An introduction to NetLogo

Learning by example by building a simple model of a complex dynamical system

Introduction

NetLogo is a modeling environment based on a simple easy-to-learn language. The environment is particularly well suited to modeling systems of a number of elements which interact according to a set of rules – agents based models – which correspond to complex systems. Model parameters can be controlled by user-defined sliders, switches and buttons, to easily explore the model behavior.

In this class we will first take an overview of the NetLogo environment, and provide links to resources for continued learning, before moving on to begin learning the programing language. Working on the basis that a programing language is most easily learned by example we will move on to develop a model from scratch.

By first downloading the NetLogo environment for your system from here:

http://ccl.northwestern.edu/netlogo/

We will work simultaneously in the class to develop the model.

The order is as follows:

1) An overview of NetLogo
2) A brief survey of models from the library
3) The NetLogo Interface
4) Learning/reference resources
5) Synchronized fireflies; a video
6) Development of a simple model

Overview

NetLogo models are composed of agents (patches, turtles, links, and the observer), which interact according to a set of rules. The model is enacted in the NetLogo **world**, which is composed of a stationary two dimensional array of **patches**, which are one type of agent. Within this world another type of agent, **turtles**, are free to roam. Turtles may be connected by **links**. **The observer** can interact with any of the agents by invoking commands.

The interactions between the agents are governed by either **commands** (an action for an agent to perform) or **reporters** (a value for an agent to compute and then report back). There is a long list of built in commands and reporters called the **primitives**, but the user can also define their own, in which case they are called **proceedures**.

We have left this overview intentionally brief and incomplete working on the basis that the general principals and the specifics of programing in NetLogo are most lucid in examples.

The models Library

A good sense of the range and capabilities of NetLogo can be attained by browsing and tinkering with the models library. This can be found in the drop-down menu under File>Models Library, or simply Ctrl-M.

The NetLogo Interface

The NotLogo interface consists of a window with a drop-down menu bar, three tabs, and the command center.

The three tabs are as follows:

1) The Interface tab: where the model window is placed, along with space for placing controls such as buttons and sliders.
2) The info tab: where notes and instructions about the model may be stored
3) The code tab: where the code is written and stored

Commands may be typed into the command center; such commands are directly enacted without becoming part of the model code.

Learning resources and reference

NetLogo is supplied with a well-developed user manual which can be found here:

http://ccl.northwestern.edu/netlogo/docs/

Complete with tutorials, and reference guides.

Introduction to example model

We shall begin to learn NetLogo by building a simple model from scratch. This should provide enough grounding from which to develop an understanding of the whole language yourself using the above referenced resources.

A natural phenomenon that inspired disbelief from the initial reports of its observation is the synchronization of the flashes of the firefly. Individual fireflies behave according to relatively simple rules, yet somehow, without the orchestration of a conductor populations of fireflies (say in the same tree) manage to synchronize their flashes. This can be seen in the following video, for example:

http://www.youtube.com/watch?v=sROKYelaWbo

Being a system of simple agents interacting by simple rules, and resulting in non-trivial dynamics, this is a perfect system to study with NetLogo.

We shall model the behavior of the fireflies by giving them a natural cycle of flashing which has a particular length **cycle-length**, such the fireflies' clock runs from 0 to cycle-length, and then returns periodically back to 0. From the time 0 to **flash-length** the firefly is flashing. While the firefly is flashing it cannot be influenced by its neighbors, but otherwise it is able to observe its neighbors within a given radius. If more than a number, **flashes-to-reset**, of the fireflies immediate neighbors are flashing then the firefly will delay its flash by resetting its clock to **flash-length**.

We will next construct a model of this system from scratch.

Our first model: Fireflies in sync

- Start a new model by opening up NetLogo.

- Working in the interface tab, install a button and label it "setup". This button is initially colored red because it has no command associated with it. This is what we will do next.

- We use the "clear-all" to wipe the slate of our model clean, "create-turtles" command to make our population of turtles, and "reset-tick" to reset the model clock. Switch to the Code tab and include the following:

```
to setup
  clear-all
  create-turtles number
  [setxy random-xcor random-ycor  ;;put in random turtles
    set size 2
    ]
Reset-ticks
End
```

- Check the code by clicking the "Check" button. Note the error for the variable *number*.

- Go back to the Interface tab and Insert the slider called "*number*" (0 to 2000 in steps of 1)

- Recheck the code

- Return to the Interface tab and try out the "setup" button – observe your population of fireflies

- By right clicking on the world change the size of the world to 35 and Set patch size to 6

- Now give the turtles some properties by adding the code:

```
turtles-own
[
  clock      ;; each fireflies clock
  threshold   ;; the clock tick at which a firefly stops its flash
  ]
```

- Set the clocks for the turtles by including the line:

```
to setup
  clear-all
  create-turtles number
  [setxy random-xcor random-ycor
    set clock random (round cycle-length)
    set size 2
    ]
Reset-ticks
End
```

- Next we want our fireflies to fly. Define a move function:

```
to move ; turtle procedure
  right random-float 90 - random-float 90
  forward 1
end
```

- We have the appropriate procedure to move the turtles defined now. We next mace a main procedure called “go”. Go the the Interface tab and put in a “go” button

- Then add the following go function to the code, which calls our “move” procedure, and advances the model clock by one tick:

```
to go
  ask turtles [
    move
  ]
  tick
end
```

- Try pressing the go button – we see the fireflies move a step

- By changing the properties of the “go” button (right-click and toggle “forever”) we can make the go procedure repeat. In order to watch at a more comfortable speed we can also change the speed slider too.

- We want to control the length of the fireflies natural cycle. In the interface tab Insert a slider for the length of the cycle “cycle-length” (5-100, value 10)

- Each firefly must keep track of its phase. To do this we Insert the increment clock function

```
to increment-clock ; turtle procedure
  set clock (clock + 1)
  if clock = cycle-length
    [ set clock 0 ]
end
```

- include the increment-clock procedure in go

```
to go
  ask turtles [
    move
increment-clock
  ]
  tick
end
```

- Set the threshold to be equal to the flash-length and inset the slider for flash-length (1 - 10, start at 1)

```
to setup
  clear-all
  create-turtles number
  [setxy random-xcor random-ycor
   set size 2
   set clock random (round cycle-length)
   set threshold flash-length
    ]
End
```

- We may or may not want to be able to observe the fireflies which are nor currently flashing so we Insert a switch for a variable called “show-dark-fireflies?”

- We make a procedure for coloring the fireflies according to weather they are flashing or not, recolor,

```
to recolor ; turtle procedure
  ifelse (clock < threshold)
    [ show-turtle
      set color yellow ]
    [ set color gray - 2
      ifelse show-dark-fireflies?
        [ show-turtle ]
        [ hide-turtle ] ]
end
```

- put the recolor function into the go procedure

```
to go
  ask turtles [
    move
    increment-clock
  ]

  ask turtles [
    recolor
  ]

  tick
end
```

- Click the setup button, then try pressing go. You should see your population of fireflies flashing now.

- Now want to let the fireflies influence each other (model gets interesting). Ask to count how many flashing fireflies are within sight (a given radius) and if that number is above a threshold (flashes-to-reset) then we reset the clock.

- First we need to put in a slider for flashes-to-reset (1, to 3, start at 1)

- Then add the function which allows the firefly to observe it's neighbours

```
to look ; turtle procedure
```

```
  if count turtles in-radius 1 with [color = yellow] >= flashes-to-reset
    [ set clock threshold ]
end
```

- Update the go procedure to include the “look” procedure:

```
to go
  ask turtles [
    move
    increment-clock
    if (clock >= threshold)
      [ look ]
  ]
  ask turtles [
    recolor
  ]
  tick
end
```

- Finally make a plot of the number of flashing fireflies against time. This is done in the Interface menu by inserting a plot. Then the settings for the plot are made by right-clicking and selecting “Edit”.

Name: flashing fireflies
Xaxis label Time, X min 0, Xmax 100
Yaxis label Number, Ymin 0, Ymax 1500

Plot setup commands set-plot-y-range 0 number

Set color to red

Pen name “flashing”
Pen update commands “plot count turtles with [color = yellow]”

- Try out the model. Investigate the effects of the different parameters.
- This simple model can be extended in any way you like. What follows is an assignment to extend the model by allowing variable cycle-lengths.

Assignment

By building upon the model made in the class do the following:

1) Allow the fireflies to have their own cycle length, and set it to be random in a range which is controlled by a slider
2) Allow the fireflies to detect the number of fireflies in a radius of 1 which have a cycle length which is longer/shorter than theirs, and make a rule for adjusting the cycle-length accordingly
3) Display the results in an appropriate way
4) How does synchronization work in this case?