NetLogo Coding Standards for Orrery Software
Note to File

Author: Garvin H Boyle
Date: 29 November 2014
Type: Note To File, R2

NetLogo Coding Standards for Orrery Software
Note to File

Table of Contents

References: 1
Model Names 1
The 'Interface' Tab – User Interface Design Elements 2
The Info tab 3
The Code Tab – Code Layout Elements 4
Major Sections 5
Comments and White Space 5
Procdure Structure 6
Variable names and Procedure names 6
Abbreviations of Commands 7
Debug code 7
Evolving Keywords and Idioms 9
Various Bits of Advice 10
Communicating Urgently with the User 11
Action keys 11

NetLogo Coding Standards for Orrery Software

NOTE TO FILE:
G H Boyle
21 June 2014
Revision 2.0
29 November 2014

References:

This document is not all entirely my own. While learning the NetLogo language, I looked for programming standards for NetLogo and discovered that there were no officially published standards that I could find for the most recent version of the language. On the other hand, I did find two unofficial documents that had been prepared by enthusiasts. One dated back to the days of NetLogo 2.x, and the other was written for NetLogo 4.x. I am learning NetLogo from the NetLogo 5.0.5 manual. So, the following document is written as a combination of my own programming style, as merged with advice from a couple of out-of-date and partially obsolete style guides. The two sources were:

1. Style Guide for NetLogo 2.x (2005)
   https://ccl.northwestern.edu/courses/mam2005/styleguide.htm
2. Summary Style Guide from Northwestern (2012)
   http://netlogo-users.18673.x6.nabble.com/Netlogo-coding-standards-td5001774.html
3. The "Info Tab" (2014)
   http://ccl.northwestern.edu/netlogo/docs/infotab.html

Model Names

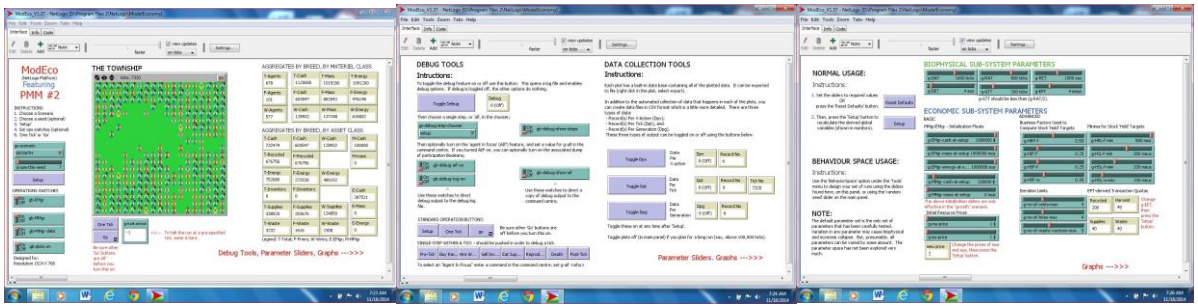
- Release numbers:
  - Assign models revision or release numbers (R3.2) or version numbers (V3.2) composed of three parts:
  - R or V, I tend to use either;
  - The digit(s) before the decimal is the public number, also used in the name of the model, which is incremented each time a new version of the model is released to the public domain; and
  - The digit(s) after the decimal are a private number, incremented each time a file is backed up during the development process.
- Name closely related models so they alphabetize together, even if it makes the names read oddly. So for example:
  - ModEco ThePMM R1.nlogo
  - ModEco Evolutionary R3.nlogo

The 'Interface' Tab – User Interface Design Elements

These ideas are somewhat in line with other's ideas of a NetLogo model interface, as described in the referenced documents, but I admit I have adjusted it somewhat for my own taste:

- Design the view for resolution 1024 pixels by 768 pixels. The right edge of the interface tab is open-ended, so a lot of extra stuff can be placed beyond the 1024 pixels, if needed, and the horizontal scroll bar will appear at the bottom to enable access to it.
- Always have:
  - A setup button;
  - A 'one tick' button; and
  - A 'go/stop' button.
- Place model name, basic operational controls and instructions to the left of the arena.
- Place 'display only' output controls and instructions, such as monitors and graphs, to the right of the arena.
- Optionally, have one or several user-modifiable switch controls that can be used to toggle the 'display-only' controls (graphs, monitors, textual message boxes) on or off, to enable speedy execution of the model when the tick-by-tick display is of little interest.
- Include onscreen notations with brief instructions for each user-modifiable control under the appropriate control, so users can decipher much of the interface with little effort, and without reference to the info tab.
- Leave key user-modifiable debug scaffolding controls in place when making a model public, but to the right of the arena. This includes the 'gb-debug-on' switch, described below.
- The area to the right of the main arena can be organized into clusters that have the appearance of being stand-alone views or panels. For example, a lot of white space between such apparent panels makes the proximity and commonality of the area not visible. For example, you can cluster debug tools in one panel, extraneous parametric controls in another, plots and charts in another, and so on. Each such panel can have its own version of the setup, one tick and go buttons.

Figure 01 – An example of apparent 'panels' from ModEco



The Info tab

The 'Info tab' is not a WYSIWYG editor (What You See Is What You Get). Rather, it is a markup editor, in which much of the information about what you want it to look like is in embedded codes (like hidden codes). You can switch between the markup editor and the WYSIWYG output of the markup editor by clicking on the 'Edit' button.

The orthodox description of how to use the info tab is at reference 3, above. Some of the markup language described at ref 3 (called, funnily, 'Markdown') does not seem to work entirely as described on my Windows 7 laptop. ?? The 'Markdown' markup language takes some of its cues from white space such as number of blank lines, or number of spaces at the beginning of a line, etc. And it is not guaranteed to be consistent in its interpretation across all platforms since it runs on a Java virtual machine within the various operating systems. So, if something funny happens, try deleting or adding a blank line or a space or two. For example, bulleted lists are handled differently depending on the number of blank lines between the lead-in paragraph and the first bullet, and depending on the number of spaces before the first dash. I strongly recommend reading reference 3. The following is gleaned from the first two references, with my own twist in a couple of places, and is meant to supplement reference 3 as alternative or practical advice. Some of it may be bad advice. So, here is my list of gleaned/personal opinions and advice.

- Model name – Don't put the model name at the top.
- First paragraph – The first paragraph only of WHAT IS IT? is what will show up in the Models Library dialog when the model is selected. So the first paragraph should be able so stand on its own as a brief and self-contained description of the model.
- Do not assume familiarity with NetLogo programming. For example, it shouldn't refer to "turtles" when it could just say rabbits or molecules or people or what have you. Similarly for "patches", "observer", and other NetLogo jargon. (This isn't necessarily a hard-and-fast, 100%-of-the-time rule... use your judgment.)
- Stick with printable ASCII characters – Only printable ASCII characters (letters, numbers, and basic typewriter-like symbols) should be in a model, but the Info tab is where you're most likely to see a violation of this rule. Non-ASCII characters, such as those with European accents, should be removed. This might seem like an easy rule to follow, but Microsoft Word automatically uses "smart" quotation marks, apostrophes, and hyphens that are angled differently at the beginning and end of a word. Pasting from MS Word into NetLogo can introduce these illegal characters. The only legal quotation marks in a NetLogo model are ' and ", and the only legal hyphens consist of one or more - characters.
- Line breaks – Remove manual line breaks from info tabs. NetLogo will do its own line-wrapping based on the width of the window (unlike StarLogoT and earlier NetLogos).
- Requiring a fixed width font – In NetLogo, the info tab markup editor is in a fixed width font, but when the text appears in the Model, or in the Models Library on the NetLogo web site, it will be in a non fixed width font. If there are any parts of the info tab that must be in a fixed width font to line up correctly, then put a "|" character at the beginning of each such line. Example:

```
|                              1%            +      -
```

```
|          H-A   + H O   <========>   H O   + A
|                    2                    3
```

- Spelling and grammar – Proofread the info text for spelling and grammar. (At a minimum, you should paste the text into a word processor and use its spell check feature.)
- Default slider settings – Make sure the default slider settings are sensible (and match what is described in the info tab, if slider settings are discussed there).
- Spaces and periods – Use two spaces after a period, not just one, to enhance readability.
- Trailing blank lines – Remove any extra blank lines from the end.
- Chooser, not choice – As of NetLogo 2.1, it's called a "chooser". (Before it was a "choice".) Some old models may still use the old name; this should be fixed.

Required sections – Be sure that each of the following sections are found in the Info Tab with the following formatting. There should be two blank lines between sections. There should be no section headers other than these. (I.e. no other lines that have all capital letters and no lowercase.) In the past it has been the practice to have subheadings in all capital letters, however, these should be changed to have lowercase letters.

- WHAT IS IT? – This section should give a general understanding of what the model is trying to explain or show.
- HOW IT WORKS – This section should explain what rules the agents use to create the overall behavior of the model.
- HOW TO USE IT – This section should explain how to use the model, including a description of each of the widgets in the Interface Tab.
- THINGS TO NOTICE – This section should give some ideas of things for the user to notice while running the model.
- THINGS TO TRY – This section should give some ideas of things for the user to try to do (move sliders, switches, etc.) with the model.
- EXTENDING THE MODEL – This section should give some ideas of things to write in the Procedures Tab to make the model more complicated, detailed, accurate to life, etc.
- NETLOGO FEATURES – This section should point out any especially interesting or unusual features of NetLogo that the model makes use of, particularly in the Procedures tab. It might also point out places where workarounds were needed because of missing features.
- RELATED MODELS – This section should give the names of related models in the Models Library.
- CREDITS AND REFERENCES – This section should contain the reference found below as well as any other necessary credits or references. If the model is based on published sources (or web sites), it's good to include reference (or URL) information here.

The Code Tab – Code Layout Elements

The code should be laid out in major sections, and within each section there should be a consistent approach to things like the order of definition of procedures, the way comments are used, or the way white space is used.

Major Sections

Always:

- Organize the code in sections, as follows:
  - A – identification of coders, purpose of application, abstract of operation of the model.
  - B – declaration and documentation of all global variables and breeds, and attributes of the agents (default and assigned attributes)
  - C – startup and setup procedures, initialize global variables and breeds
  - D – go procedure and operational sub-procedures, including pre-tick and post-tick hooks.
    - D_1 – pre-tick procedure, including checks for 'stop' conditions.
    - D_{n-1} – other procedures within a tick, in order of execution.
    - D_n – post-tick procedure, including calls to non-operational display updates, and reset-ticks.
  - E – drawing and maintenance and other non-operational procedures
- Mark major sections with 80-character Hollerith-length dashed lines before and after section title, which help to indicate reasonable length of lines of code. I do not use the special markers at characters 7 and 72.
- Call the main routines 'startup', 'setup' and 'go', connect the 'setup' and 'one tick' and 'go' buttons of the interface tab to these routines. Place these routines in sections C and D respectively, at the beginning of the section.

Here's an example of a section break:

```
;;-------------------------------------------------------------------------------|
;; SECTION B - INITIAL DECLARATIONS OF GLOBALS AND BREEDS
;;-------------------------------------------------------------------------------|
;;
```

Comments and White Space

- Blank Lines
  - Separate major and minor sections with two or one blank lines respectively.
  - Break long stretches of code (say 10 or more lines) into paragraphs with blank lines.
  - Conditional lines of code often have a few lines of initialization prior to the condition. You might initialize a count, or set a Boolean to establish a status for the conditional. Group such lines with the associated condition, and set off with a blank line.
- Comments
  - Use ;; for comments.
  - Use 80 character hollerith lines as code width indicators, and try to stay within them. For readability, or for long lines you want to be able to 'comment out' such as calls to debug dumps and traces, you may exceed this width from time to time.
  - The first comment within a procedure (a routine) identifies the type of agent that executes it.
  - Put in-line comments on all global or '-own' declarations.

  - Use comments to document hidden and implicit global variables in section B. This would be unnecessary for those who know NetLogo, but important for those (like me) who don't.
- White space
  - Standard tab stops are 2 characters.
  - Use indentation for all code within a procedure.
  - For each additional level of conditional code, use another tab stop.
  - If a code line goes on for more than one line, line up repeated structures, if possible.
  - If a command block is brief, put the brackets and contained code on one line after the Boolean conditional. The total, including all tab stops, should be less than 80 characters.
  - If a command block is not brief, put the two square brackets each on their own non-indented lines, and put the command on contained lines, indented one tab stop.
  - Line up consecutive in-line comments, if possible.

Procdure Structure

- Place a Holerith line and a brief description of the function before each function name
- Place an indication of the type of agents that can execute a function as the first line after the title
- See the section on names for procedure names

Here's an example of a procedure break:

```
;;-----------------------------------------------------------------------------|
;; The setup button is a standard button/procedure.
to setup
  ;; this routine is to be executed by the observer.
...  (Code goes here.)
end
```

Variable names and Procedure names

- Use all lower case in most names. Some variance is allowed for readability, but NetLogo does not recognize differences in capitalization. E.g. 'emgr' and 'EMgr' are viewed as identical variable names in NetLogo.
- Use '-' in names in place of spaces, capitals, underscores, dots, etc. E.g. name variables like this: this-customer; and not like this: thiscustomer or ThisCustomer or This_Customer.
- Choose names that more-or-less conform to natural language and explain the purpose of the variable. Noun phrases tend to work for numeric or textual variables. Statements of status tend to work for Boolean variables.
- The procedures 'startup', 'setup' and 'go', attributes of breeds, and locally-defined variables do not need a special prefix. All other procedures and variables should have prefixes as follows:
  - In 'Section D' where the 'go' function resides, it is broken into a set of serial steps which are the major sub-functions of 'go'. Each of these sub-functions should be named as 'do-*action*' or 'do-*function*', such as 'do-eat-supplies' or 'do-reproduction'.
  - Use f- as a prefix for all global functions except for the functions 'startup', 'setup' and 'go', and except for the major sub-functions of go, which start with do-.

suppressed or removed when the model is functioning well. The debug trace capability can be implemented like this:

- Use the 'show' command in combination with the 'word' and 'sentence' commands (or other string generation and concatenation commands) to place output of current variables in the command centre window.
- Use the 'file-show' command in combination with the 'word' and 'sentence' commands (or other string generation and concatenation commands) to place output of current variables in a debug log file that was previously opened for write. Note that an 'opened' file registers as non-existent until the first record is written.
- Use a global Boolean variable 'gb-debug-on' to invoke each such show command by bracketing it. Here are two examples
  - if(gb-debug-on) [show (word "msg1 – " variable)]
  - if(gb-debug-on) [show (word "msg1 – " variable1 "; msg2 – " variable2)]
- Be able to turn the debug trace feature on or off for a particular section or sub-section of code. For example, decide the sub-sections of code into which you want to place a debug trace, construct a chooser with an option for each such sub-section, then use the chooser options to turn the global debug variable 'gb-debug-on' on or off.
- So, the chooser picks the sections to be traced, the chooser options turn the global debug variable 'gb-debug-on' on or off at the appropriate times and places, and that global variable, in turn, enables the debug trace (i.e. show) commands in the section indicated by the chooser,
- Only debug output should go to the command centre. For other types of output instead, either use USER-MESSAGE, or add an output area to the interface and use OUTPUT-SHOW/TYPE/PRINT

Here is an example of the chooser options used to toggle gb-debug-on on or off.

```
;;-----------------------------------------------------------------------------|
;; D6 – reproduction procedures(s)
;;-----------------------------------------------------------------------------|
to do-reproduction

  ifelse( ( g-debug-step-chooser = "all" ) or ( g-debug-step-chooser = "reproduction" ) )
  [
    set gb-debug-on true
    show ""
    show word "Reproduction debug on; tick = " ticks
    show word "Control variables: g-RAT - " word g-RAT word "; g-GTT " g-GTT
  ]
  [ set gb-debug-on false ]

  ...  (Section code goes here.)

  if( gb-debug-on )
  [ show "Reproduction procedure completed." ]
end
```

And, here is an example of a debug message for an interim set of non-persistant and persistant variables:

```
if( gb-debug-on )
[
  show ""
  show word "    Transaction #: " ( no-of-transactions-completed + 1 )
  show word "        FRMR WHO#: " who
  show word "    Cash on hand - " cash
  show word "       Inventory - " inventory
  show word "    CONSUMER WHO#: " [who] of this-customer
  show word "    Cash on hand - " [cash] of this-customer
  show word "        Supplies - " [supplies] of this-customer
  show word "    Supply limit - " supplies-limit
  show word "Meus to purchase - " ( supplies-limit - [supplies] of this-customer )
  show word "   Dynamic quota - " dynamic-quota
]
```

Evolving Keywords and Idioms

It appears that NetLogo is an evolving language, and some keywords are invented to do things simply that used to take a lot of coding. When this happens, eventually, some of the old keywords are no longer needed, and are marked for eventual removal from the language. They hang around for a while until older applications have been revised and the old keywords have been removed. This 'lame duck' status is referred to as 'deprecated'. The life cycle of a keyword in an evolving language is extension (new keyword just added), standard, deprecated (on its way out), and obsolete (no longer supported). It seems that a few keywords are no longer recommended for use in NetLogo, and will probably eventually disappear. Avoid those. Here I include those I am aware of.

Also, sometimes an extension is invented to simplify a common programming idiom. An idiom develops when a task must be performed very often in almost every application. When a single keyword can replace an idiomatic phrase, an extension is added, and the old idiom can be replaced with instances of the new idiom.

A lot of the suggestions in the referenced documents seem to fall into a category of advice on using new keywords to replace old idiomatic phrases (OIs), or avoiding presumably deprecated keywords (DKs). Here is a table of what I have gleaned from the references:

S/N	OI or DK	Purpose	Outdated Idioms (OI) and Deprecated Keyword(s) (DK)	New keywords or idiomatic phrases
1	DK	Some old models use without-interruption to enforce serial execution. This is no longer necessary and should be removed.	without-interruption	Since NetLogo 4.0, ask is now serial rather than concurrent.
2	DK	Unknown.	ppd and ppu abbreviations Now obsolete?	??
3	DK	To cause the edges of the arena to connect in a torus.	-nowrap primitives Now obsolete?	Turn off wrapping in "Settings".
4	DK	To compute a heading, use towards.	if distance agent > 0 [set heading towards agent]	face agent

S/N	OI or DK	Purpose	Outdated Idioms (OI) and Deprecated Keyword(s) (DK)	New keywords or idiomatic phrases
5	DK	To compute a heading, use towardsxy.	if distancexy x y > 0 [set heading towardsxy x y]	facexy x y
6	DK	Generate random numbers	random-int-or-float Now obsolete?	Use random or random-float
7	DK	display no-display	were explicit calls	now implicit in 'tick'
8	DK	Reference to neighbouring cells.	nsum nsum4 Now obsolete?	neighbors neighbors4
9	DK	Creating plot pens.	create-temporary-plot-pen	Put pens in plot dialog.
10	DK	To send data to an output window.	report That usage is obsolete.	output-print, output-show, output-type, output-write
11	OI	Conversion of agentset to list.	VALUES-FROM ... [SELF] (or [SELF] OF ...)	For small to medium sized agentsets, use: SORT or SORT-BY
12	OI	To locate an agent randomly in the arena.	Complicated math with variables such as min-pxcor and world-width to place agents randomly within the arena.	Use random-xcor, random-pxcor. Use random-xcor, random-pxcor.
13	OI	To move an agent to the location of another agent.	setxy [xcor] of a [ycor] of a face a fd distance a	move-to a face a move-to a move-to a
14	OI	Fork variegated turtles.	sprout 1 [set breed wolves ...]	sprout-wolves 1 [...]
15	OI	Fork clone turtles.	hatch 1 [set breed wolves ...]	hatch-wolves 1 [...]
16	OI	A user-defined function for auto-start on loading.	to startup	Recommended to not be used.
17	OI	Determine outcome of a step in current heading.	dx and dy Deprecated.	patch-ahead, patch-here, patch-at patch-at-heading-and-distance patch-left-and-ahead patch-right-and-ahead
18	OI	Assign a shape to a turtle.	set shape "s"	set-default-shape "s"
19	OI	Turtle's reference to itself.	turtle who who != [who] of myself	self self != myself
20	OI	To set up a plot parameters prior to plotting.	Setup-plot procedure in the code tab.	Set the appropriate values in the plot edit dialog, in the: - Plot Setup Commands; and - Pen Setup Commands.

Various Bits of Advice

Be turtle-centric! E.g. avoid the use of 'set heading' unless the current heading is irrelevant:

- rewrite set heading (heading + 10) as rt 10
- rewrite set heading (random 360) as rt random 360

Avoid double negatives. E.g.:
Replace: not any? turtles with [color != red]

With: all? turtles [color = red]

Communicating Urgently with the User

If the model needs to tell the user that something has gone wrong, use user-message. If you use show or print, the user might not see it because they might have hidden the command center.

If the model wants to print messages from time to time as it runs, it should use print. If you do this, consider adding an Output area to the interface. Doing so has several advantages:

If the model is used as an applet (and every public model in our library can be run from our website as an applet), the Output area will be included, so the output will be visible. NetLogo applets don't have command centers.

The output will be visible even if the user has hidden the command center.

You can give the output a fairly small area of the screen if you want, whereas the command center can get quite large if more than a few lines of output need to be visible at once.

If you use an output area, consider adding a call to clear-output to the model's setup procedure. This might or might not be appropriate depending on whether you think the user might want to refer back to output from previous runs. When in doubt, don't include it.

Action keys

We can now assign keyboard equivalents to buttons in the Interface tab, so that (for example) pressing "G" will be the same as clicking on the "Go" button. However, you should refrain from assigning such action keys unless the model will significantly benefit. Games are a good example of where action keys can be particularly useful.