Barebone Netlogo

The text is an adaptation of the NETLOGO 4.0 – QUICK GUIDE by Luis R. Izquierdo, available at https://ccl.northwestern.edu/netlogo/resources/NetLogo-4-0-QuickGuide.pdf

Agents

The NetLogo world is made up of agents. Agents are beings that can follow instructions. There are four types of agents:

1. *Turtles* are agents that move around in the world.
2. *Patches*. The world is two dimensional and is divided up into a grid (lattice) of patches. Each patch is a square piece of "ground" over which turtles can move.
3. *Links* are agents that connect two turtles. Links can be directed (from one turtle to another turtle) or undirected (one turtle with another turtle).
4. *The observer* doesn't have a location – you can imagine it as looking out over the world of turtles, links and patches.

Instructions

Instructions tell agents what to do. There are four characteristics that are useful to remember about them:

1. Whether the instruction is implemented by the user (procedures), or whether it is built into NetLogo (primitives). Once you define a procedure, you can use it elsewhere in your program. The NetLogo Dictionary has a complete list of built-in instructions.
2. Whether the instruction produces an output (reporters) or not (commands).
3. Whether the instruction takes an input (or several inputs) or not. Inputs are values that the instruction uses in carrying out its actions.
4. Comments are written after semicolon(s) and end with a carriage return:

```
; this is a comment
```

Variables

Variables are places to store values (such as numbers). A variable can be a global variable, a turtle variable, a patch variable, a link variable, or a local variable (local to a procedure). To change the value of a variable you can use the `set` command.

- *If a variable is a global variable, there is only one value for the variable, and every agent can access it.* You can make a global variable by adding a switch, a slider, a chooser or an input box to your interface, or by using the `globals` keyword followed by a list that enumerates them (at the beginning of your code), e.g., `globals [ I L C ]`

- *Turtle, patch, and link variables (and breeds).* Turtle, patch, and link variables can be built-in or defined by the user. They heve built-in variables like a color variable. Users can define turtle, patch and link variables (and also define breeds) for instance with turtles-own (breed-own) keywords.

- *Local variables*: A local variable is defined and used only in the context of a particular procedure or part of a procedure. To create a local variable, use the `let` command.

- *Setting and reading the value of variables.* Global variables can be read and set at any time by any agent. Every agent has direct access to her own variables, both for reading and setting. Sometimes you will want an agent to read or set a different agent's variable; to do that, you can use `ask`. You can also use `of` to make one agent read another agent's variable. A turtle can read and set patch variables of the patch it is standing on directly, e.g. `ask turtles [ set pcolor red ]`

Ask

NetLogo uses the `ask` command to specify commands that are to be run by turtles, patches or links. Usually, the observer uses ask to ask all turtles or all patches to run commands. Here's an example:

```
ask patches [ if (pxcor > 0) [ set pcolor green ]]
```

which has a natural interpretation. The statement uses lists extensively.

Lists

In the simplest models, each variable holds only one piece of information, usually a number or a string. The list feature lets you store multiple pieces of information in a single variable by collecting those pieces of information in a list. Each value in the list can be any type of value: a number, a string, an agent, an agentset, or even another list.

Constant lists

You can make a list by simply putting the values you want in the list between brackets, e.g.:

```
set my-list [2 4 6 8]
```

Changing list items

Technically, lists cannot be modified, but you can construct new lists based on variables, e.g.:

```
set newlist (list var1 var2 var3)
```

Map

Map is a reporter that returns a list. It takes as inputs the input list and a reporter, and returns an output list containing the results of applying the reporter to each item in the input list.

Agentsets

An agentset is a set of agents; all agents in an agentset must be of the same type (i.e. turtles, patches, or links). An agentset is not in any particular order. In fact, it's always in a random order . What's powerful about the agentset concept is that you can construct agentsets that contain only some turtles, some patches, or some links. These agentsets can then be used by ask or by various reporters that take agentsets as inputs. Once you have created an agentset, there are some simple things you can do:

- Use `ask` to make the agents in the agentset do something.
- Use `any?` to see if the agentset is empty.
- Use `all?` to see if every agent in an agentset satisfies a condition.
- Use `count` to find out exactly how many agents are in the set.

Synchronization When you ask a set of agents to run more than one command, each agent must finish before the next agent starts. One agent runs all of the commands, then the next agent runs all of them, and so on. The order in which agents are chosen to run the commands is random. You can make agents execute a set of commands in a certain order by converting the agentset into a list.

Tick counter

In many NetLogo models, time passes in discrete steps, called "ticks". NetLogo includes a built-in tick counter so you can keep track of how many ticks have passed. The current value of the tick counter is shown in the interface above the view (the black window where you can see patches and turtles).

In code, to retrieve the current value of the tick counter, use the ticks reporter. The tick command advances the tick counter by 1. The clear-all command resets the tick counter to 0. If you want to reset the counter to 0 without clearing everything, use the reset-ticks command.

Skeleton of many NetLogo Models

```
globals [ ...  ]
   turtles-own [ ...  ]
   patches-own [ ...  ]
   links-own [ ...  ]
   breeds-own [ ...  ]
   ...  to setup
   setup-global-variables
   setup-turtle/patch/link/breed-variables
   clear-all reset ticks setup-graphs
   ...
   end
   ...
   to go
   conduct-observer-procedure
   ...
   ask turtles [conduct-turtle-procedure]
   ...
   ask patches [conduct-patch-procedure]
   ...
   tick (updates graphs and starts the next loop)
```

Common Primitives

Turtle-related: `die, forward (fd), move-to, my-links, myself, nobody, of, other, other-end, patch-here, right (rt), self, setxy, turtle, turtle-set, turtles, turtles-at, turtles-here, turtles-on, turtles-own.`

Patch-related: `clear-patches (cp), distance, import-pcolors, myself, neighbors, neighbors4, nobody, of, other, patch, patch-at, patch-set, patches, patches-own.`

Link-related: `both-ends`, `clear-links`, `create-link-from`, `create-link-to`, `create-link-with`, `in-link-neighbor?`, `in-link-neighbors`, `in-link-from`, `is-directed-link`, `link`, `link-length`, `link-neighbors`, `link-set`, `link-with`, `my-in-links`, `my-links`, `other-end`, `out-link-neighbor?`, `out-link-neighbors`, `out-link-to`, `tie`.

Agentset primitives: `all?`, `any?`, `ask`, `ask-concurrent`, `count`, `in-radius`, `is-agentset?`, `max-one-of`, `min-one-of`, `n-of`, `neighbors`, `of`, `one-of`, `other`, `sort`, `sort-by`, `with`, `with-max`, `with-min`.

Control flow and logic primitives: `and`, `ask`, `foreach`, `if`, `ifelse`, `ifelse-value`, `let`, `loop`, `map`, `not`, `or`, `repeat`, `report`, `set`, `stop`, `startup`, `wait`, `while`, `with-local-randomness`, `without-interruption`, `xor`.

World primitives: `clear-all (ca)`, `clear-patches (cp)`, `clear-turtles (ct)`, `display`, `max-pxcor`, `min-pxcor`, `no-display`, `random-pxcor`, `reset-ticks`, `tick`, `ticks`, `world-width`, `world-height`.