Project 1

Rabbits & Grass Ecosystem

Written by

Prof. Erin N. Bodine

Prepared for

Math 314: Agent-Based Modeling

Rhodes College

Last Edited: January 10, 2018

In this lab we introduce the Rabbits ABM which is simulated in `Rabbits.nlogo`. If you have not done so, download this file from Moodle. In Lecture we will have ran many simulations of this model, and explored some of the behavior of the model using different parameter values. In the lab, we will take a look at what is under the hood (i.e., at the code) of the model.

Description of the Model:

> This agent-based model explores a simple ecosystem made up of rabbits and grass. The rabbits wander around randomly, and the grass grows randomly. When a rabbit bumps into a patch of grass, it eats the grass and gains energy. If the rabbit gains enough energy, it reproduces. Each rabbit loses energy every time it moves and reproduces. If a rabbits loses too much energy it dies.

Introduction to NetLogo

When coding a model into NetLogo, you typically have a SETUP procedure (which initializes the model), a GO procedure which runs the model, and set of other procedures which represent the "submodels" of the ABM.

In the remainder of the NetLogo skills section, we will the code of this model. You should note that in NetLogo comments are inserted by placing a `;` (semicolon) in front of the text or line that you want to use as a comment. Also, note that unlike MATLAB, NetLogo does not use line numbers in the code.

World Settings

There is one portion of the model that is not defined in the code, and that is the world settings. The "world" refers to the grid of patches represented in the model. To access the world settings, select the Interface tab (see Figure 1), then click on the "Settings..." button. This should open up a window that looks like Figure 2.

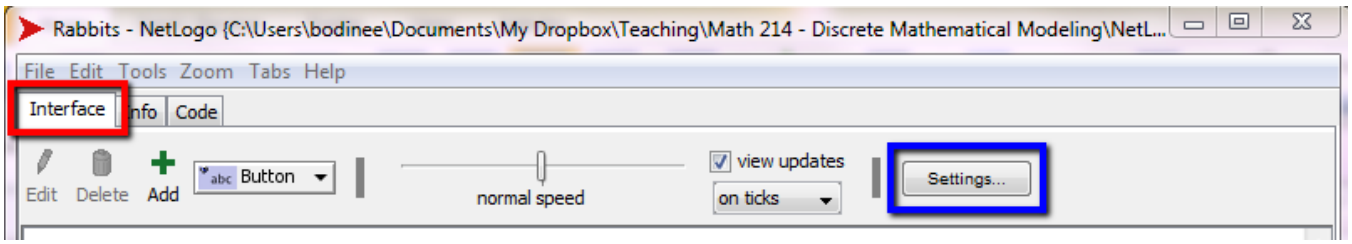


Figure 1: Header bar of the NetLogo Program when the Interface tab is selected. Boxed in red is the Interface tab. Boxed in blue is the button which opens the world settings window.

Within the world settings window, you can see (and change) how the patch grid is set up. The current settings are that the patch with coordinates $(0, 0)$ is located in the center, the world is 41 patches high and 41 patches wide, and there is both horizontal and vertical world wrapping. This means that if an agent moves past the right boundary of the world it will re-emerge on the left boundary (and vice-versa), and if the agent moves past top boundary it will re-emerge on the bottom boundary (and vice-versa). If the world does not wrap, then the agents are not allowed to move past the boundaries.

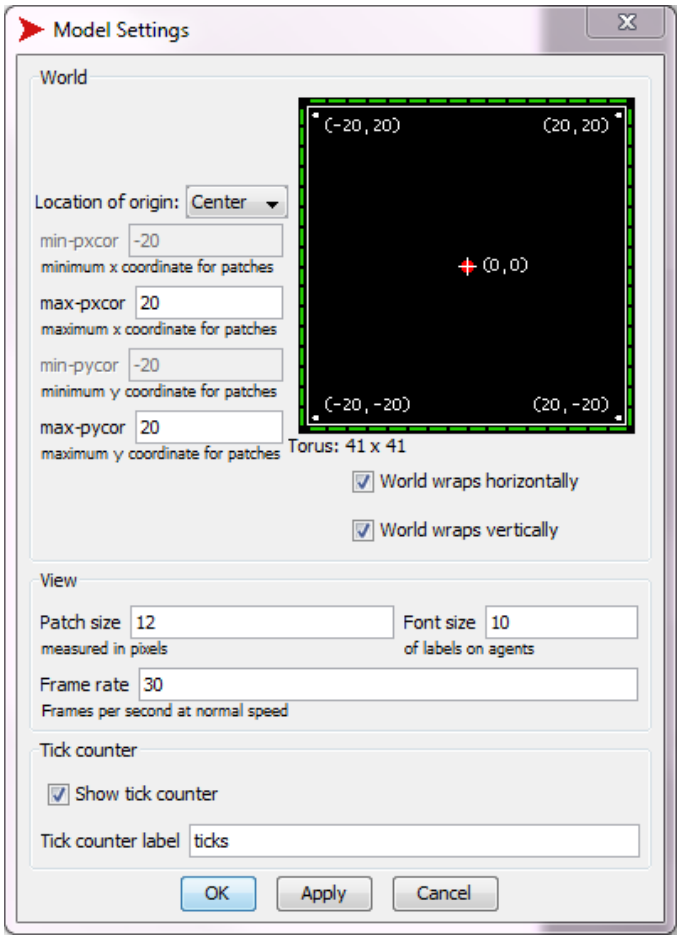


Figure 2: World settings window.

Creating Variables

The first part of any NetLogo code establishes the variables of agents and patches, and any global variables within the model. Note agent and patch variables are values that can be unique to each agent and patch. Thus, if all the agents have an age variable, one agent can have `age = 5`, while another agent can have `age = 39`. Global variables, on the other hand, are set to the same value for all entities (agents and patches) within the model.

Agent & Patch Variables: The agent and patch variables of the ABM are set up within the code. Select the code tab to view how these variables are created.

Snippet of Rabbits.nlogo

```
;-------------------------------------------------
;----- Create Agents -----------------------------
;-------------------------------------------------
breed [rabbits rabbit]
rabbits-own [ energy ]
```

In the snippet of code above, first notice that the first three lines are comment lines which tell us at which portion of the code we are looking. The first line of actual code is one the third line; it establishes what we will call our agents: we will call the collective group of agents `rabbits` and any single agent `rabbit`. If you do not define a name for the agents, then they will by default be called `turtles`.

In the second line, we define the variables of our agents called rabbits. In this case, we only create one variable called `energy`. However, for every agent, NetLogo reserves several built-in agent variables:

- `who` – a unique ID number for each agent.
- `shape` – defines the appearance of the agent in the world view in the interface tab. To see the available shapes, check `Tools > Turtle Shapes Editor`. The default is `default` and appears as an arrowhead.
- `color` – defines the color of the agent in the world view in the interface tab.
- `size` – defines the size of the agent in the world view in the interface tab and default set to the size of a single patch (which has a value of 1).
- `xcor` – defines the x–coordinate of the agent's current position, a real number representing the horizontal location on the patch grid. The x-coordinate of the red agent in Figure 3 is -1.5.
- `ycor` – defines the y–coordinate of the agent's current position (a real number representing the vertical location on the patch grid). The y-coordinate of the red agent in Figure 3 is 0.8.
- `heading` – defines the direction (in degrees) in which the agent is pointing. Example heading values are shown in Figure 4. For example, if the agent is heading north, then their heading value is 0 or 360, but if the agent is heading east, then their heading value is 90 or -270.
- `breed` – defines the type of agent. This becomes an important feature when a model has more than one type of agent.

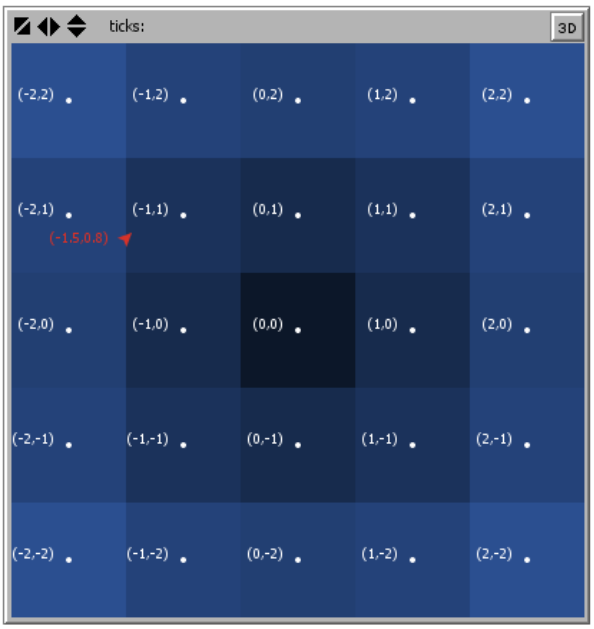


Figure 3: Coordinates for world grid where the origin is located at the center of the grid. The coordinates of each patch is the coordinate value at the center of that patch, e.g., the patch coordinates of the top left patch are $(-2, 2)$. The red turtle is located at the coordinates $(-1.5, 0.8)$.

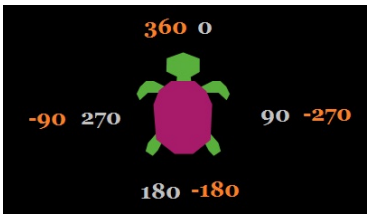


Figure 4: Diagram of heading values of an agent.

Because these variables are built-in to each agent, we do not need to create them when we create the agent variables.

You can also define variables for patches using the `patches-own` command, however we do no do that in this ABM. However, we will use the built-in patch variable `pcolor` which defines the color of a patch. We will see the use of this in the GROW-GRASS procedure.

Global Variables: The global variables of the ABM are set up on the interface. Select the interface tab to view see the global variables (see Figure 5).

The global variables are

- `initial-num-rabbits` – The number of rabbits (agents) at the beginning of the simulation. The value can be set to any integer from 0 and 500.
- `birth-threshold` – The minimum amount of energy required for a rabbit to reproduce. The value can be set to any integer from 0 to 20.
- `grass-energy` – The amount of energy contained in a patch of grass. This is the amount of energy a rabbit will gain if it eats a patch of grass. This value can be set to any multiple of 0.5 from 0 and 10.
- `grass-growth-rate` – The growth rate at which new patches of grass appear. For example, a growth rate value of 0.015 means that at each time step each patch has a 1.5% chance of growing a patch of grass. The `grass-growth-rate` can be set to any multiple of 0.001 from 0 to 0.05.

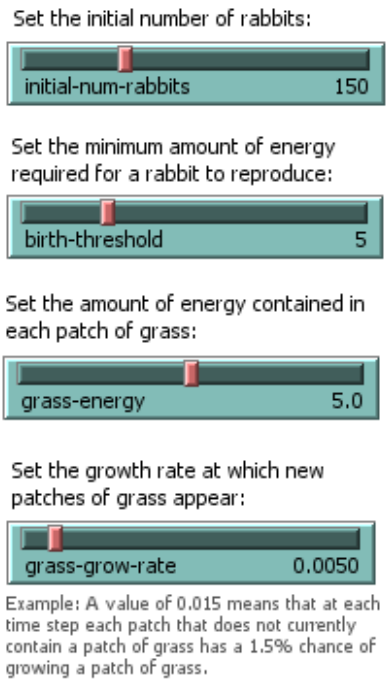


Figure 5: Global variables as they appear in the interface tab.

The SETUP Procedure

In ABMs it is typical to run a small portion of code which sets up the model before running the actual model. We will usually refer to this as the SETUP procedure.

Snippet of Rabbits.nlogo

```
;----------------------------------------------------
;----- Setup ----------------------------------------
;----------------------------------------------------
; Observer Procedure
to setup
  clear-all
  ask patches [grow-grass]

  set-default-shape rabbits "rabbit"
  create-rabbits initial-num-rabbits
  [ set color white
    setxy random-xcor random-ycor
    set energy random 10  ;start with a random amt. of energy
  ]

  reset-ticks
end
```

The first three lines are comment lines and indicate which portion of the code this is. The fourth line is also a comment line and tells "who" executes this line of code. Procedures can be called by the observed (you), by agents, or by patches. The SETUP procedure is always called by the observer.

The next line (and the last line) indicate the beginning and the end of the procedure. All procedures are framed in this fashion.

```
to procedure-name
  ...
  Commands of procedure
  ...
end
```

After the `to setup` we begin with the code that is executed when this procedure is run. In NetLogo, built-in functions are referred to as ***primitives***. First the `clear-all` primitive is executed which clears all variables (global variables, patch variables, and agent variables).

Next, we ask each patch in the world to execute the `grow-grass` procedure. You should note that the patches are asked in a random order.

Next, we set the default shape for our agents `rabbits` to be the `''rabbit''` shape, then we create n rabbits where $n =$`initial-num-rabbbits`. As we create the rabbits we can set values of each rabbit's variables. We set the color of each rabbit to white, and place each rabbit in a random location in the world (`setxy random-xcor random-ycor`), and set the `energy` variable to a random integer in the range $[0, 10)$ (note this means 10 cannot be selected, but zero and all positive integer below 10 can be randomly selected).

NetLogo has a built-in method for tracking the current time step. It is an internal counter called `ticks`. The command `tick` automatically increments the tick counter, and the command `reset-ticks` automatically set the counter back to 0. The last command in the SETUP procedure resets the ticks to 0.

The GO Procedure

The next procedure is the GO procedure. This is the main procedure within the model. Within the GO procedure, we call on all of the other procedures (submodels). The flow-diagram in Figure 6 shows tasks executed by the GO procedure, including the execution of submodels.

Snippet of Rabbits.nlogo

```
;------------------------------------------------
;----- Go ---------------------------------------
;------------------------------------------------
; Observer Procedure
to go
  if not any? rabbits [ stop ]

  ask patches [grow-grass]

  ask rabbits
```

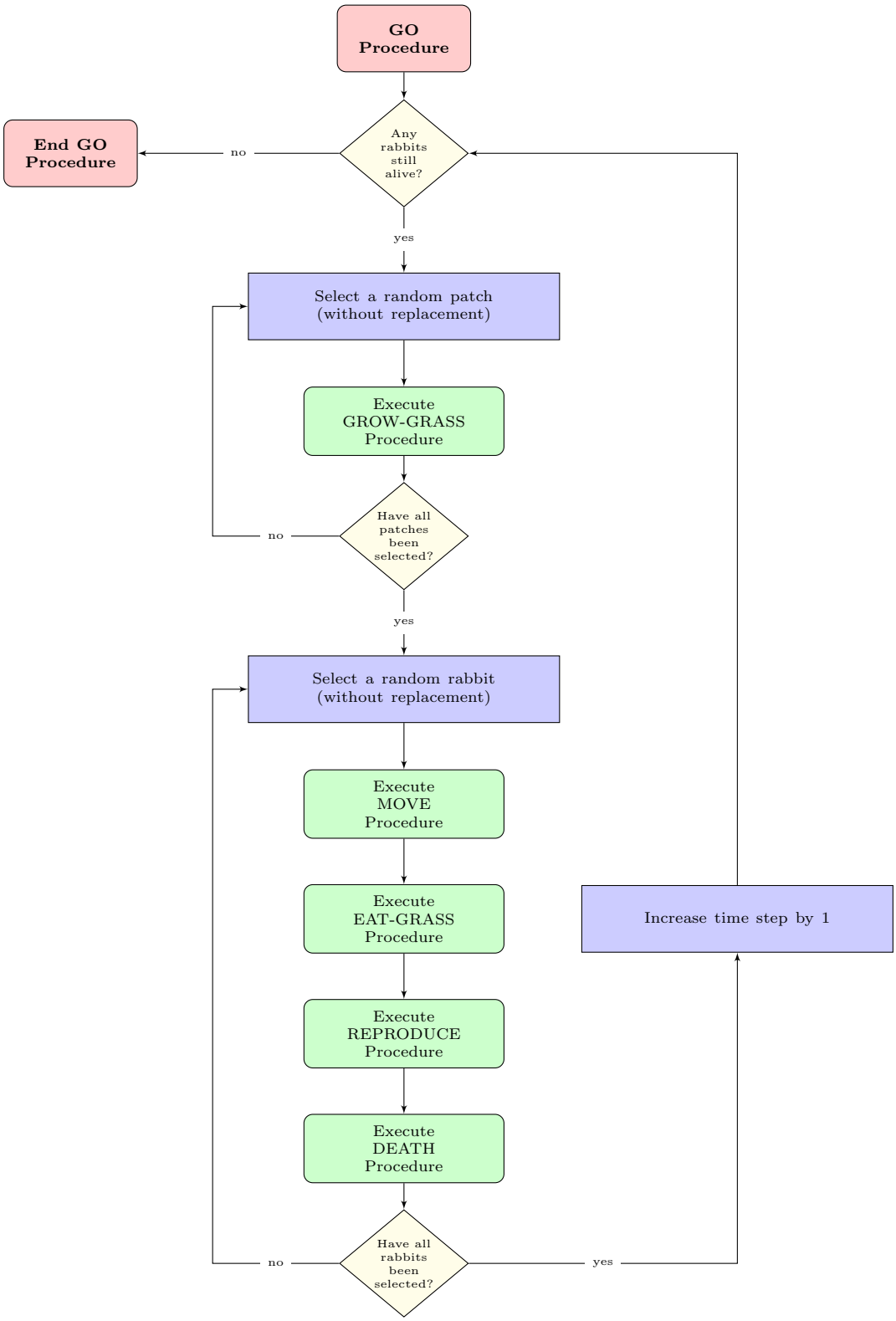


Figure 6: Flow diagram of GO procedure. A submodels are shown in green.

```
  [ move
    eat-grass
    reproduce
    death
  ]

  tick
end
```

First, note that the GO procedure is executed by the observed. Inside the GO procedure (after the `to go` command), we first determine if there are any rabbits left in the simulation and stop the simulation if there are not more rabbits:

```
if not any?  rabbits?  [stop]
```

Next, we ask each patch to execute the GROW-GRASS procedure. Note, this means that GROW-GRASS procedure must be a patch procedure.

Next, we ask each rabbit to execute the MOVE procedure, followed by the EAT-GRASS procedure, followed by the REPRODUCE procedure, followed by the DEATH procedure. Since each of these procedures are called by a rabbit, they must each be a rabbit (agent) procedure.

Lastly, we increase the `ticks` counter by 1 by executing the command `tick`.

The GROW-GRASS Procedure

The only procedure called/executed by the patches is the GROW-GRASS procedure.

Snippet of Rabbits.nlogo

```
;-------------------------------------------------
;----- Grow Grass --------------------------------
;-------------------------------------------------
; Patch Procedure
to grow-grass
  if (pcolor = black) and (random-float 1 < grass-grow-rate) [ set pcolor green ]
end
```

If you do not count the commenting, this procedure is only three lines of code, and two of those lines are used to indicate the beginning and ending of the procedure. In this procedure if the patch is colored black (`pcolor = black`), and if a real number randomly selected from the uniform distribution $\mathcal{U}[0, 1]$ is less than `grass-grow-rate` (`random-float 1 < grass-grow-rate`), then the color of the patch will be set to green. Note that when using `=`, `+`, `-`, `*`, `/`, `>`, `<`, `>=`, or `<=` in NetLogo you must have spaces on either side of the mathematical symbol. If not, NetLogo will return an error, or interpret your code incorrectly.

Since this *if-statement* has two conditions, both conditions must be satisfied in order for the patch color to be changed

to green. If one of the conditions is not met, i.e. it is false, then the command in the if-statement (`set pcolor green`) will not be executed. Table 1 show outcomes given various combinations of each condition being satisfied or not satisfied.

Table 1: The if-statement in the GROW-GRASS procedure has two conditions. This table shows the outcome of the if-statement given whether or not each condition is satisfied (i.e., true).

Condition 1 `pcolor = black`	**Condition 2** `random-float 1 < grass-grow-rate`	**Outcome** Command in if-statement executed
True	True	True ⇒ patch color is set to green
True	False	False ⇒ patch remains black
False	True	False ⇒ patch is already green and remains green
False	False	False ⇒ patch is already green and remains green

The MOVE Procedure

The MOVE procedure is one the procedures called by the rabbits (the agents). It changes the direction in which a rabbit is moving, and then moves them 1 unit (1 patch length) in that direction.

Snippet of Rabbits.nlogo

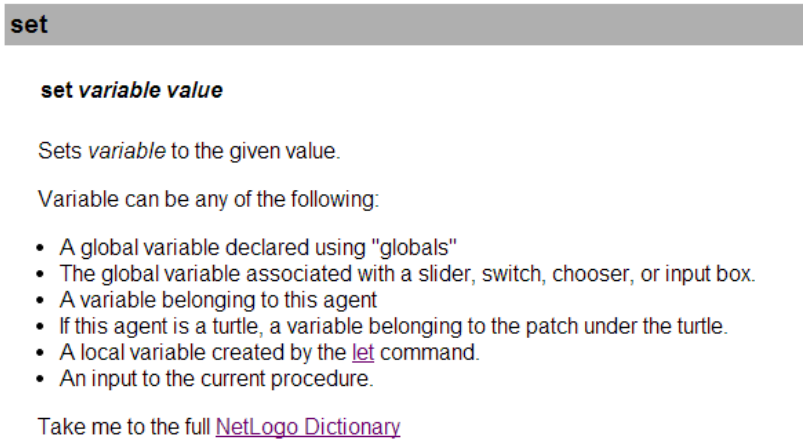
```
;------------------------------------------------
;----- Move -------------------------------------
;------------------------------------------------
; Rabbit Procedure
to move
  set heading heading + (random-float 100 - 50)
  fd 1
  set energy energy - 0.5  ;; moving takes some energy
end
```

The first line executed within the MOVE procedure changing the current heading of the rabbit. Note that the structure for the `set` command is

`set` *variable value*

This means first you have the command `set`, followed by a space, followed by the name of the variable you are setting, followed by the value you want to set that variable to. If at any point you forget how to use a certain command, place the cursor somewhere in the command and press F1. This will open the NetLogo dictionary to the entry on that particular command. For example, if you place the cursor somewhere in the word `set`, a window will open with the entry shown in Figure 7.

In the case of the first set command used in the MOVE procedure, we are setting the rabbit variable `heading` to the current value of `heading` plus a random amount. The command `random-float 100` randomly selects a real number from the uniform distribution $\mathcal{U}[0, 100)$. Note that the value 100 can never be selected but 99.99999999 can. Thus,



set

set *variable value*

Sets *variable* to the given value.

Variable can be any of the following:

- A global variable declared using "globals"
- The global variable associated with a slider, switch, chooser, or input box.
- A variable belonging to this agent
- If this agent is a turtle, a variable belonging to the patch under the turtle.
- A local variable created by the let command.
- An input to the current procedure.

Take me to the full NetLogo Dictionary

Figure 7: Example of entry for `set` in the NetLogo Dictionary.

the command `random-float 100 - 50` randomly selects a real number from the uniform distribution $\mathcal{U}[-50, 50)$. This means we are taking the current heading (which is measured in degrees) and adding a random number between -50 and 50. Suppose the randomly selected value is n. If $n < 0$, then the rabbit will turn to the left n degrees. If $n > 0$, then the rabbit will turn to the right n degrees.

The primitive `fd` is a NetLogo abbreviation for "forward". The command `fd 1` moves an agent forward (in the direction of the current heading) by 1 unit (the length of 1 patch).

The last command in the MOVE procedure sets the rabbit variable `energy` to 0.5 less than its current value. Thus, moving costs a rabbit 0.5 units of energy.

The EAT-GRASS Procedure

The EAT-GRASS procedure is one of the procedures called by the rabbits. This procedure enables a rabbit to eat a patch of grass and gain the energy contained in that grass.

Snippet of Rabbits.nlogo

```
;---------------------------------------------------
;----- Eat Grass -----------------------------------
;---------------------------------------------------
; Rabbit Procedure
to eat-grass
  ;; gain energy by eating grass
  if pcolor = green
  [ set pcolor black
    set energy energy + grass-energy
  ]
end
```

The first line executed in the EAT-GRASS procedure is an if-statement. The condition for the if-statement is

```
pcolor = green
```

Recall, that `pcolor` is a patch variable. However, in NetLogo, agents have access to the variables of the patch they currently occupy. So, this if-statement is asking if the patch color of the patch the rabbit currently occupies is green. If it is, then the if-statement will execute two commands. The first is

```
set pcolor black
```

which changes the color of the patch currently occupied by the rabbit to black. The second is

```
set energy energy + grass-energy
```

which changes the rabbit's `energy` variable to its current value plus the amount of energy contained in the grass (`grass-energy`).

The REPRODUCE Procedure

The REPRODUCE procedure is one of the procedures called by the rabbits, and it governs the generation of new rabbits.

Snippet of Rabbits.nlogo

```
;---------------------------------------------------
;----- Reproduce -----------------------------------
;---------------------------------------------------
; Rabbit Procedure
to reproduce     ;; rabbit procedure
  ;; give birth to a new rabbit, but it takes lots of energy
  if energy > birth-threshold
  [ set energy (energy / 2)
    hatch 1 [ fd 1 ]
  ]
end
```

Like the EAT-GRASS procedure, the first line executed in the REPRODUCE procedure is an if-statement. The condition for the if-statement is

```
energy > birth-threshold
```

So, the if-statement will execute of the the value of the rabbit's variable `energy` is greater than the value of the global parameter `birth-threshold`. If the condition is true, then two commands will be executed. First,

```
set energy (energy / 2)
```

which sets the rabbit's energy variable to half of its current value. Notice, in this command we use parentheses around `energy / 2`. Including these parentheses is optional, but it can often make the code easier to read. The second command in the if-statement executed is

```
hatch 1 [fd 1]
```

. You should place your cursor somewhere in the word hatch and press F1 to read the NetLogo dictionary entry on the primitive `hatch`. So, the rabbit who called the `REPRODUCE` procedure "hatches" one offspring who inherits all of its variables from the parent rabbit. Thus, if the parent rabbit had a heading of 287° and an energy level of 6.5, then the offspring rabbit will also have a heading of 287° and an energy level of 6.5. Once the offspring is hatched any commands in the square brackets after the hatch command are executed by the offspring. So the offspring moves forward one unit in the direction of their current heading.

The DEATH Procedure

The DEATH procedure is one of the procedures called by the rabbits. This procedure determines if a rabbit will die or not.

Snippet of Rabbits.nlogo

```
;-------------------------------------------------
;----- Death -------------------------------------
;-------------------------------------------------
; Rabbit Procedure
to death
  ;; die if you run out of energy
  if energy < 0 [ die ]
end
```

There is only one line of code executed by the DEATH procedure and it is an if-statement,

```
if energy < 0 [die]
```

This checks if the rabbit's `energy` variable is negative. If it is then the rabbit dies. Note the primitive `die` kills the agent executing the command. You can check the NetLogo dictionary for more details on the `die` primitive.

This takes us through all the elements of the code.

Plots in the Interface

There is one last element of the interface we should examine: the plot window. You can open the plot window for a plot by right clicking on the plot, and selecting Edit. At the top of the window you can enter the name (or title) of the plot, the x-axis label, and y-axis label. You can set the scale of these axes, or select "Auto scale?". In the "Plot setup commands" you should see the command

```
set-plot-y-range 0 initial-num-rabbits
```

This sets the initial range of the y-axis to be $[0, \texttt{initial-num-rabbits}]$. The "Plot pens" box is where you define what you would like plotted. This plot is generating two curves. One called `grass`, and one called `rabbits`. If you click on the color to the left of each pen name, a window will pop up and you can select the color in which you would like the curve to be plotted. The grass curve will plot the total number of patches that are green. The command for this is

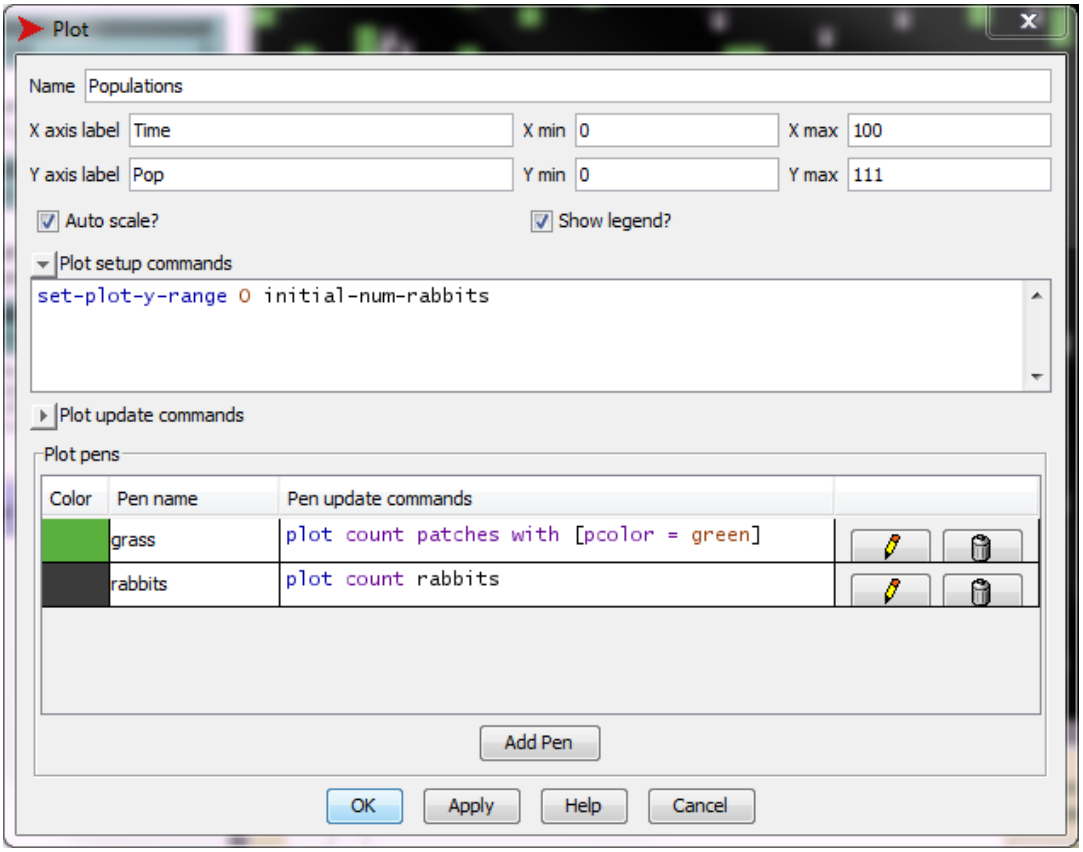


Figure 8: Plot window for the `Rabbits.nlogo` file.

```
plot count patches with [pcolor = green]
```

The rabbits curve will count the total number of rabbits. The command for this is

```
plot count rabbits
```

You should take a moment to look up the primitives `plot`, `count`, and `with` in the NetLogo dictionary.

Assignment

Now that we have gone through each component of the NetLogo code for the model, you will make some modifications to the code and to the plot.

1. Start by opening the `Rabbits.nlogo` file and saving it as `P01_YourLastName.nlogo`.
2. Create a patch variable called `grass-energy`. To do this, look up the `patches-own` primitive in the NetLogo dictionary. Note that the default value for any variable is 0.
3. After you have added the new patch variable, you can delete the global variable called `grass-energy` on the interface tab. To do this, right click on the `grass-energy` slider bar and select delete. You can delete the corresponding text box in the same fashion.

4. Modify the GROW-GRASS procedure so that when a patch is changed to green (i.e., it grows grass), the `grass-energy` variable of that patch is set to a random integer in the range $[1, 10]$. To do this, look up the `random` primitive in the NetLogo dictionary.

   To test that your have used the `random` primitive correctly, go to the interface tab and run the SETUP procedure. Click on the double-arrow button in the Command Center subwindow:  ; this will allow you to have a better view of the output of the command you are about to execute. Now, in the Command Center type the command

   ```
   ask patches with [pcolor = green] [ show grass-energy ]
   ```

   Did you get only integer values in the interval $[1, 10]$. Run SETUP again, and execute the command again, just to make sure. If there are any values outside the interval $[1, 10]$, you will need to edit your code initializing the `grass-energy` of green patches. Once you have your code correct, i.e., no values outside the interval $[1, 10]$, take a screenshot of the Command Center subwindow and save the file as `Lab08_YourLastName-Test.jpg`

5. Modify the EAT-GRASS procedure so that when a rabbit eats a patch of grass, the `grass-energy` variable of that patch is set to 0.

6. Open the plot window. Add a third curve to the plot. Use the pen name `grass-energy`. Plot the total amount of grass energy over all of the patches. To do this you will need to use the primitive `sum`. Look up this primitive in the NetLogo dictionary. Once you have made your changes in the plot window, click OK.

7. Modify the GO procedure so that if the value of `ticks` is greater than or equal to 1000, the simulation will stop. Note, this stop condition should be in addition to the stop condition if there are no more rabbits. Thus, if the value of `ticks` ≥ 1000 or if there are no more rabbits then the simulation should stop.

8. Set your global parameters such that you will start with 150 rabbits, the birth threshold is 7 units of energy, and the grass growth rate is 0.6%. Run the simulation for 1000 ticks or until all the rabbits die, whichever comes first. Take a screenshot of your simulation. Make sure the graph and global variables are visible in the screenshot. Save the image as `P01a_YourLastName.jpg`.

9. Set your global parameters such that you will start with 150 rabbits, the birth threshold is 7 units of energy, and the grass growth rate is 0.3%. Run the simulation for 1000 ticks or until all the rabbits die, whichever comes first. Take a screenshot of your simulation. Make sure the graph and global variables are visible in the screenshot. Save the image as `P01b_YourLastName.jpg`.

10. **Submit via Box:**

    - Your NetLogo file (`P01_YourLastName.nlogo`)
    - Screenshot of GROW-GRASS test (`P01_YourLastName-Test.jpg`
    - In a Word document (`P01_YourLastName.docx`) answer the question: What effect did lowering the grass growth rate have on the simulation? You may want to run the simulation multiple times for each parameter value to get an idea of what is happening. Include the simulation screenshots in the Word document (do not submit them individually to Box).