GEOG 5160/6160: Spatial modeling

Lab 01: Cellular Automata with Netlogo

Introduction

In this lab, you will build Conway's Game of Life using the software **Netlogo**. Netlogo is a widely used, java-based software for modeling complex systems. It is mainly designed for agent-based models, but can also simulate cellular automata.

We will discuss the structure and syntax of Netlogo in more detail, but for now there are two important things to understand:

- Netlogo considers a landscape of 'patches', comparable to a raster layer. Each cell or 'patch' can have a number of attributes, and according to those attributes (and those of the neighboring patches), we can change the value of the attributes of a given patch
- Netlogo can have 'global' variables that can be used to influence the behavior of the model over all patches

Ruleset

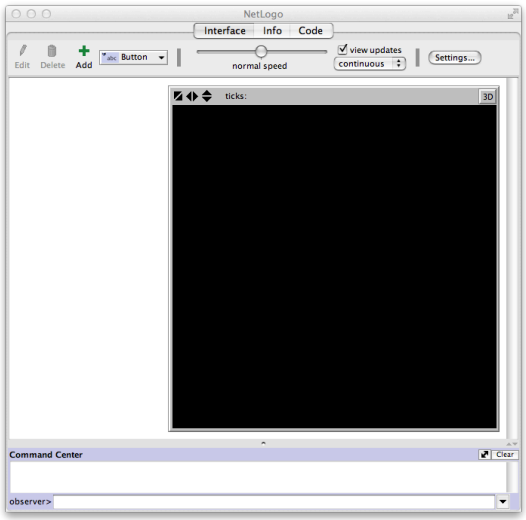
The Game of Life as described by Conway has two basic rules
- If a dead cell as three neighbors, then it becomes alive
- If a live cell has less than 2 or more than 3 neighbors it dies

This simple rule set makes this one of the easiest models to set up and run. To do this, we will first program the overall structure of the model, then fill in the details. This allows us to get a working flow for the model, rather than getting lost in the details. As we start to add further parts, we will run tests to make sure that the model is developing correctly.

Setting up model structure

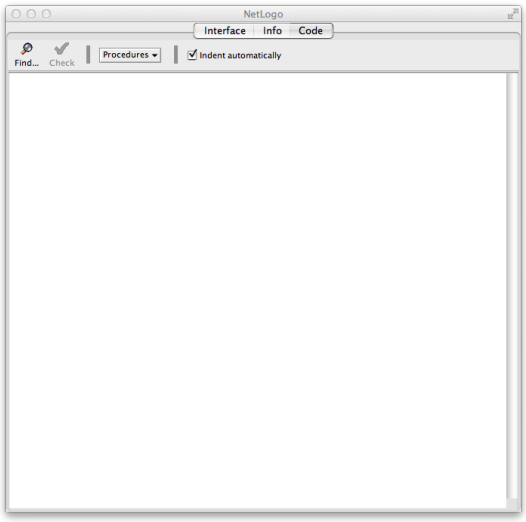
Start Netlogo from the Start menu (Windows) or Applications menu (Mac). This will open the Netlogo window on the interface tab as shown below:

Netlogo in OSH labs: Netlogo may **not** show up in the start menu on the computers in the OSH labs. If so, start the program by browsing to 'C:\Program Files (x86)\Netlogo 5.0.4', and clicking on 'Netlogo 5.0.4'.



From the File menu in Netlogo, create a new Netlogo program, and use File/Save save this as “myGameOfLife.nlogo” in an appropriate folder. If you close Netlogo at any point, you can reopen this model, by finding the model file in this folder, and double-clicking on the icon.

Now we start by entering the state (general) variables. Click on the “Code” tab (this is the “Properties” tab in older versions of Netlogo), which should change the Netlogo window to look like the following image:



Now insert the following code:

```
globals [ ]
patches-own [ ]
```

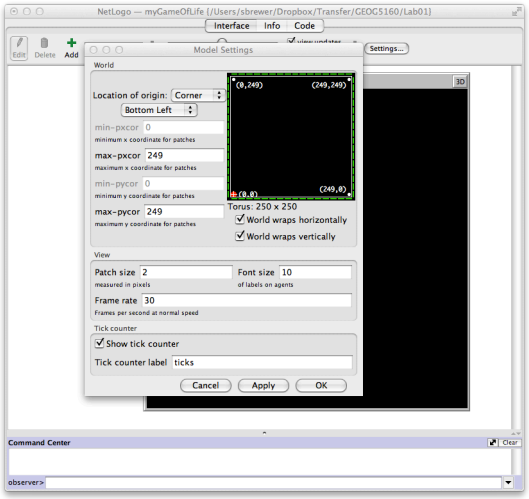
These code sections will allow us to define variables for the entire system (global) and the landscape (patches). Click the check button to make sure that the syntax is entered correctly (it is worth clicking this button after every update).

The patches require two variable to store their status (live or dead), and the number of living neighbors (to allow the rules to be assessed). Enter the following code:

```
patches-own [
    living?
    live-neighbors
]
```

The '?' after the variable 'living' tells Netlogo that this is a Boolean (0/1) variable.

Now we need to set up the spatial extent of the model. This is done from the "Interface" tab, so click on this, then on "Settings", which will open the following window:



We will use a landscape of 100x100 pixels

1. First change "Location of origin" to "Corner" and "Bottom left".
2. Then change the maximum 'x' (*max-pxcor*) and 'y' (*max-pycor*) coordinates to 249 (note that the origin is zero)
3. Now uncheck the two world wrap boxes (with this on, the left side of the region is wrapped to the right side and so on)
4. Lastly, change the patch size to 3 (or 4) – this shrinks the size of the visible View window so that it will fit better on your screen (you may need to resize your window as well)
5. Click "OK" to save your changes

Finally, we will add the setup routine into the code. Add the following lines, after the code you have already entered:

```
to setup
    ca
    ask patches
        [

        ]
    reset-ticks
end
```

Click the [Check] button to make sure that the syntax is entered correctly

Netlogo procedures

Any procedure in Netlogo starts with the keyword “to” and ends with the keyword “end”. The next command “ca” is short for “clear-all” and removes all existing data stored in the model. This is nearly always the first command used in a model. The command “reset-ticks” is nearly always the last command, and sets the model clock back to zero to allow for a new model run.

The command “ask patches” will be used to set initial values for each landscape patch – when Netlogo runs this, it will loop through each patch in a random order, then carry out any commands within the brackets. The “ask” command can also be used with turtles, as we will see later.

Initializing the model

We want to initialize the model with a fraction of the cells set as ‘alive’. The first thing we will need is a variable that will control this fraction. We will call this ‘initial-density’. As this will affect all patches, we need to set this as a global variable by adding it to the globals section:

```
globals [ initial-density ]
```

Our first model will use a value of 25 or 25% of the patches will be alive. Note that we don’t set a value of this variable here, but instead do this at the end of the “setup” procedure, using the “set” command. This command is called a ‘primitive’ - a group of predefined commands in Netlogo (Go here for other primitives):

```
set initial-density 25
```

Now we can use this value to set each patch to alive or dead. Here we use the “ask patches” command that we have added to the “set-up” procedure. This command will loop through each patch on the landscape in random order and allow us to interrogate each one. Add the following code between the ‘[’ and ‘]’ following “ask patches”:

```
ask patches
    [
    ifelse random-float 100.0 < initial-density
        [ cell-alive ]
        [ cell-death ]
    ]
```

We use the “random-float” command to generate a random number between 0 and 100.0 (“random-float 100.0”) and compare this with the variable controlling the density. The “ifelse” command allows to to choose one of two outcomes depending on this comparison:

- If the random number is less than the density value (25), we set the cell to alive

- If the random number is greater than the density value, we set the cell to dead.

Now click on the [Check] button. This time, Netlogo should complain about there being nothing called “cell-alive defined”. This is because there is no existing command in Netlogo called this (or cell-death), so we will need to create them.

Use the following code to add a skeleton procedure (i.e. with no working code) for these two commands after the end of the “setup” procedure. We will fill in the details later:

```
to cell-alive

end

to cell-death

end
```

Click [Check] to make sure everything is ok.

Cell-alive procedure

In the cell-alive procedure, we need to do two things: a) set the variable ‘living?’ to true (remember this is a boolean 0/1 variable); and b) set the color (“pcolor”) of the patch to differentiate it from dead cells. We do both of these using the “set” command:

```
to cell-alive
    set living? true
    set pcolor green
end
```

Cell-death procedure

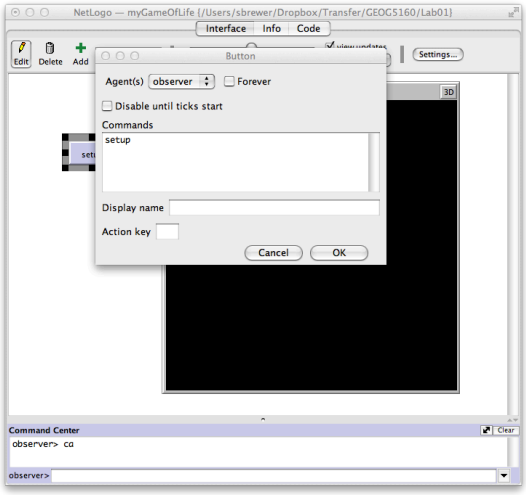
In the cell-death procedure, we simply need to do the opposite of the cell-alive procedure:

```
to cell-death
    set living? false
    set pcolor black
end
```

Setup button

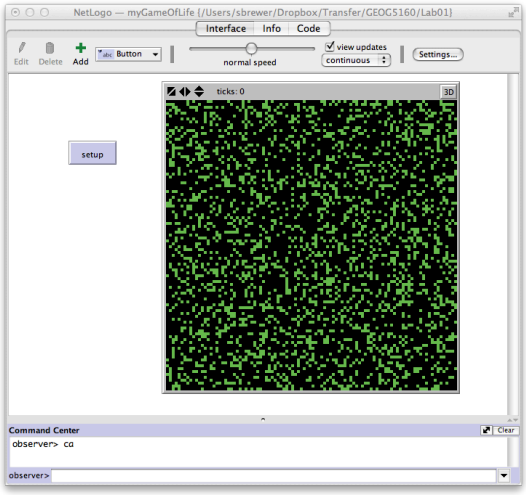
The final stage is to include a button on the interface tab to run the setup procedure. Click on the interface tab, then press the [Button] button, select “Button”, then place the cursor somewhere to the left of the View window. Click the

mouse button and a button will appear, together with a dialog box:



Type “setup” in the Commands field, and click “OK”

Now this button has been linked to the setup procedure, you can click on it, and a landscape should be produced populated with live and dead cells.



Go back to the code and change the value for “initial-density”, and try re-running the setup procedure using this button.

Model flow

We now need to code the model to go through a series of steps where it will update the status of each patch according to the number of neighbors each one has (following the ruleset described above).

This procedure is typically called “go”, so we will add a blank procedure with this name to the end of our code:

```
to go
    tick
end
```

The primitive “tick” controls the time counter - this simply advances it by one. Click on the [Check] button.

As with the “setup” procedure, we need to add a control to the interface to start the model running. Go to the interface tab, and add a “go” button to link to the go procedure. Follow the same steps used to create the setup button. Once done, click [setup], and you’ll see the number ticks (in the bar above the View window) increase by one each time you click [go].

Adding the ruleset

We are now ready to add the final section to the model, which will, at each time step, update the status of each patch according to it’s neighbors. We will need to do this in two steps.

First, we use the “ask patches” command to cycle through each patch and calculate the number of live neighbors. Add this code to the “go” procedure (before the “tick” command"):

```
  ask patches [
    set live-neighbors count neighbors with [living?]
    ]
```

A number of new primitives are used here. As the “ask” command loops through, the command “neighbors” returns the list of neighboring patches for the patch under consideration. The command “with” selects only neighboring patches whose variable “living?” is set to true, and the command “count” counts these. Finally, the command “set” assigns this number to the variable “live-neighbors” for that patch. Click [Check].

Second, we loop through the patches again, but this time, we update the status of the patch, according to the value of “live-neighbors”. Add the following code (again, before the “tick” command):

```
ask patches
[
ifelse live-neighbors = 3
    [ cell-alive ]
    [ if live-neighbors != 2
        [ cell-death ]
    ]
]
```

Again, we use the "ask patches" command to loop through the cells. We then use the "ifelse" command with the live-neighbors variable. If this is equal to three then the first set of brackets is chosen and the "cell-alive" procedure is run, setting that cell to be alive (whether it was dead or not before). If it is equal to any other value, then the second set of brackets is executed. This is the first rule: cells with three neighbors are set to alive.

In this second set of brackets, we have a simple "if" statement. Here if the number of neighbors is not equal to 2, then the cell is set to dead. This is the second rule: cells die if they have less than 2 or more than 3 neighbors (remember we tested for 3 neighbors in the first loop).

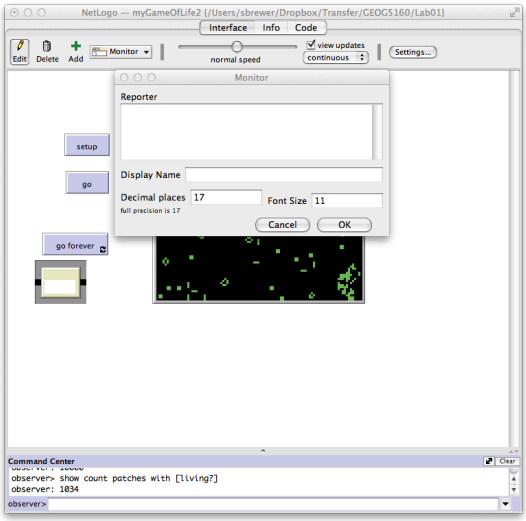
Click [Check], and everything looks good, return to the interface tab, and setup and run your code. Keeping clicking [go] to advance the model. You should see cells die and get replaced, as well as some emergent patterns forming.

Running multiple iterations

Rather than continually clicking [go], it would be easier to let the model run and decide when to stop. Add another button to the interface, enter "go" in the Commands section, click the Forever tickbox and make the Display name "go forever". Now redo the setup and click this button. When you want to stop, simply click it again.

Tracking cell counts

It is very easy in Netlogo to record the values of certain variables during the run of the model. On the interface tab, click on the [Button] button and select "Monitor". The following window will appear.



Add the following code into the "Reporter" window:

```
count patches with [living?]
```

This counts the number of patches in the model that have their variable "living?" set to true (i.e. all that are alive). Set the Display name to "Total alive". Now click [OK] and return to the interface tab. Click [setup] then [go]. You'll see

the number of live cells change as the model runs.

Assignment 01

Use a word document to record your answers and output. Assignments should be submitted to Canvas by midnight on Monday 3rd February.

1. Briefly describe the ruleset underlying this model (1)
2. Change the extent of the landscape to 250x250 and the color of the live cells to blue. Setup and run the model for several time steps. Again, take a screenshot of the output. (1)
3. Change the initial-density of the alive cells to 10%. Once you have done this, check and setup the model. Provide a screenshot of the output (right-click on the world-view window and select "Copy view"; then paste this into a word document). (1)
4. Now run a series of experiments in which you vary the initial-density from 10 to 90 in steps of 10. For each value, run the model and record the number of live cells after 20 time steps. Make a plot in Excel (or similar) showing how this number varies with the value of density, and briefly describe the changes. (2)

Simon Brewer 01/26/2014