

13.1 NetLogo Tutorial 1a

*Introduction to Computational Science:
Modeling and Simulation for the Sciences, 2nd Edition*

Angela B. Shiflet and George W. Shiflet
Wofford College
© 2014 by Princeton University Press

Introduction

This first tutorial on *NetLogo*® gives an introduction to the system and prepares you to understand a *NetLogo* implementation of the model in Module 11.2, "Agents of Interaction: Steering a Dangerous Course," and to use the software to complete various projects in Chapters 11 and 14. The tutorial is in three parts (1a, 1b, 1c), and Tutorial 1a, which is in this document, develops a simulation of unconstrained growth.

One technique of modeling the movement of individuals and the spread of disease amongst the animals is a cellular automaton simulation. A related alternative is a grid-based, **agent-based (individual-based) simulation**.


For a *cellular automaton simulation*, the state of a grid cell might indicate the number of cattle at that location as well as attributes, such as weight(s), associated with the animal(s). Transition rules that specify the relationship of a cell with its neighbors determine the state of the cell at the next time step. For each time step, a cellular automaton simulation sweeps through every cell of the grid, updating its state.

With an *agent-based simulation*, each animal is modeled as an autonomous, decision-making **agent** that has a **state**, which is represented by a set of **state variables**, or attribute values, and **behaviors**, which control its actions. A **method** or **procedure**, which is associated with a class, or breed or group, of agents, is a function that captures some or all of an agent's behavior. A simulation frequently includes several **global simulation variables**, which all agents can access. Agents often operate in an **environment** that arranges cells in a rectangular **grid**. (Individually based models exist that are not grid-based.) The environment, its neighboring agents, and the states and behavior of an agent determine the agent's new state. For each time step, instead of iterating through each grid cell, an agent-based simulation proceeds through each agent, revising its state.

Creation of a Model of Unconstrained Growth

In this section, we will create a model of unconstrained growth of bacteria on a petri dish. We start with one bacterium, allow it to move to a neighboring patch with a 15% probability, and have it divide with a 10% chance.

Start NetLogo. If you have been working with another model in NetLogo, from the *File* menu, select *New* or use the indicated shortcut to begin a new model. In a separate document, type the answers to all quick review questions.

First, we want to put a *setup* button in our model and indicate what to do when the user clicks this button. Because the drop-down menu towards the top, left shows, **Button** , clicking **Add** (Add) to the menu's left will start the process of adding a button.

Quick Review Question 1 Click *Add* and then click in the white space towards the top of the window. What two things happen?

In the pop-up menu called *Button*, type “setup” in the *Commands* textbox and click *OK*. As a result, “setup” should now appear in the new button. However, the name is in red, because we do not have an associated action when the user presses the button. To start remedying the situation, click the **Code** tab at the top, right of the NetLogo window.

Quick Review Question 2

- What appears when you press the **Procedures** button?
To start a procedure associated with the *setup* button, type the following:
- Where does the cursor appear when you press <RETURN> or <ENTER>?
- Press <RETURN> or <ENTER> again so that we have a blank line, and type the following:

```
to setup
```

```
end
```

Where does "end" align? With the checkbox **Indent automatically** checked at the top of the screen, the NetLogo code editor helps us with the proper indentation.


In the blank line after "to setup," type **clear-all** or its abbreviation **ca**. Execution of this command will erase whatever was left from the last time we ran the model. After pressing <RETURN> or <ENTER>, type the following **set-default-shape** command so that our bacteria, which turtles represent in this application, will have the shape of a **circle**:

```
set-default-shape turtles "circle"
```

With the shape established, type the following **create-turtles** (or **crt**) to create one bacterium with **color red** in the middle of the window, which has xy-coordinates (0, 0):

```
crt 1 [ set color red ]
```

Save the file by selecting **Save** from the *File* menu. Make sure you save the file to your disk with a meaningful name, such as *Tutorial1a.nlogo*. After subsequent work, save by selecting *Save* or using the indicated shortcut. **SAVE OFTEN**, particularly before you print or run a simulation.

Quick Review  **Question 3** Change "crt" to an incorrect command, such as "xyz." Click

Check () to check the syntax of your commands. What appears? Restore the correct command, *crt*, and click *Check* again

It is quite a challenge to remember what various procedures do. Consequently, we should write a comment before each procedure and at important points within a procedure. The rest of a line after a **semicolon (;)** is a **comment** that NetLogo does not attempt to execute. On a line before "to setup," type a semicolon and a description of the procedure, such as "put red turtle at location (0,0)." Your complete code should now appear as follows:

```
; put red turtle in location (0, 0)
to setup
  clear-all
  set-default-shape turtles "circle"
  crt 1 [ set color red ]
end
```

Quick Review Question 4 Add a comment at the top of the code with your name and another comment with a model description, such as "Unconstrained growth simulation." Give the second comment on your answer sheet.

Quick Review Question 5 Click the *Interface* tab at the top of the window.

- What color is "setup"? This color indicates that we now have a procedure associated with the button.
- Click the button and describe what happens.

Quick Review Question 6 With the setup accomplished, we are ready to have things happen. How do you add a button called "go"? Do this task.

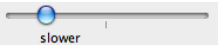
On many models the *setup* and *go* buttons are next to each other. To move a button, right-click (CTRL-click on Macintosh), choose *Select* in the dropdown menu, move the button, and then click elsewhere. Similarly, to edit or delete the button, we make the corresponding selection in the dropdown menu.

We start by asking our turtle (actually, each turtle) to move to a neighboring patch. In this case, we consider neighbors to be those eight surrounding patches in its Moore's neighborhood (see Figure 10.2.3). The command is as follows:

```
ask turtles [ move-to one-of neighbors ]
```

Under the *Code* tab, add a *go* procedure associated with the *go* button that asks the turtle to move to a neighbor. Check that the procedure is correct. Save.

Quick Review Question 7 In the interface, click *go*. How many times does the red circle, which represents our turtle (bacterium), move?

We want the circle to keep moving. Thus, edit the *go* button, and in the popup menu, check the **Forever** checkbox. Now, after clicking *go*, the circle moves continuously. Using the slider bar () at the top of the window, adjust the speed of the simulation to be slow enough to observe the action. Notice that eventually, the red dot will travel off one side of the field of patches and reappear at the opposite side. Thus, we say that the simulation uses **periodic boundary conditions** (see Module 10.2, "Diffusion"). Stop the simulation by pressing *go*.

It seems unlikely that a bacterium would move at every time step. Thus, suppose we want this agent to move with a 15% chance. As in Figure 9.3.4, if a random floating point number between 0 and 1 is less than 0.15, the bacterium should move to a neighboring patch. The form of an **if statement** in NetLogo is as follows:

```
if condition [ body ]
```

If the condition is true, then the computer executes the body, which appears between brackets.

To investigate how to get a random floating point number, from the **Help** menu, select **NetLogo Dictionary**, and click the Alphabetical letter "R" for "random." As the dictionary states, **random** is used to obtain random integers, or whole numbers. However, the description directs us to the command we need, **random-float**: "If *number* is positive, reports a random floating point number greater than or equal to 0 but strictly less than *number*."

Quick Review Question 8 On the bottom of the interface, type in the **observer>** textbox the command to return a random floating point number greater than or equal to 0 and less than 1.

- a. Give the command that you typed. Press the up arrow and then <RETURN> to execute the command again. Do this process several times to notice two things: NetLogo adds **show** in front of your command and displays a different number each time.
- b. Press the up arrow again. At the end of the line, type " < 0.15" with a blank on each side of "<" and press return. (NetLogo requires blanks to surround operators, such as "<" or "+".) Using the up arrow, re-execute the command several times. Each time, we are asking the computer to find a random number and tell us if it is less than 0.15 or not. What are the two possible values to which the condition evaluates?
- c. Write the NetLogo statement in *go* procedure for the following pseudocode, or English description, which moves the turtle with a 15% chance:

```
ask turtles
  if a random floating point number between 0 and 1 is less than 0.15
    move to one of the neighbors
```

Remember to use two sets of brackets, one set to enclose what we are asking the turtles to do and one set for the body of the *if* statement, which has a turtle moving to a neighbor. Check the code, and then, from the interface, click

setup and then *go*. The turtle (bacterium) should now move only about 15% of the time.

Ten percent of the time, we wish a bacterium to "divide." So that we can easily adjust this growth rate and the probability of movement from the previous quick review question, at the top of the file declare variables *growth-rate* and *chance-move* to be global variables, or **globals**, such as follows:

```
globals [
  growth-rate
  chance-move
]
```

Then, in the *setup* procedure, we assign the value 0.1 to *growth-rate* with a **set** command as follows:

```
set growth-rate 0.1
```

Add both the *globals* and *set* commands to the appropriate places in the code.

Quick Review Question 9

- Give the command in *setup* to assign 0.15 to *chance-move*, and add the command to your code.
- We can ask a turtle to "give birth" with the **hatch** command, where the number following *hatch* is the number of children. Write the statement to do the following and add the command to *go*:

```
ask your turtles
  if a random floating point number is less than growth-rate
    hatch one child
```

Remember to include a set of brackets around the task you ask the turtles to do and a set of brackets around what is to be done if the condition of the *if* statement is true. Click *Check* to check your work. If errors occur, use the dictionary under the *Help* menu to look up commands.

- After the syntax is correct and you have saved the program, in the interface, adjust the speed of the simulation to be slower, and click *setup* and *go*. Describe what happens.

With each turtle hatching another turtle about 10% of the time, the number of turtles grows exponentially. Press *setup* and in the *observer>* textbox type the command to display the **count** of turtles:

```
show count turtles
```

With a slow simulation speed, start and stop the simulation several times, each time showing the count of turtles. Turtles (circles) can be on top of each other.

A much better way to display the information is to plot the count versus time. First, we need to establish time. At the end of *setup*, start the clock at zero ticks with the command ***reset-ticks***. Then, at the end of *go*, advance the clock with the command ***tick***. Add these commands to your code.

To add a graph to your system, first click the *Add* button in the interface. Then, change the dropdown menu to its right, which currently displays *Button*, to show ***Plot***, and click in whitespace. In the resulting popup menu, name the graph, such as "Number of Bacteria vs. Time;" type a label for the x-axis, such as "Time;" and a label for the y-axis, such as "Bacteria."

Quick Review Question 10 Give the default command that appears in this menu for graphing the number of turtles.

Click *OK* on this menu. If necessary to make more room, right-click on the simulation window, which has the red circle on the black patches, choose *Select*, and drag the window to the desired location. Similarly, we can select the graph window, and drag the entire window to a new place or drag on one of the black handles to expand its size. Save, click *setup* and *go*, and observe the exponential growth of turtles.

To regulate automatically how long the simulation runs, we can also use *ticks*. Suppose we want to stop the simulation if the number of ticks is 100 or more. The symbol for the "**greater than or equal to**" operator is ***>=***, and the command to terminate the simulation is ***stop***.

Quick Review Question 11 Give the command to do the following:

if the number of ticks is greater than or equal to 100
stop the simulation

Be sure to use brackets appropriately and to have blanks surrounding "***>=***." If you have difficulties, consult the NetLogo dictionary. Add the command to the end of the *go* procedure; check the code; setup; and run the simulation.