

NetLogo “Mystery Pattern”

Introduction

Sometimes, we may observe that very simple behaviors, when repeated by many agents – or even by a smaller number of agents, acting over a long period of time – can produce very rich, interesting patterns.

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

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

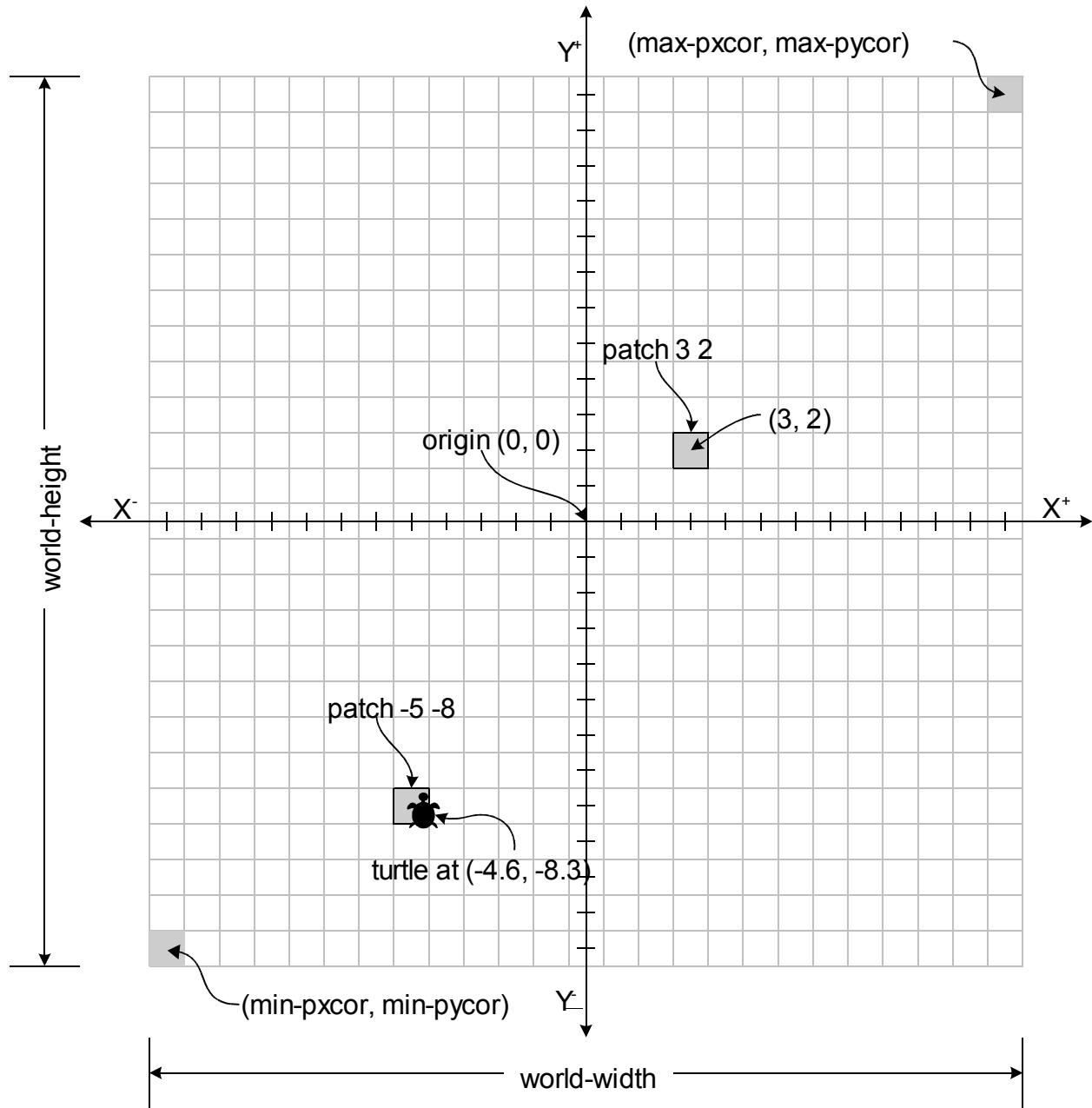
To begin, each turtle will be located on one of the three target points. (Alternatively, we could start with each agent midway between two of the target points, since those midpoints are where all of the agents will move to in the first step, anyway.)

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.

The NetLogo Coordinate System

In setting up the vertices of our triangle (the three target points that will be used by the agents), it will be useful to know a little bit about the coordinate system used by NetLogo. The following diagram illustrates some important points to remember.



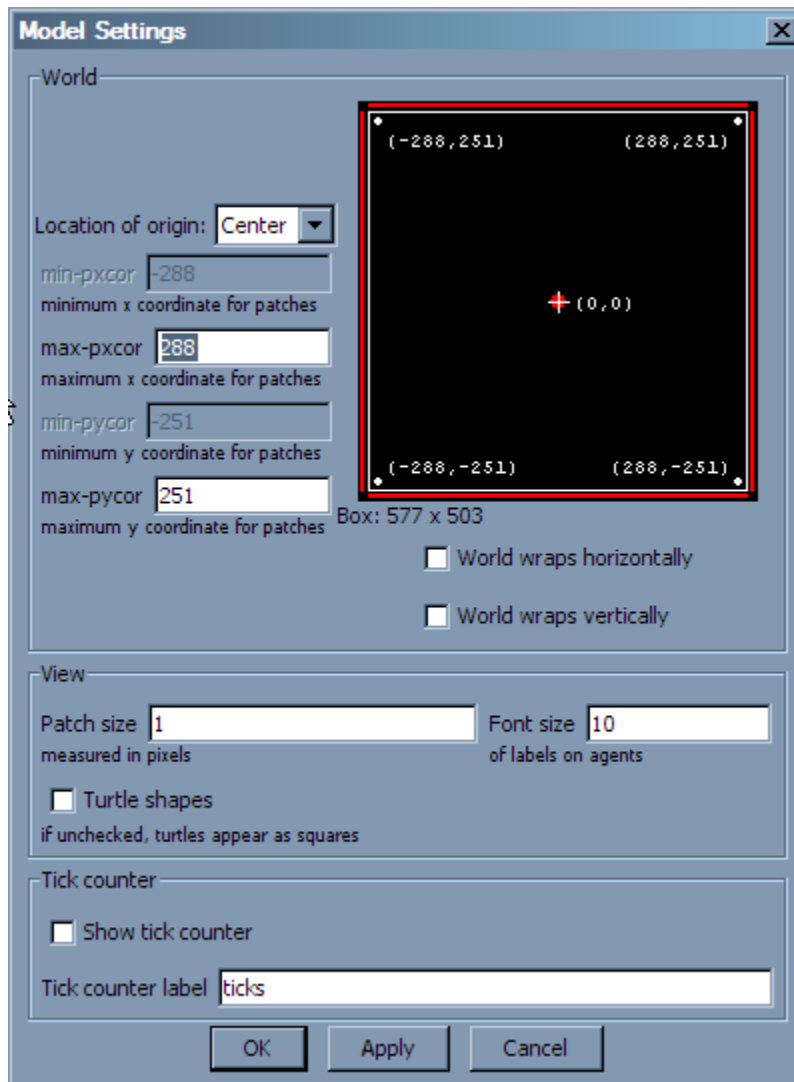
1. Like the Cartesian coordinate system used in Algebra, Analytic Geometry, and Calculus, the NetLogo world has an X and a Y axis. The center of the coordinate system is the origin (which is usually located in the physical center of the NetLogo world, as well), where both X and Y have values of zero (0).
2. Overlaid on the coordinate system is a grid of "patches". Each patch has a color, and an optional label; a NetLogo program can also define additional variables for a patch.
3. The center of a patch is a point in the coordinate system where the X and Y values are integers; these coordinates are used to refer to the patch. For example, `patch 3 2` in the diagram is a square with its center at (3, 2); this square is the region where $2.5 \bullet X < 3.5$ and $1.5 \bullet Y < 2.5$.
4. A patch's coordinates are always integer values, but that is not necessarily true for a turtle on a patch. In the diagram, there is a turtle located at (-4.6, -8.3), which is on the patch centered at (-5, -8). Though a turtle may be drawn so that it looks like it is on two or more patches at once, the turtle's center point is what matters: the center point will be at the coordinate location of the turtle, and the patch in which the center point falls is considered to be the patch on which the turtle is standing.
5. The user can change the width or height of the NetLogo world at any time (in fact, we will do that very thing in this activity); however, a NetLogo program can use `world-width` and `world-height` to get the current values at any time.
6. The patches on the extreme right-hand side of the NetLogo world have an X coordinate of `max-pxcor`; those on the top of the canvas have a Y coordinate of `max-pycor`. Similarly, `min-pxcor` and `min-pycor` are the X and Y coordinates (respectively) of the patches on the extreme left-hand side and bottom (respectively) of the NetLogo world. These variables are related to the overall size of the canvas by the formulas $world-width = (max-pxcor - min-pxcor) + 1$ and $world-height = (max-pycor - min-pycor) + 1$.

Getting Started

Start NetLogo & Configure Canvas

1. From the Macintosh **Applications** folder, or from the Windows **Start/All Program/NetLogo** menu, launch the **NetLogo v4.0.2** 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 will need to set up the NetLogo canvas with a large number of very small patches. To do this, click the **Edit...** button at the top of the canvas, 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 **Turtle shapes** checkbox.
 - h. Uncheck the **Show tick counter** checkbox.

The **World & View** window should now look like this:



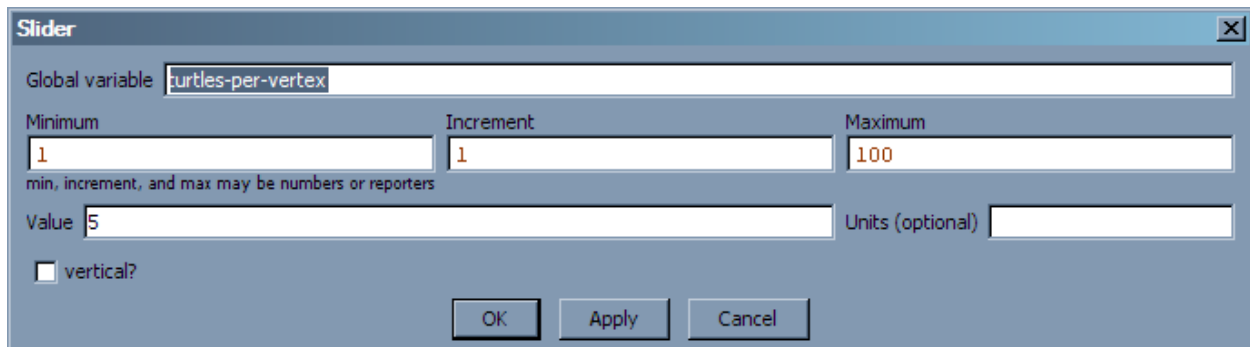
3. Click the **OK** button; we now have a canvas that is 503 patches tall and 577 patches wide, with each patch being 1 pixel X 1 pixel in size.

Controlling the Number of Turtles

One experiment we will do is to see if the results are any different when we use a small number of turtles, vs. a large number. Since all of 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. Click the **Slider** button in the toolbar at the top of the screen.
2. Click somewhere in the white space to the left of the canvas; this will display the **Slider** configuration window.
3. Make the following changes:
 - a. Set the Global **variable** value to "turtles-per-vertex" (leave the quotes out when you type it).
 - 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:



The screenshot shows the 'Slider' configuration window in NetLogo. The window has a title bar with the text 'Slider' and a close button. The main area contains several input fields and a checkbox. The 'Global variable' field is set to 'turtles-per-vertex'. The 'Minimum' field is set to '1', the 'Increment' field is set to '1', and the 'Maximum' field is set to '100'. Below these fields is a note: 'min, increment, and max may be numbers or reporters'. The 'Value' field is set to '5' and the 'Units (optional)' field is empty. At the bottom left, there is a checkbox labeled 'vertical?' which is currently unchecked. At the bottom center, there are three buttons: 'OK', 'Apply', and 'Cancel'.

4. Click the **OK** button.
5. Now we need to write the code that will use the value from this slide to create the specified number of turtles – and make them follow the behavioral rules described previously, of course.

Writing the Model

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

Imagine drawing a triangle on the NetLogo canvas, 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 canvas. As it turns out, the three corners (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 canvas, then coordinates of the top, lower-left, and lower-right vertices are, respectively, $(0, \text{max-pycor} - 10)$, $(\text{min-pxcor} + 10, \text{min-pycor} + 10)$, and $(\text{max-pxcor} - 10, \text{min-pycor} + 10)$. Of course, we just set the dimensions of the screen ourselves (in fact, we set them so that the resulting triangle would be equilateral), so we could just use the numbers we typed into the **World & View** 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 will 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 will be using a variable called "corners" to keep track of the three vertices of the triangle:

```
globals [  
  corners  
]
```

3. Immediately 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):

```
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
```

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. In fact, there will be an error message, indicating that NetLogo doesn't know what we mean by **setup-corner**; don't worry about that error, since we will fix that next. However, if there are other others,

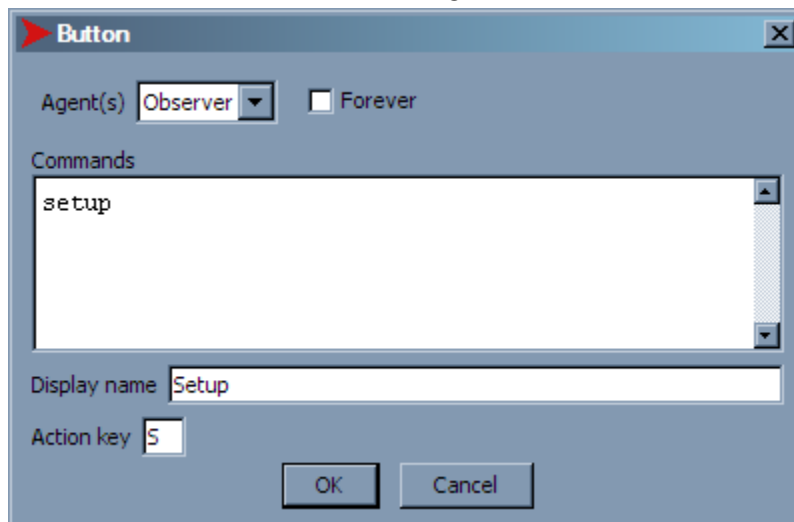
5. Let's review what the `setup` procedure is doing:
 - a. We begin by using `clear-all` to clear the canvas, any turtles, and any variable values.
 - b. Then, we tell NetLogo that the global variable `corners` will hold the set of patches located on the three vertices of our triangle.
 - c. Finally, 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 – we haven't written it! (But we will do so in a moment.)
6. Now would be a good time to save your program. In 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. After the end of the `setup` procedure (again, give yourself some blank lines), write the `setup-corner` procedure:

```
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. Reviewing `setup-corner`:
 - a. We begin by setting the color of the patch to white (that way, all three vertices of our triangle start out white).
 - b. Then we use the `sprout` command, which you might not have seen before. `sprout` is the command a patch uses to create turtles right on that patch, and then to ask those turtles to do something. 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 those turtles to set its color to blue, and to set itself 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 back to the window that shows the NetLogo canvas.

12. Click on the **Button** button on the toolbar near the top of the window.
13. Click somewhere in the white space to the left of the canvas; the **Button** configuration window 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” (leave out the quotes when you type it) 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).

Your **Button** window should look something like this:



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 canvas. 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 them to stop).

In our `setup` procedure, we put the patches that will be the vertices of our triangle in the `corners` variable; they are 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. After the `end` statement that ends the `setup-corner` procedure, type the following code:

```
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 this procedure:
 - a. First, the turtle uses `one-of` to select (at random) one of the set of patches stored in the variable `corners`.
 - b. Now that the turtle has selected one of the vertices, and stored it in the variable `target`, it changes direction, to turn towards the selected vertex. There are a couple of 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 between the turtle and the target vertex.
 - d. Finally, when the turtle comes to rest, it changes the color of the patch on which it is standing (using `set pcolor`), making it 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

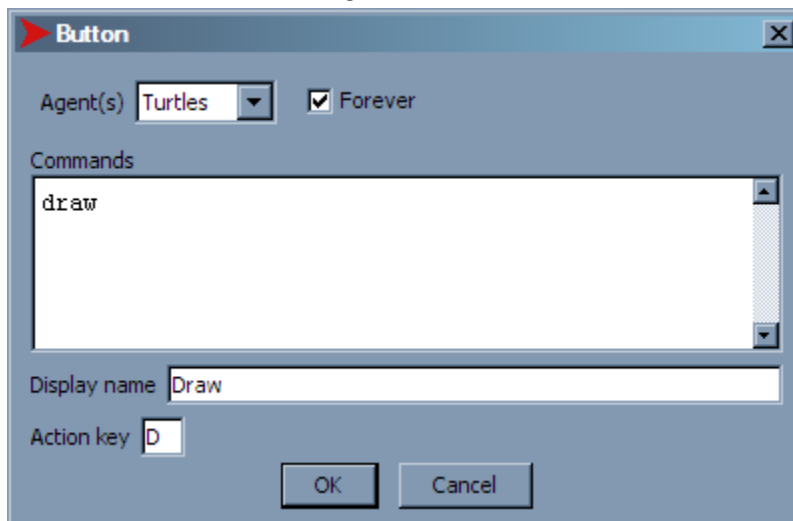
to setup-corner
  set pcolor white
  sprout turtles-per-vertex [
    set color blue
    set size 2.0
  ]
end

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 canvas.
8. Click the **Button** button in the toolbar.
9. Click in the white space to the left of the canvas; the **Button** configuration window will appear.

10. This button will be used to set the turtles into motion, running the **draw** procedure you wrote. So, 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" (without the quotes).
 - d. If desired, type a **Display name** and **Action key**.

When you are done, the **Button** configuration window should look something like this:



11. Click the **OK** button.
12. Save your program.

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?

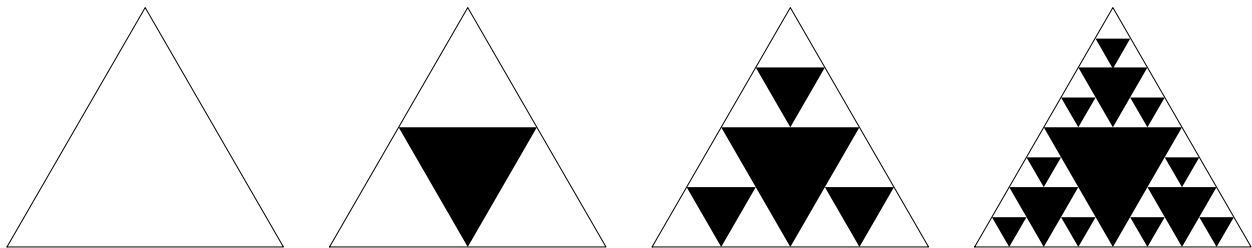
4. Change the value of **turtles-per-vertex** to a higher value, and repeat steps 2 and 3. Has anything changed?
5. Repeat step 2, then slow your turtle movement down, using the slider at the top of the NetLogo world. Now repeat step 3, and watch how the turtles move. How would you describe this movement,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

What is the Mystery Pattern?

The pattern drawn by the turtles is called "Sierpinski's Triangle", or "Sierpinski's 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:



In this activity, we've used an entirely different method to draw the same figure (in fact, there are several methods for producing Sierpinski's Gasket). In this approach, the participants – not just the turtles, but probably the programmers as well – had no idea that the rules being followed would give the same result as that obtained by cutting holes out of triangles. Aside from the fact that the turtles were always moving halfway toward one of the vertices of a triangle, there was little if any hint that the final pattern produced would have anything to do with triangles at all.

Additional Exercises: Modifying the Model

1. There are a number of ways to change a turtle's location. So far, we have used the `forward` procedure, which moves the turtle forward in the direction it is 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 entire path of the turtle.

Modify your `draw` procedure, to use `jump` instead of `forward`. What is 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 modify to control whether a turtle is visible (alternatively, we can use the `hide-turtle` and `show-turtle` procedures).

Modify your `setup-corner` procedure, so that all turtles are invisible.

3. Currently, each turtle modifies the color of the patch on which it lands, setting it to white; it does this whether the patch color has already been turned white by another turtle or not. However, a turtle can condition its behavior on the color of the patch it is on, using an `if` statement.

Modify your `draw` procedure, using an `if` statement to check the color of the patch on which the turtle lands; set the patch color to white only if it is not already white. (Refer to the the NetLogo user manual for more information, if necessary.)

Questions for Reflection on Further Modifications

1. Your model uses an equilateral triangle (the dimensions we set for the NetLogo canvas 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. What do think would happen if we were to start with a shape that wasn't triangular – i.e. with more than three target vertices to choose from? For example, what if we chopped the top off of our triangle, and started with a trapezoid? How difficult do you think it would be to modify your program to answer that question?