness [ print n-values 10 [random 10] ]
;; affiche [8 9 8 4 2 4 5 4 7 9]
print n-values 10 [random 10]
;; affiche [8 9 8 4 2 4 5 4 7 9]
```

## without-interruption

```
without-interruption [ commands ]
```

L'agent exécute toutes les commandes du bloc `[commands]` sans permettre à d'autres agents utilisant `ask-concurrent` d'interrompre ce travail. Il en résulte que les autres agents sont placés « en attente » et n'exécutent aucune commande tant que l'exécution des commandes du bloc n'est pas terminée.

**Note** : cette commande n'a de sens qu'utilisée avec la primitive `ask-concurrent`. Dans les versions précédentes de NetLogo, elle était souvent nécessaire, mais depuis NetLogo 4.0, elle n'est utile que si vous utilisez aussi `ask-concurrent`.

Voir aussi `ask-concurrent`

## word

```
word value1 value2
(word value1 ...)
```

Concatène les entrées (les met les unes à la suite des autres sans espaces) dans l'ordre où elles sont données et retourne le résultat sous forme de chaîne de caractères.

```
show word "tur" "tle"
=> "turtle"
word "a" 6
=> "a6"
set directory "c:\\foo\\fish\\"
show word directory "bar.txt"
=> "c:\foo\fish\bar.txt"
show word [1 54 8] "fishy"
=> "[1 54 8]fishy"
show (word 3)
=> "3"
show (word "a" "b" "c" 1 23)
=> "abc123"
```

## world-width
## world-height

```
world-width
world-height
```

Ces reporters retournent la largeur totale (`world-width`) ou la hauteur totale (`world-height`) du monde NetLogo.

La largeur est égale à `max-pxcor - min-pxcor + 1` et la hauteur est égale à `max-pycor - min-pycor + 1`. L'unité de mesure est le patch (quelle que soit sa taille en pixels).

Voir aussi `max-pxcor`, `max-pycor`, `min-pxcor` et `min-pycor`

## wrap-color

```
wrap-color number
```

La primitive `wrap-color` teste si la valeur de l'entrée *number* fait partie de l'ensemble des couleurs NetLogo, couleurs dont les numéros vont de 0 à 140 (140 non-compris). Si ce n'est pas le cas, `wrap-color` ajuste l'entrée numérique pour la faire entrer dans l'intervalle [ 0 , 140 [ .

L'ajustement est effectué en additionnant ou en soustrayant 140 (de manière répétée si nécessaire) au nombre fourni en entrée jusqu'à obtenir une valeur entrant dans l'intervalle des couleurs NetLogo. (C'est le

même type d'ajustement que celui qui est fait automatiquement si vous spécifiez un numéro de couleur trop petit ou trop grand pour la variable tortue `color` ou la variable patch `pcolor`.)

```
show wrap-color 150
=> 10
show wrap-color -10
=> 130
```

## write

`write` *value*

Cette commande affiche les données de l'entrée *value*, qui peuvent être un nombre, une chaîne de caractères, une liste, un booléen ou `nobody` dans la zone d'affichage du Centre de commande, non suivi d'un retour de chariot (contrairement à `print` et à `show`).

L'agent appelant n'est pas affiché devant les données, comme c'est le cas avec `show`. Cette sortie comprend toujours les guillemets autour des chaînes et est précédée d'un caractère espace.

```
write "hello world"
=>  "hello world"
```

Voir aussi `print`, `show`, `type` et `output-write`

# X

## xcor

xcor


`xcor` est une variable tortue préprogrammée. Elle contient la coordonnée x courante de la tortue. Vous pouvez modifier cette variable pour changer l'emplacement de la tortue.

Cette variable est toujours dans les limites du monde NetLogo, c'est-à-dire plus grande ou égale à `(min-pxcor - 0.5)` et strictement plus petite que `(max-pxcor + 0.5)`.

Voir aussi `setxy`, `ycor`, `pxcor` et `pycor`

## xor

*`boolean1`* `xor` *`boolean2`*

Cette primitive est le « ou exclusif » (« ou bien, ou bien »). Elle retourne `true` (vrai) soit si *`boolean1`* est vrai, soit si *`boolean2`* est vrai, mais pas quand les deux sont vrais.

```
if (pxcor > 0) xor (pycor > 0)
  [ set pcolor blue ]
;; le cadrant supérieur gauche et le cadrant inférieur droit sont bleus
```

# Y

## ycor

```
ycor
```


`ycor` est une variable tortue préprogrammée. Elle contient la coordonnée y courante de la tortue. Vous pouvez modifier cette variable pour changer l'emplacement de la tortue.

Cette variable est toujours dans les limites du monde NetLogo, c'est-à-dire plus grande ou égale à `(min-pycor - 0.5)` et strictement plus petite que `(max-pycor + 0.5)`.

Voir aussi `setxy`, `xcor`, `pxcor` et `pycor` ?

## ?, ?1, ?2, ?3, ...

```
?, ?1, ?2, ?3, ...
```

Ce sont des variables locales spéciales. Elles contiennent les entrées courantes d'un reporter ou d'un bloc de commandes pour certaines primitives (par exemple, l'élément courant d'une liste traitée par `foreach` ou par `map`).

`?` est toujours équivalent à `?1`.

Vous ne pouvez ni initialiser ni modifier ces variables, et vous ne pouvez pas les utiliser sauf avec certaines primitives qui sont pour le moment `foreach`, `map`, `reduce`, `filter`, `sort-by` et `n-values`. Voir les entrées de ces mots-clés pour des exemples d'utilisation.