NetLogo Tutorial Series: Mystery Pattern

Nicholas Bennett
Grass Roots Consulting
nickbenn@g-r-c.com

July 2010

Copyright

Acknowledgments

Development of this curricular material and the accompanying NetLogo models was funded in part by:

- Santa Fe Institute Summer Internship/Mentorship (SIM) program for high school students;
- New Mexico Supercomputing Challenge;
- Project GUTS.

Participants in the above programs have also provided invaluable feedback on earlier versions of this material.

Introduction

Sometimes we find that simple behaviors, when repeated by many agents – or even by a small number of agents, repeating those behaviors many times – can produce rich, unexpected patterns.

In this activity, we'll be building a NetLogo model that has agents with a very simple behavior. In each step, a turtle will perform the following actions:

1. Select one of three target points (the vertices of a triangle) at random.
2. Move half the distance toward the selected target point – and stop.
3. Mark its new location with a white dot (on a black background).

To begin, each turtle will be located on one of the three target points. (Obviously, if a turtle selects the same vertex as its first target, it won't move in that iteration.)

What pattern (if any) will result from the turtles following their programmed behaviors? How many repetitions of the above steps will it take before any such pattern emerges?

We could try to answer the questions above by doing the exercise on paper, but it would probably become very tedious, very quickly – possible long before the "mystery pattern" is clear.

Task 1: Getting Started

(Before following the steps below, consider reading "The NetLogo Coordinate System". Even if you've read it already, it might be useful to review it.)

Start NetLogo & Configure World

1. From the Macintosh **Applications** folder, or from the Windows **Start/All Program/NetLogo** menu, launch the **NetLogo v4.1.1**[1] application (not **NetLogo 3D**).
2. Because we want the marks made by the agents to be placed very precisely, and because NetLogo colors and entire patch at once, we want to set up the NetLogo world with a large number of very small patches. To do this, click the **Settings…** button at the top of the NetLogo **Interface** window, and make these changes:

    a) Leave **Location of origin** set to **center**.

    b) Set the **`max-pxcor`** value to 288.

    c) Set the **`max-pycor`** value to 251.

    d) Uncheck the **World wraps horizontally** checkbox.

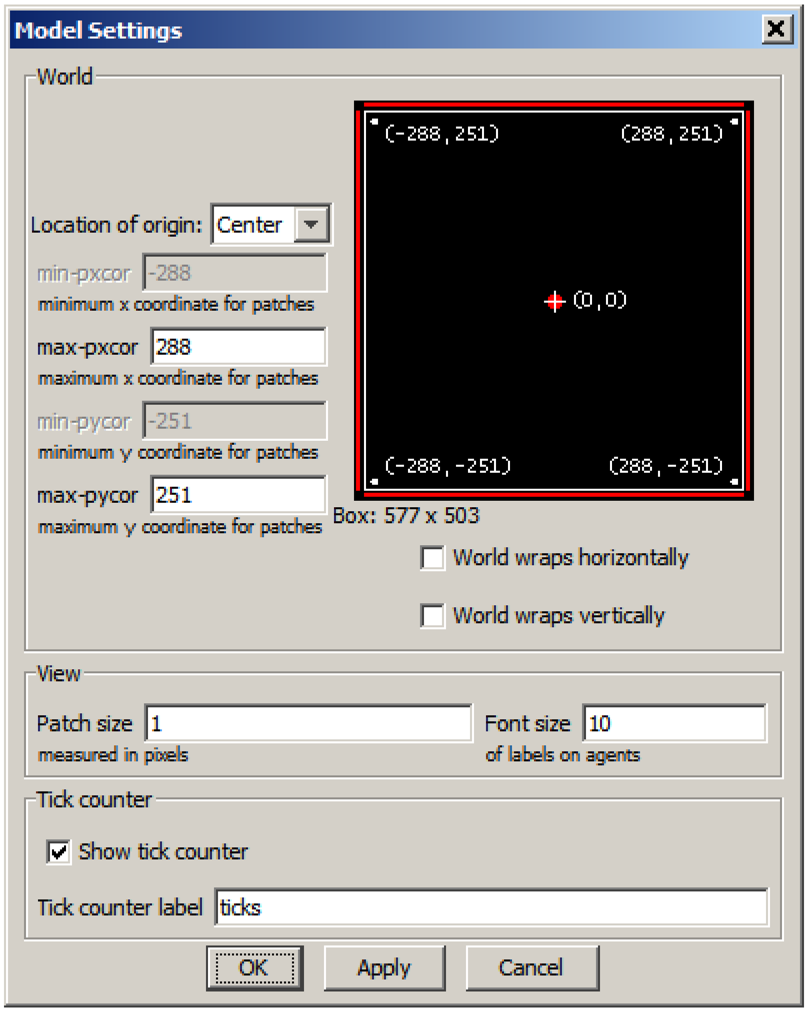
    e) Uncheck the **World wraps vertically** checkbox.

    f) Set the **Patch size** value to 1.0.

    g) Uncheck the **Show tick counter** checkbox.

1 This tutorial was originally written for NetLogo v4.0.4, then updated for and tested with NetLogo v4.1 and v4.1.1. As written, it should work with any 4.x version, though there might be some slight differences between the screen captures and the actual displays.

The **Model Settings** window should look like this:


Model Settings

World

Location of origin: Center

min-pxcor -288
minimum x coordinate for patches

max-pxcor 288
maximum x coordinate for patches

min-pycor -251
minimum y coordinate for patches

max-pycor 251
maximum y coordinate for patches

Box: 577 x 503

☐ World wraps horizontally

☐ World wraps vertically

View

Patch size 1
measured in pixels

Font size 10
of labels on agents

Tick counter

☑ Show tick counter

Tick counter label ticks

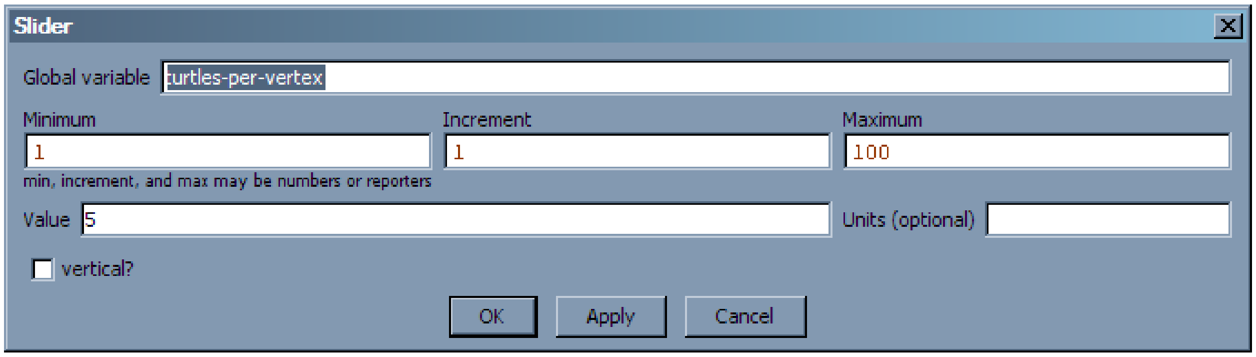
3. Click the **OK** button; we now have a world that is 503 patches tall and 577 patches wide, with each patch being 1 pixel X 1 pixel in size.

Controlling the Number of Turtles

One experiment we'll do is to see if the results vary with the number of turtles. Since all the turtles will begin on one of the three vertices of a triangle, we'll use a slider to control the number of turtles per vertex.

1. Select the **Slider** tool from the pull-down menu in the toolbar at the top of the screen.
2. Click somewhere in the white space to the left of the world; this will display the **Slider** configuration window.
3. Make the following changes:
   a) Set the **Global variable** value to **`turtles-per-vertex`**.
   b) Set the **Minimum** to 1.
   c) Set the **Increment** to 1.
   d) Set the **Maximum** to 100.

   The Slider configuration window should now look something like this:



Global variable: turtles-per-vertex
Minimum: 1
Increment: 1
Maximum: 100
min, increment, and max may be numbers or reporters
Value: 5
Units (optional):
vertical? ☐

4. Click the **OK** button.

Now we need to write the code that will use the value from this slide to create the specified number of turtles – and to make them follow the behavioral rules described previously.

Task 2: Writing the Model

Setting up the Target Points (the Vertices of the Triangle)

Imagine drawing a triangle on the NetLogo world, with its base stretching all the way across the bottom (with a little space left over, so we can see things more easily), and with the top vertex centered almost at the top of the world. As it turns out, the three vertices of that triangle are pretty easy to describe in NetLogo: if we leave a margin ten patches wide on all four sides of the world, then coordinates of the top, lower-left, and lower-right vertices are, respectively,**`(0, max-pycor – 10)`**, **`(min-pxcor + 10, min-pycor + 10)`**, and **`(max-pxcor - 10, min-pycor + 10)`**. Of course, we just set the dimensions of the screen ourselves (in fact, we set them to values chosen to make the resulting triangle equilateral), so we could just use the numbers we typed into the **Model Settings** window – but to be on the safe side, let's assume we don't know the specific dimensions.

For the next few minutes, we'll be writing NetLogo code. So, do the following:

1. Click on the **Procedures** tab, near the top of the NetLogo window.

2. Type the following at the top of the code editing area, to tell NetLogo that we'll be using a variable called **`corners`** to keep track of the three vertices of the triangle:

```
globals [
  corners
]
```

3. Below what you just typed (I recommend giving yourself a blank line or two first), type the following **`setup`** procedure (remember, spaces, spelling, and square brackets are *very* important; parentheses are less critical, except where noted):

```
to setup
  clear-all
  set corners (patch-set
    patch 0 (max-pycor - 10)
    patch (max-pxcor - 10) (min-pycor + 10)
    patch (min-pxcor + 10) (min-pycor + 10)
  )
  ask corners [
    setup-corner
  ]
end

to setup-corner

end
```

4. After typing the code, use the **Check** button at the top of the window to verify that you typed the code without spelling or spacing errors. If there are errors, review your code to make sure that you haven't accidentally substituted a space for a dash (or vice versa), that your spelling matches that shown above.

5. Let's review what the **`setup`** procedure is doing:

   a) We begin by using **`clear-all`** to clear the world, any turtles, and any variable values.

   b) Then, we tell NetLogo that the global variable **`corners`** will hold the set of patches located on the three vertices of our triangle.

   c) Finally, we ask each of those patches that we put into the **`corners`** variable to execute the **`setup-corner`** procedure. What does **`setup-corner`** do? Nothing yet – but it will soon!

6. Now would be a good time to save your program. From the **File** menu, select **Save**, and navigate to a folder where you can save files – the Desktop folder, or the folder assigned to you on a network file server, for example. You can give your program any name you like, as long as you use an extension of ".nlogo".

7. Modify the **`setup-corner`** procedure, as follows. Note that there's no need to retype the first and last lines of the procedure, since you already typed them. In this tutorial, code you've already typed will be shown in a gray typeface, while the lines you need to add are shown in a bold blue typeface. Where previously typed code and new code is shown together, the previous code is further distinguished by italicized type.

```
to setup-corner
  set pcolor white
  sprout turtles-per-vertex [
    set color blue
    set size 2.0
  ]
end
```

8. Again, use the **Check** button to check your typing. There might be logical errors in the code that you don't catch until you try to run your program, but using **Check** will help you catch typing problems early.

9. Now, let's review **`setup-corner`**:

    a) The patch (a vertex of the triangle) sets its color to white.

    b) The patch then calls the **`sprout`** command (which you might not have seen before) to create turtles. Here, the patch is creating as many turtles as are specified with the **`turtles-per-vertex`** slider that we created at the start.

    c) Finally, the patch asks each of the newly created turtles to set its color to blue, and its size to twice the normal size. (Normally, turtles are as large as the patches; since the patches in this model are so small, the turtles will be small as well. Setting the size to 2.0 will make them a bit easier to see.)

10. Save your program again, by using the **File/Save** menu option, or by typing Ctrl-S.

11. Click on the **Interface** tab to switch to the window showing the NetLogo world.

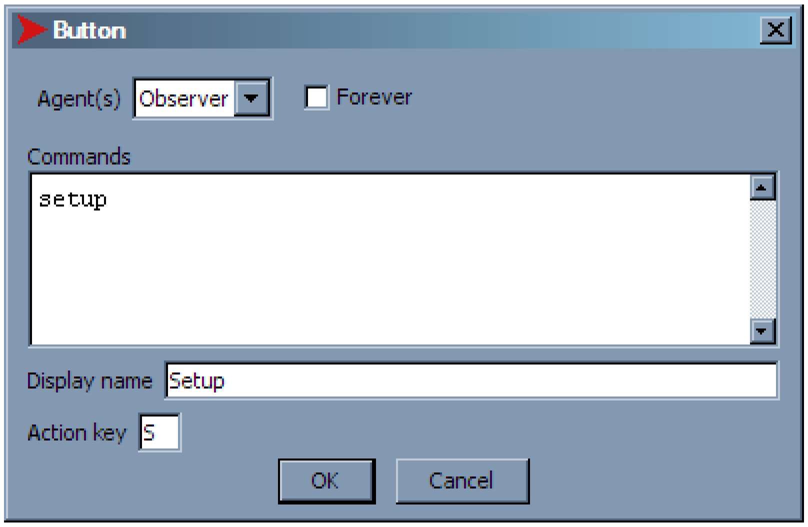
12. Select **Button** from the pull-down menu near the top of the window, next to the **Add** button.

13. Click somewhere in the white space to the left or right of the NetLogo world; the **Button** settings dialog will appear.

14. This button will be used to run the **`setup`** procedure you just wrote; to set it up correctly, set the following configuration values:

    a) Type the word **`setup`** into the **Commands** text box.

    b) If you want, you can give this button a different display name (by typing something in the **Display name** field), and/or a shortcut key, by typing a letter in the **Action key** field (as I have done in the example).



Agent(s) Observer
Forever [ ]
Commands
```
setup
```
Display name Setup
Action key S

15. Click the **OK** button.

16. Click the **Setup** button you just created. If your program is working correctly, you should see three blue dots, forming a triangle on the NetLogo world. Why are they blue? Didn't we ask those three patches to set their colors to white? Yes, we did. However, each of those three patches also created turtles, which are all standing – stacked up on top of each other, if there is more than one turtle per vertex – on those patches; each of those turtles is blue. When the turtles start moving, they will uncover the white patches.

17. If you got an error message at any point (e.g. when you clicked the **Check** button in the procedures window, or when you clicked your **Setup** button), read the message displayed, and look at the highlighted code. Did you leave necessary spaces out? Did you misspell a NetLogo command? Did you spell one of your own procedure or variable names inconsistently? If you can find the error, correct it and try again; if not, ask for help from the instructor.

Making the Turtles Move

As described earlier, the turtles in this model will follow a very simple set of rules. At each step, a turtle will pick one of the three vertices at random, and move towards the selected vertex, stopping after moving half of the original distance between the turtle and the vertex; then, the turtle will change the color of the patch it is on to white; then it will repeat the same process again (and again, and again, *ad infinitum* – at least until we tell it to stop).

In our **`setup`** procedure, we put the patches that will be the vertices of our triangle in the **`corners`** variable; they're still in that variable, even after **`setup`** is complete, so the turtles can use that variable when they need to select one of the vertices as a target point.

NetLogo has built-in statements that allow a turtle to change its direction, so that it is headed towards a patch or another turtle. It also has statements for calculating the distance from one agent (a turtle or patch) and another. We'll be using both of these facilities in writing the procedure for our turtle behavior.

1. Click on the **Procedures** tab, so that you can resume typing your NetLogo program.

2. Below the code you've already written, type the following:

```
to draw
  let target (one-of corners)
  face target
  forward (0.5 * distance target)
  set pcolor white
end
```

3. Use the **Check** button to make sure you don't have spelling or space errors.

4. Let's review the **`draw`** procedure:

   a) First, the turtle uses **`one-of`** to select (at random) one of the set of patches stored in the variable **`corners`**. A reference to the selected patch is stored in the local variable **`target`**.

   b) Next, the turtle changes direction, turning towards the selected vertex. There are a few ways to do this; in this case, we use the procedure **`face`**, which turns the turtle to face some other turtle or patch.

   c) Then, the turtle moves **`forward`** by half the distance to **`target`**.

   d) When the turtle comes to rest, it calls **`set pcolor`** to change the color of the patch where it's standing to white.

5. Review the contents of your **Procedures** window; it should look something like this:

```
globals [
  corners
]

to setup
  clear-all
  set corners (patch-set
    patch 0 (max-pycor - 10)
    patch (max-pxcor - 10) (min-pycor + 10)
    patch (min-pxcor + 10) (min-pycor + 10)
  )
  ask corners [
    setup-corner
  ]
end

to setup-corner
  set pcolor white
  sprout turtles-per-vertex [
    set color blue
    set size 2.0
  ]
end

to draw
  let target (one-of corners)
  face target
  forward (0.5 * distance target)
  set pcolor white
end
```

6. Save your program.

7. Click the **Interface** tab, to return to the window showing the buttons, sliders, and NetLogo world.

8. Select **Button** from the pull-down menu near the top of the window, next to the **Add** button.

9. Click in the white space to the left or right of the NetLogo world; the **Button** settings dialog will appear.

10. This new button will be used to set the turtles into motion, running the **`draw`** procedure you wrote. So we'll set the configuration as follows:

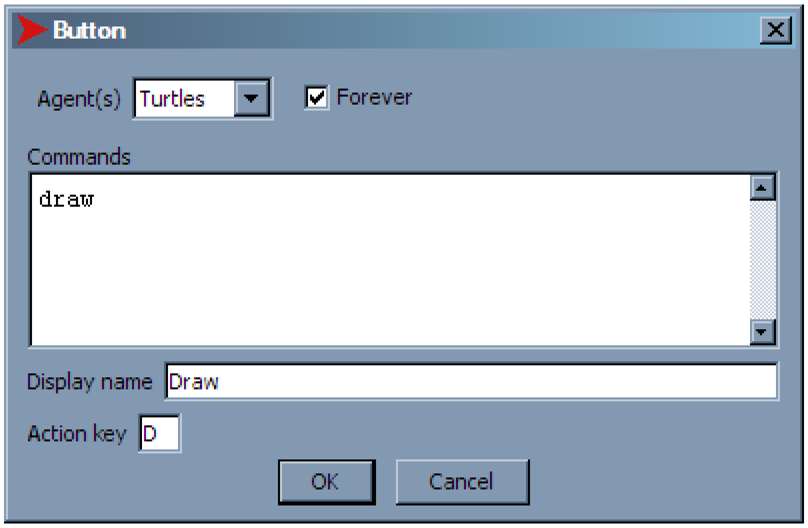
    a) Select "Turtles" from the **Agent(s)** pull-down menu.

    b) Put a check mark in the **Forever** checkbox.

    c) In the **Commands** text box, type **`draw`**.

    d) If desired, type a **Display name** and **Action key.**

    The **Button** settings dialog should like something like this (with possible differences in the optional **Display name** and **Action key** entries):



Agent(s) Turtles ☑ Forever

Commands

```
draw
```

Display name Draw

Action key D

11. Click the **OK** button.

12. Save your program.

Task 3: Running the Program

The Moment of Truth

If you've managed to catch your typing errors as you go, your program should be ready to run. Have you formulated a hypothesis about the type of pattern – if any – that will result, when the turtles start moving around and turning the patches they land on white? What do you think will happen?

Will it make a difference if you start with one turtle per vertex, vs. 100 turtles per vertex? Do the turtles interact directly in any way? Do they interact indirectly – e.g. by contending for space or other constrained resources?

Let's try it:

1. Set the **`turtles-per-vertex`** slider to a value of 1.
2. Click the **Setup** button.
3. Click the **Draw** button. If you get any errors, try and fix them – by yourself, or with the instructor's help.

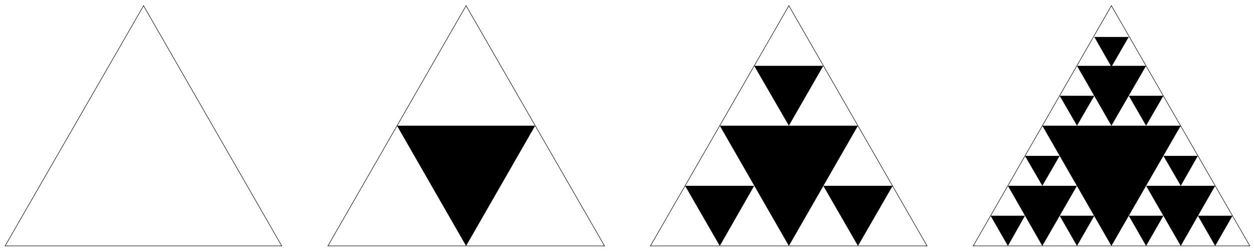
   Assuming your program is running correctly, what pattern do you notice?
4. Change the value of **`turtles-per-vertex`** to a higher value, and repeat steps 2 and 3. Has anything changed?
5. Repeat step 2, then slow your turtle movement down, using the slider at the top of the NetLogo world. Now repeat step 3, and watch how the turtles move. How would you describe this movement, in general terms?

What's the Mystery Pattern?

The pattern drawn by the turtles is called the Sierpinski triangle, or the Sierpinski gasket. The best-known recipe for creating this figure is as follows:

1. Start with a triangle.
2. From the original triangle, cut out the triangle formed by connecting the midpoints of all three sides of the original triangle.
3. Continue the process with the smaller triangles created, treating each one as if it were the original triangle.

The first few iterations of this process are illustrated below:



Actually, there are several methods for producing the Sierpinski gasket; the approach we used is a specific version of an algorithm called the *chaos game* [1]. In contrast with the construction method described immediately above, the rules followed in the approach we used give very few clues as to the final result – unless, of course, we already have some experience with the chaos game.

Task 4: Modifying the Model

Refining Turtle Movement and Display

1. There are a number of ways to change a turtle's location. So far, we've used the **`forward`** procedure, which moves the turtle forward in the direction it's already facing. Another approach is to use the **`jump`** procedure, which works in exactly the same manner as **`forward`**, except that NetLogo doesn't try to animate the travel of the turtle along its path.

   Modify your **`draw`** procedure, to use **`jump`** instead of **`forward`**. What's the result?

2. If the model uses **`jump`**, it might not be that important for the turtles to appear at all. In NetLogo, each turtle has a **`hidden?`** variable, which we can set to **`true`** or **`false`**, to to control whether a turtle is visible (alternatively, we can use the **`hide-turtle`** and **`show-turtle`** procedures).

   Modify your **`setup-corner`** procedure, to make all turtles invisible when they're created.

Questions for Reflection on Further Modifications

1. Your model uses an equilateral triangle (the dimensions we set for the NetLogo world at the start of the activity were chosen specifically to give us such a triangle). What do you think would happen if we used a triangle with different proportions? Is there any easy way to test your hypothesis?

2. The chaos game generalizes the problem, using the vertices of a regular n-gon, and moving a fraction r of the distance to the randomly selected vertex. The specific case we've been working with has $n = 3$, $r = 0.5$. How would you modify the model to support any n value in the set $\{3, 4, \ldots, 12\}$, and any r value in the set $\{0.01, 0.02, \ldots, 0.99\}$?

References

[1] Weisstein, Eric W. “Chaos Game”, *MathWorld*, August 2010. [Online]. Available: http://mathworld.wolfram.com/ChaosGame.html. [Accessed: August 5, 2010].