GEOG 5160/6160 Lab 05

Simon Brewer

1/27/2018

## Lab 05: Fire spread model

## Introduction

In this lab, we will take some of the ideas of cellular automata discussed in class and in the previous lab, and use them to build a simple model of the spread of fire.

The basis of this model is that fire is a diffusive process that spreads between a burning cell and a non-burnt cell. Cells can only catch fire if they have fuel (vegetation), so empty or bare cells can provide a barrier to fire spread.

We will start by building a model with a single parameter controlling the density of vegetation, then look at the influence of this parameter on fire spread. We will then expand the model to include stochastic ignition and asymmetry to represent wind effects.

### Objectives

1. Understand how to code a simple diffusive process in Netlogo
2. Include stochasticity into a cellular automata model
3. Include an asymmetric neighborhood structure

## Model setup

Start Netlogo from the Start menu (Windows) or Applications menu (Mac). From the File menu in Netlogo, create a new Netlogo program, and use File/Save save this as *firespread1.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.

To start with, we will set the global variables. These are variables that are linked to the whole system, not any single patch. The first section of a model (the globals section) defines these. For now, we have a single global variable density, which will control the ratio of vegetation to bare ground, so add the globals section, with this variable:

globals [   
 density  
]

We will also define variables associated with patches in the section patches-own. Add a variable which defines whether or not it is currently burning. The ? at the end of the variable name defines this as a Boolean variable, which can only be true or false.

patches-own [   
 burning?  
]

Click the [Check] button to make sure that the syntax is entered correctly (remember to click this button after every update).

### Setting up the landscape

We need to define a setup procedure to populate the landscape with vegetation. First we will define the landscape parameters. Click the [settings] button in the interface tab, and turn off world wrapping in both horizontal and vertical directions. Leave the size of the grid as the default settings for now.

Add the following setup procedure, to initialize the model:

to setup  
 ca  
 set density 50  
 reset-ticks  
end

We can represent vegetation/non-vegetation (and ultimately fire) simply by using colors, so we’ll start by populating the landscape with green trees. Add the following to the setup procedure after the previous code:

ask patches   
[  
 if random-float 100.0 < density  
 [ set pcolor green ]  
]

This is basically the same routine that we used in the previous lab to initialize the game of life. We will be using this type of routine several times during this lab, so it is worth exploring what it does in a a little more detail here.

* The first line ask patches returns a list of all the patches/cells in the current model. It then runs the commands between the [ and ] on every patch, one at a time.
* The if statement tests a condition, and then runs the command between the second set of brackets if that condition is true
* The condition to be tested is whether a random number chosen by the command function random-float between 0 and 100 is less than the value of the density variable
* If true, then we set the color of the patch to green, which will represent vegetation cover in our model

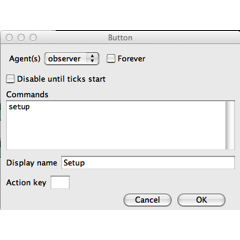
Finally, we need to ignite at least one cell to begin the model. To start with, we will simply pick the center cell (at patch coordinate 0,0) and ignite this. Add the following code to the setup procedure:

ask patches with [pxcor = 0 and pycor = 0]  
 [ ignite ]

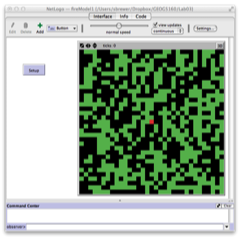
Netlogo will now complain that there is no ignite procedure, so add the following, after the ‘setup’ procedure:

to ignite  
 set pcolor red  
 set burning? true  
end

Now hit check to make sure everything so far is in place. If there are no errors, go back to the interface tab, and add a setup button to the interface. Click on [Button] and select Button, then add ‘setup’ to the commands to link this button to the setup procedure.



Now click this button and you should see something similar to the following:



## Model run

### Basic diffusion

With the model setup, we now need to plan how the diffusion will take place. Our first attempt will simply assume that if a vegetated cell is next to a fire, it will ignite. To do this, we use the ‘ask’ command to loop through all patches, then investigate the status of their neighbors. Start by adding a ‘go’ procedure, which simply increments the time interval using the command ‘tick’:

to go  
  
 tick  
end

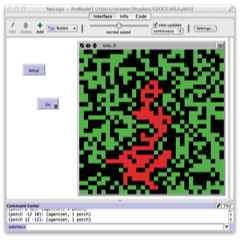
In Netlogo, time is not recorded in any specific unit (months, days, minutes), but as a series of ticks, a simple counter which is advanced by one by the command tick. Each tick should represent the base unit of time for the model.

The most efficient method will be to only investigate patches that are currently burning. To do this, we can include a conditional statement in the ask command, which will select only the patches where pcolor is red. Add the following to the go procedure.

ask patches with [pcolor = red]  
[  
 ask neighbors4 with [pcolor = green]  
 [  
 ignite  
 ]  
]

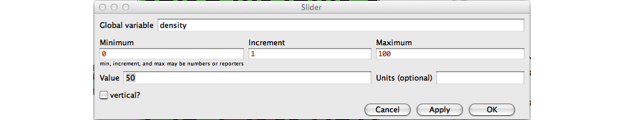
Here we use ask patches to get the list of all the patches that are colored red. Each of these patches is then passed, one at a time, to the next ask statement. We use the neighbors4 command to get the list of the four neighboring patches to the north, west, south and east (the command neighbors will return all 8 neighbors in a 3x3 window). If any of these are green (i.e. have vegetation), we ignite them.

Check this code is ok, then add a go button to the interface and link it to this procedure. Try running the model, and you should (after a few iterations), get something like the following:



### Modifying the basic model

Netlogo comes with a set of buttons, sliders and switches to control model behavior. We will now add a slider to allow us to set the value of density, controlling the ratio between vegetation and bare ground. Click on the [Button] button on the interface, and select slider from the drop-down menu. Click on the blank space to the left of the interface (where the setup and go buttons are), and a new slider will be created. The following dialog window will open, allowing you to set the parameters of this slider:

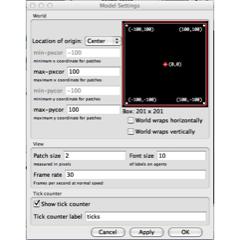


Set the variable to density, the minimum value to 0, increment to 1, maximum to 100 and the value (default value) to 50. Click [OK].

Netlogo will now tell you that density is already defined. The problem arises here as the slider declares its own variable, which we have previously declared in the model code. Simply remove the line density from the global section, and the line in the setup procedure that sets density to a given value. Click [Check] and now try running the model after changing the density.

### Increase landscape size

Use the settings menu to increase the size of the landscape to 200x200. Keep the location of origin in the center, and set the max pxcor and pycor to 100. Remember to decrease the “patch size” to fit the window in on the screen.



### Stopping the model

We’ll set the model to stop after 100 iterations, to help compare between different parameters settings. The simplest way to do this is to check the value of ticks (the time intervals) during each iteration. Add this code to the start of the ‘go’ procedure:

if ticks >= 100 [ stop ]

Right-click on the [go] button, and check the box ‘run forever’. Now spend a little time running the model with different settings for the vegetation density, to see how sensitive the model is to changes in this parameter.

## Including stochasticity

The fire model described in class used stochastic (random) parameters to control the spread of fire. We will now add these to our existing model to simulate the ignition and extinction of fires as random events. This will involve changing some of the basic structure of the original model, so it is advisable to make a copy of your working code, and modify that.

### Fire extinction

Fires will become extinguished as the fuels run out. We could include this in the model as a counter (i.e. each fire lasts for a set number of time steps or ticks), or as a random event. We will do the following, as it allows us to easily experiment with the probability of extinction.

Add a new slider to the interface, controlling a variable extinction. Set the range to between 0 and 1 with a step of 0.05. Set the default to 0.25.

Now add a new section of code to the go procedure, following the existing ask patches function. Note that we do this after the ignition loop, so all ignitions will have taken place.

ask patches with [pcolor = red]  
[  
 extinguish  
]

And add a new procedure extinguish.In here, we will set the color of the patch to dark red. Colors in Netlogo can also be given by a number, shown on [this chart](http://ccl.northwestern.edu/netlogo/docs/programming.html#colors). We will use 12 (dark red).

to extinguish  
 if random-float 1.0 < extinction  
 [   
 set pcolor 12  
 set burning? false  
 ]  
end

Check your code and try running the model. You may need to increase the vegetation density to clearly see how the spread has changed.

As the color has been changed, then these are no longer considered as ‘burning’ and are not picked up in the ignite conditional statement.

### Fire ignition

We’ll start be introducing a simple set probability for ignition. Add ignition as a variable declared in the globals section, and set the value of this to 0.75 in the setup procedure.

set ignition 0.75

We now need to include a random test, as in the extinction procedure, to test for ignition. We could add this to the ignite procedure, but that would affect the initial ignition event, which we want to keep constant. Instead, we will add this into the go procedure, by changing the ‘ask neighbors4’ statement

ask neighbors4 with [pcolor = green]  
[  
 if random-float 1.0 < ignition   
 [ ignite ]  
]

Check and run the model. Try playing with the settings for extinction and density to get some idea about how the simulated fire spread changes.

## Measuring fire spread

So far, we have been using the model to show visually how the parameters affect spread. It would be more useful to measure a value that indicates the overall spread, for example the number of cells that have burned, which we can do by calculating the percentage of the vegetation that has burned. We will need the initial number of vegetated patches (initial-trees), and a variable that records the number of burned patches (burned-trees). These are both global variables, so add them to the global section.

The initial number of trees can be calculated in the setup procedure, following the section in which a proportion of the cells have their color randomly set to green.

set initial-trees count patches with [pcolor = green]

The set command assigns a values to a variable - here the count (or sum) of patches that are green.

Now we need to calculate the number of burnt patches. First, set the value of the variable to zero in the setup routine, just after the previous line.

set burned-trees 0

Now we need to increase the value of this variable by one, every time a patch is ignited. So add the following to the ignite procedure:

set burned-trees burned-trees + 1

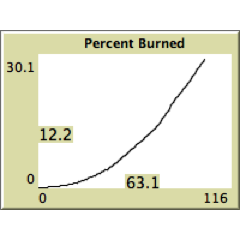
Add a new type of procedure to calculate and *report* the percentage. This procedure, specified by to-report, creates a variable that can be used in plots and other output. This will create the variable ‘percent-burned’:

to-report percent-burned  
 report (burned-trees / initial-trees) \* 100  
end

Now add a plot to the interface to show the change in this value over time. Give the plot a suitable name, and change the pen update command to

plot percent-burned

Click [Ok], then return to the interface, and re-run the model. You should obtain figure similar to this, where the x-axis is the number of ticks, and the y-axis the percent burned:



## Changing the neighborhood structure

We will now change our model again, this time to include the effects of prevailing winds on fire spread. This will require a set of modifications to the basic structure of the models, so save your current model, then make a copy of it and work on that.

The basic idea is that for any given cell that contains fire, the probability of ignition of the neighboring cells will be greater in the direction of the prevailing wind. In order to do this, we need to be able to set separate probabilities for neighbors in different directions, and do our random ignition test according to these different values.

### Setting directional probabilities

We start by including new variables to represent the probabilities for the four cardinal directions (N, W, S, E). Those are global, so add the following definitions to the globals section

ignitionN  
ignitionW  
ignitionS  
ignitionE

Then set values for each of these in the setup procedure, e.g. set ignitionN 0.75. We’ll start by simulating a prevailing westerly wind, with probabilities as follows:

* North: 0.75
* West: 0.25
* South: 0.10
* East: 0.25

### Accessing neighboring patches

Having set the probabilities, we now need to use these to simulate ignition around any burning cell. Previously, we used ‘ask neighbors4’ to get the list of neighbors around a given patch. However, this list of neighbors is returned in a random order, making it difficult to use with a list of probabilities. Instead, we will query the patches individually, starting with the patch to the north of a cell that is burning. To do this, we use the at-points command, which selects a patch (or patches). This requires that we give it the relative coordinates of the patch we want to test, i.e. relative to the burning patch. Coordinates are given between brackets: [x y], where positive x indicates east, negative x indicates west, positive y indicates north and negative y indicates south.

For example at-points [[0 1]] selects the patch located zero steps to either east or west and one step to the north; at-points [[-1 -1]] selects the patch located one step to the west and one step to the south.

Start by deleting the existing code from the ‘go’ procedure that is used to simulate ignition. This should look something like this:

ask patches with [pcolor = red]  
 [  
 ask neighbors4 with [pcolor = green]  
 [  
 if random-float 1.0 < ignition [ ignite ]  
 ]  
 ]

Delete everything from ask neighbors4 to the second-to-last ]. Now add the following code to simulate ignition in the north patch:

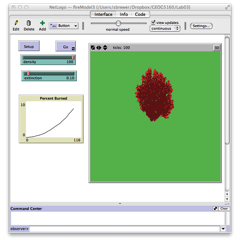
; Ignition of north patch  
ask patches at-points [[0 1]]  
[  
 if pcolor = green  
 [  
 if random-float 1.0 < ignitionN  
 [ ignite ]  
 ]  
]

In this block of code, the first line is simple a comment describing what this section does. The second line then finds the patch to the north. The first if statement looks to see if it is occupied by unburned vegetation, and the second if statement does a random test against the ignition probability for the north, igniting the patch is the test is successful.

Check your code, and return to the interface. If you now setup the landscape with 100% vegetation density and run the model, you should see the fire moving in a straight line to the north.

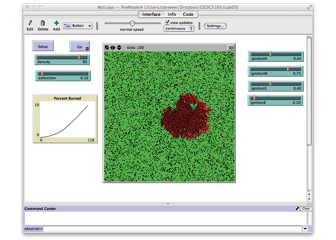
If that worked, add a similar section for the patches to the west, south and east of the burning patch. You can copy and paste the code for the north - but remember to change the patch location and the ignition variable.

When you have all this set and added to the model, check, setup and run the model. You should obtain something similar to the following image (with 100% vegetation density).



### Adding sliders

**This part is optional.** Try adding sliders for the four ignition variables to allow you to more easily experiment with different parameter settings. Remember to remove the definition of the variable from the global list, and the code in the setup that sets the value of the variable. You may need to expand the size of the Netlogo window to fit all these in:



## Exercise

Use a word document to record your answers and output. Assignments, to include both the word document and .nlogo script file, should be submitted to Canvas before class on Monday, XXXXXX XX. Ensure your assignment has been saved using the following naming convention: Lab05\_lastname\_script and Lab05\_lastname\_report.

1. Add a second plot to the interface, showing the number of patches that are actively burning at each time step. Remember, patches with pcolor = red are currently burning. [1]
2. Change the go procedure, so that instead of stopping the model run at 100 ticks, the model stops when no more patches are actively burning. Remember that the command count allows you to get the number of patches that have a certain condition [1]
3. Change the model so that it simulates ignition of the ordinal neighboring patches (NW, SW, SE, NE), as well as the cardinal directions (N, W, S, E). You can either set the probabilities directly using sliders, or calculate the probabilities as an average (i.e. the probability for the NW will be the average of the probabilities for N and W) [1]
4. Describe how you would extend the model to include either of the following:
   1. The model currently has a uniform probability for the extinction of fire in each patch, but it would be useful to adjust this according to the amount of fuel (vegetation) in each patch. First describe how you might incorporate this in the model, if you were given an image that described the fuel availability in each patch on a scale of 0 (bare ground) to 1 (maximum fuel availability). Second, describe how this might be used to adjust the extinction rate. Assume that fuel availability is inversely correlated with extinction, and that fire will reduce the fuel over time [2]
   2. Using the examples given in class for long distance dispersal, describe how you might incorporate long-distance ignition events. First describe how to choose between the short-distance diffusive spread that is currently in the model, and long-distance events. If a long-distance event is needed, how could you estimate which patch to ignite? This will require a selection of both direction *and* distance [2]