NetLogo Tutorial Series:
Langton's Ant

Nicholas Bennett
nickbenn@g-r-c.com

February 2013

Copyright and License

Acknowledgments

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

- Santa Fe Institute Summer Internship/Mentorship (SIM) and Summer Complexity and Modeling Program (CaMP) for high school students;
- New Mexico Supercomputing Challenge;
- Project GUTS;
- New Mexico Computer Science For All.

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

Introduction

One condition which sometimes produces complex behavior is high throughput. This throughput can be the product of a large number of agents acting and interacting simultaneously; a large amount of some physical material or property (e.g. electrical charge, fluid, gas molecules) acting in a constrained space or medium; or even a much smaller number of agents repeating simple behaviors a large number of times in a finite space, with feedback.

In this activity, we'll build a NetLogo model with unpainted (black) and painted patches,[1] inhabited by one or more ants[2] that follow a simple rule. In each time step, each ant will perform the following actions:

1. If the patch where the ant is standing is not painted (i.e. it's black), then:
   a. Turn right 90°.
   b. Paint the patch the color of the ant itself.

   Otherwise:

   a. Turn left 90°.
   b. Clear the patch (i.e. change its color it black).
2. Move forward one patch.

This virtual ant is known as *Langton's ant*, after its inventor Christopher Langton, a pioneer in the field of artificial life [1][2].

We'll begin with a single ant, starting at the center of the NetLogo world. Then, we'll extend the model by adding more ants at different initial positions.

1. What pattern will result from a single ant with the behavior described above?
2. How many repetitions of the above steps will it take for a pattern to emerge?
3. Will the combined behavior be different with two or more ants, instead of one?

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

---

1 In Langton's original description of his ant behaviors, white spaces – instead of black – are considered unpainted.

2 Although the agents in this model don't behave like real ants, that's what we'll call them. Logo agents that move are generically called *turtles*; we'll use *ants* and *turtles* interchangeably in this exercise.

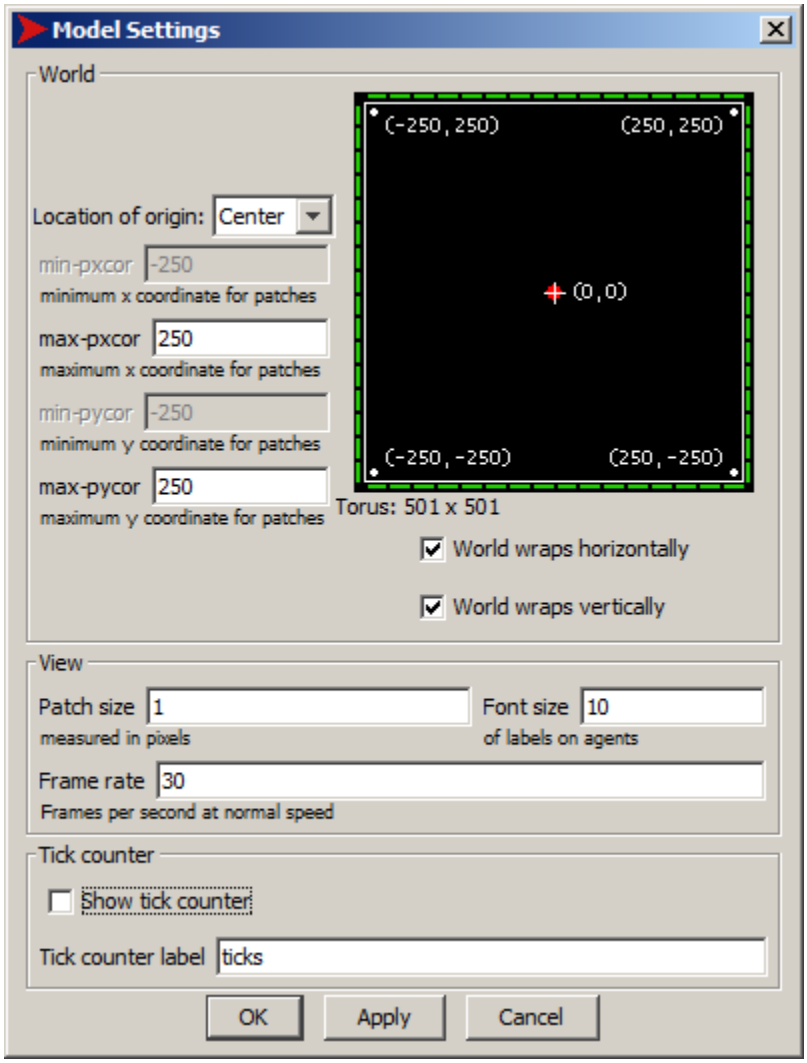
Task 1: Getting Started

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

1. From the Macintosh **Applications** folder, or from the Windows **Start/All Program/NetLogo** menu, launch the **NetLogo** application[3] (not **NetLogo 3D**).
2. Because we want to give our ants lots of room to maneuver, and because NetLogo colors an entire patch at once, let's set up the NetLogo world with a large number of very small patches. To do this, click the **Settings…** button, near the top of the NetLogo window, and make these changes in the **Model Settings** dialog:
   a. Leave **Location of origin** set to **center**.
   b. Set the `max-pxcor` value to 250.
   c. Set the `max-pycor` value to 250.
   d. Make sure the **World wraps horizontally** checkbox is checked.
   e. Make sure the **World wraps vertically** checkbox is checked.
   f. Set the **Patch size** value to 1.0.
   g. Uncheck the **Show tick counter** checkbox.

3 This tutorial was originally written for NetLogo v4.x, then updated for and tested with NetLogo v5.0-v5.0.3. Though we recommend completing this tutorial with NetLogo v5.x, it should work with any 4.x or 5.x version. There are some slight differences in user interface and terminology between the versions, however. For example, there are some checkboxes and other inputs in the **Model Settings** and **Buttons** dialogs of NetLogo v5.x that aren't present in the NetLogo v4.x dialogs; these can be safely ignored when using this tutorial with NetLogo v4.x.

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



3. Click the **OK** button; we now have a world that's 501 patches tall and 501 patches wide, with each patch displayed as a single pixel. Adjust the size and position of the NetLogo application window, as necessary, to display the entire world.

4. Save your model using the **File/Save** menu command. (You should do this after completing each task in this activity.)

Task 2: Setting Up the Model

Since we typically run a model several times in a single session, without closing and re-opening the model (or closing and re-opening NetLogo itself), we need a way to clean up anything left over from a previous run (if any), and prepare the model for a new run. By convention, when building NetLogo models, we usually refer to this cleanup and preparation operation as the model's *setup*; our immediate task is to teach NetLogo how to perform this operation for our ant model.

To teach NetLogo to perform an operation, we instruct it in much the same way we would a very literal-minded person: "To do X, first do A, then do B, then …", with each step spelled out unambiguously. In this case, the first thing we'll tell NetLogo to do is remove any previously created ants, and reset all of the patches to black. Then, we'll tell it to create a single ant, at the origin.

(Before following these steps, we recommend you read "Introduction and Core Concepts – Programming in NetLogo", if you haven't already done so.)

1. In NetLogo, click on the tab or button labeled **Code**. Here you see a blank area for writing the instructions that NetLogo uses to execute the logic of the model.
2. Write the following in the space provided:

```
breed [ants ant]

to setup
  clear-all
  set-default-shape ants "bug"
  create-ants 1
  [
    set color red
    set size 15
    set heading 0
  ]
end
```

The code starts by declaring a **`breed`** of turtle agents called **`ants`**. It then tells NetLogo that the following steps make up a procedure named **`setup`**:

a. Execute the **`clear-all`** command. Among other things, this removes all turtle agents, and sets the color of all the patches to black.

b. Set the shape named "bug" as the default shape for the **`ants`** breed. (To review the shapes available for use, select the **Tools/Turtle Shapes Editor** menu option.)

c. Create a single ant. In the **`create-ants`** command (like the more generic **`create-turtles`** command), the numeric value following the command is the number of agents to be created, and the subsequent set of square brackets contains a list of commands that the newly-created agents are asked to execute. In this case, we are asking our single ant to do the following:

   i. Set its **`color`** to **`red`**. NetLogo uses a spectrum in which each color is identified by a number from 0 to 139.9; however, there are some symbolic constants – such as **`red`** – which are preassigned to the correct number value, so we usually don't have to remember the numbers themselves. (To see the entire NetLogo color spectrum, select the **Tools/Color Swatches** menu option.)

   ii. Set its **`size`** to 15 times the default size. Without this, we wouldn't be able to see the bug shape we've selected for our ant, since we've configured the NetLogo world to use only 1 pixel for each patch, and a turtle of size 1 (the default size) fits inside a single patch.

   iii. Set its initial **`heading`** to 0 (i.e. up or north).

   Note that we didn't include any instructions for setting the initial location of the ant. We'll learn how to do that soon enough – for now, we'll take advantage of the fact that NetLogo initially places turtles at the coordinates (0, 0) – i.e. the origin in the Cartesian coordinate system.

We now need to create a button to execute the new **`setup`** procedure:

3. Click on the **Interface** tab/button, so that the NetLogo world is visible again.

4. On the toolbar at the top of the **Interface** pane, select **Button** from the pull-down menu (next to **Add**); then click on the white space to the right or left of the NetLogo world.

5. In the **Button** properties dialog that appears, make the following changes (as necessary):

   a. From the pull-down menu labeled **Agent(s)**, select **Observer**.

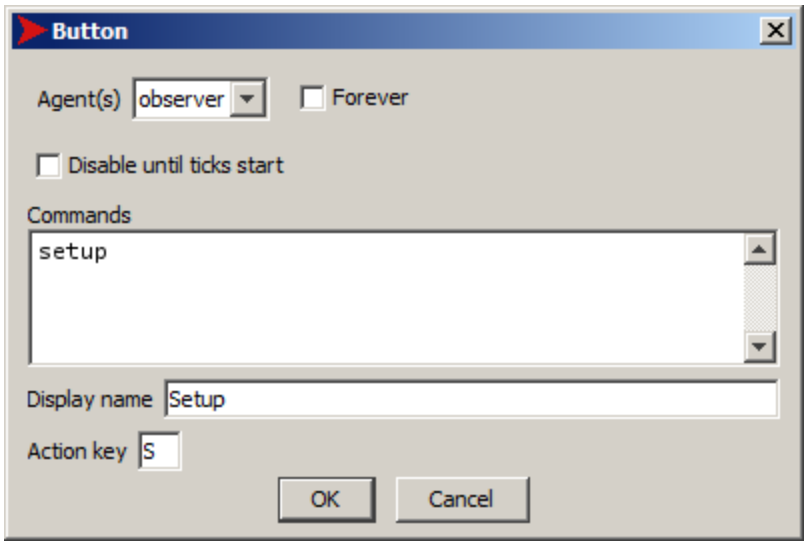
   b. Make sure that the **Forever** and **Disable until ticks start** checkboxes are *not* checked.

   c. In **Commands**, type **`setup`**.

   d. If desired, type the text that you want to appear as the button title in **Display name** (by default, whatever you have typed in **Commands** will be used as a button title).

   e. If desired, specify in **Action key** a shortcut letter that can be used to invoke the button commands with the keyboard.

   Except for possible differences in the optional items, you should now have something like this:



6. Click the **OK** button; you now see your new button in the NetLogo interface.

7. To test your work so far, click the button you created; you should see a single ant in the middle of the NetLogo world.

Task 3: Teaching Our Ant How to Behave

Now we'll write a procedure implementing the ant behavior described at start of this document. By convention, we often call the procedure containing or invoking the main behaviors of our turtles the **go** procedure.

1. Click the **Code** tab/button.
2. *After* the **end** that marks the end of the **setup** procedure, add these lines to your code:

```
to go
  ifelse (pcolor = black)
  [
    right 90
    set pcolor color
  ]
  [
    left 90
    set pcolor black
  ]
  forward 1
end
```

   Be sure to put a space before and after the equals sign in the line that begins with **ifelse** – those spaces, like many in NetLogo, are essential!

   This code tells NetLogo that in order to **go**, a turtle must follow these steps:

   a. If the color of the patch (**pcolor**) where the turtle is standing is black, then:

      i. Turn **right** 90°.

      ii. Set the patch's color (**pcolor**) to be the color of the turtle (**color**).

      Otherwise:

      i. Turn **left** 90°.

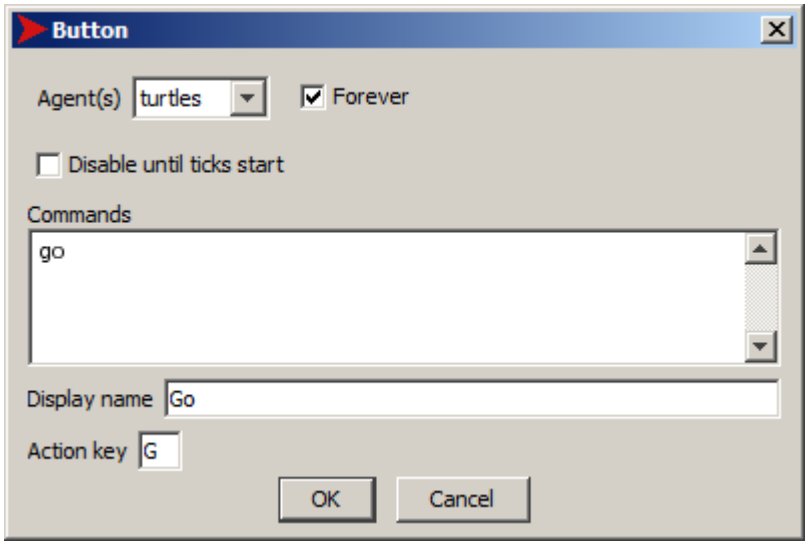
      ii. Set the patch color (**pcolor**) to **black**.

   b. Move **forward** a distance of 1.

Now that we have a procedure containing the steps that we want our ant to follow, let's create a button to tell the ant to execute that procedure repeatedly:

3. Switch back to the **Interface** pane.
4. Follow the same steps as those you previously followed to create the **Setup** button; this time, we'll create a **Go** button.
5. In the button definition dialog that appears, specify the following:
   a. Select **Turtles** from the **Agent(s)** pull-down menu.
   b. Select/check the **Forever** checkbox. (This tells NetLogo to hold the button down when it's clicked, and to ask the specified agents to execute the specified commands repeatedly while the button is down.)
   c. Leave **Disable until ticks start** unchecked.
   d. In **Commands**, type **go**.
   e. If desired, specify a **Display name**.
   f. If desired, specify in **Action key** a shortcut key that can be used to execute the button commands with the keyboard.

   You should now have something like this (again, the **Display name** and **Action key** values are optional; the important parts are the **Agent(s)** selection, the **Forever** and **Disable until ticks start** checkboxes, and the **Commands**):



6. Click the **OK** button to place your **Go** button in the NetLogo interface.

Task 4: Check For Errors – and Fix Them

If the titles of either of the buttons you've created are displayed in red text, that means that NetLogo's telling you there's an error in the button definition, or that the commands specified for the button aren't valid for the type of agent specified. Similarly, if a yellow bar appears at the top of the **Code** window when you switch to another window or when you click the **Check** button, this means that one or more procedures has an error. Pay very close attention to spelling: computers aren't very good at understanding that you meant "turtles", if you wrote "trutles" (for example). Also, note that square brackets and parentheses are not interchangeable in NetLogo; the same goes for dashes, underscores, and spaces (e.g. NetLogo understands **`clear-all`**, but not **`clear_all`**, nor **`clear all`**).

If you see an error message, read it carefully, then use the message and the highlighted code to locate and correct the problem.

Task 5: The Moment of Truth – Running the Model

1. Click the **Setup** button you created, to prepare your model for execution.
2. Click the **Go** button to execute the model. (You may want to use the speed slider, located at the top of the NetLogo world, to slow execution down, so that you can see what's happening more easily.)

Questions for Discussion

1. Are the results what you expected?
2. How would you characterize the results?
3. Were you able to identify different kinds of patterns in the result? Did the ant behavior seem to be more orderly at some times than at others?
4. Did knowing exactly what the ant is doing in each iteration help you to predict what the accumulated effect of thousands of iterations would be?
5. We've configured the NetLogo world as a torus; given that, what do you think would change – in the long-term behavior – if the ant were to start in a different initial position, or face in one of the other three primary directions?
6. Do you think adding a second ant to the model would make the overall behavior more predictable, or less so?

Task 6: Adding a Second Ant to the Model

Before modifying the model, use the **File/Save As…** menu command to save it under a new name; this will ensure that any changes won't result in breaking the model that was working correctly at the end of the last task. (This is a good practice to follow when making a significant change to any working computer program.)

Let's add a second ant to the model. (This time, the code to do this is left up to you.) For now, the new ant should be placed as far as possible from the first. Clearly, the patches located at the four corners of the NetLogo world are as far as it's possible to get from the center patch. Pick any one of these four corner patches, and figure out its *X* and *Y* coordinates in terms of the built-in *reporters*, **`min-pxcor`**, **`min-pycor`**, **`max-pxcor`**, and **`max-pycor`**. (A reporter is simply an instruction that returns a value that can be used in your code. The reporters mentioned here can be found in the **World** section of the NetLogo Dictionary [3].)

Make the necessary additions to the **`setup`** procedure to create a second ant, with the color of your choice, located in the corner you selected, facing south.

Remember that the rules to be followed by the second ant are the same as those followed by the first. Given that, do you think we need any changes to the **`go`** procedure? Do we need a second procedure with the same instructions?

Review the definition of the **Go** button by right-clicking (or c*trl*-clicking) on it, and selecting **Edit…** from the pop-up menu. Do you think that when this button is pressed, only the first ant will execute the **`go`** procedure, or will all ants do so?

Once you've completed the necessary changes to the model (and ensured that two ants are created correctly when you click the **Setup** button), save and run the modified model. Observe the model for several minutes (if necessary), until you're reasonably confident that you can describe the effect of combining the behavior of two ants.

Questions for Discussion

1. Were the results what you expected?
2. Did the combined behavior differ from that observed with only one ant? If so, in what ways did it differ?
3. Do you think the combined behavior would be affected by changing the relative starting positions and headings of the two ants? (See if you can test your predictions.)

Task 7: Giving Each Ant a Random Color, Location, and Heading

In preparation for creating a user-specified number of ants (in the next task), let's create a version of the model we created in task 6 – but this time with random colors, random locations, and random headings for both ants.

If you previously had two **`create-ants 1`** commands in your setup, combine them now into a single **`create-ants 2`** command. In the command block following **`create-ants 2`**, use **`setxy`** with the **`random-pxcor`** and **`random-pycor`** reporters to place each ant on a random patch. For assigning a random color, use the **`one-of`** reporter to select a random color from the base-colors list. For a random heading, you could use arithmetic to transform the value of the **`random`** reporter to obtain one of the four values {0, 90, 180, 270}; alternatively, you could construct a list of the possible headings, and use **`one-of`** to select a random item from that list. For now, don't worry about the possibility that the randomly selected colors, locations, or headings are the same for both ants.

After dealing with any syntax error messages that appear, save and run the modified model. You may need to let it run for some time to for the outcome to become apparent.

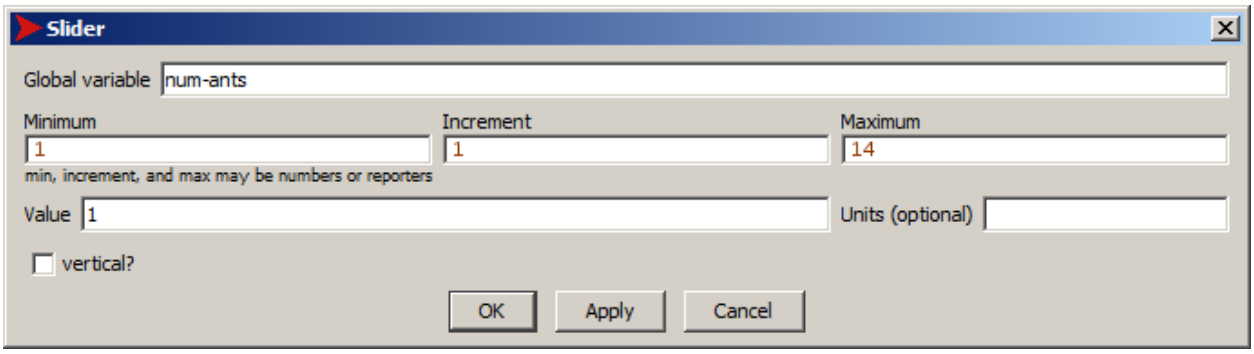
Questions for Discussion

1. Were the results what you expected?
2. Did the combined behavior appear to depend on the relative placement and headings of the turtles?

Task 8: Up to 14 Ants with Random Colors, Locations, Directions

Now that we've seen how to create two ants with random starting locations and headings, we'll create a new version of the model that supports up to 14 ants.

Start by adding a slider to allow the user to set the number of ants to create. The properties of this slider should be similar to these:



Next, modify your **`setup`** procedure to create the specified number of ants, and to place those ants on *distinct*, randomly chosen patches, with each ant facing one of the four main compass directions, chosen at random. Also, give the ants *distinct* colors, selected at random from the NetLogo **`base-colors`** list.

There are multiple ways to complete this task. To place the turtles on distinct patches, you could select a set of patches at random using **`n-of`** and the **`patches`** set. You could then **`ask`** each of those patches to create a single ant, using **`sprout-ants`**. Alternatively, you could use **`setxy`** to place each ant on a random patch, and repeat the placement (with **`while`**) until the ant is on a patch by itself. For assigning distinct random colors, the simplest approach is to use **`one-of`** to select a random item from a list of colors; one complication is that you'll need to find a way to remove each from the list as it's assigned to an ant. NetLogo won't let you modify the **`base-colors`** list, but you can put a copy of that list into a variable you define, and then modify your copy of the list.

Questions for Discussion

1. After previously seeing the results with 1 and 2 ants, were the results with 3 or more ants what you expected?
2. Even though each ant is a different color, their behavior is still conditioned only on whether a patch is black or not. Can you think of some changes to the behavior that would take different patch colors into account? What kinds of patterns might result?

Challenge Tasks: Topology, Manners, and Color Awareness

Topology

So far, our ants have been moving on a toroidal terrain. Are there other topologies that can be modeled in NetLogo, that can be used for Langton's ant? For example, if we disable horizontal wrapping, vertical wrapping, or both, we might implement "bouncing" off the resulting walls: when a step forward is not possible because an ant is at the edge of the world's bounding box, the ant could "bounce" off the wall, and end the step forward exactly where it started, but facing the opposite direction. Could this be implemented as a version of our model? Can you predict the long-term effects of this change, if any? (Before building a NetLogo model to answer the question, try it with pencil, paper, and logic. Consider a short sequence of iterations moving away from the wall vs. the sequence that brought the ant to the wall.)

Manners

Even though it appears that all of the ants are moving at the same time, they're actually taking turns: NetLogo instructs one ant to execute its instruction up to a certain point; it then does the same with the next ant, then the next ant, and so on. Since every ant is repeating a virtually identical sequence of operations (test, turn, paint, move), and since we're using a turtle button to invoke the **`go`** procedure – which causes the turtles agent set to be shuffled only when the button is pressed, rather than every iteration – the ants are taking turns in a *mostly* consistent, *mostly* predictable fashion.[4]

On the other hand, in models where an observer forever button invokes a procedure with an **`ask turtles`** (or, in this case, **`ask ants`**), the order in which turtles take turns changes from iteration to iteration. Would changing our model to work that way (which would have the positive side effect of making it easier to keep a tick count, update plots, etc.) affect the results obtained from 2, 3, or more ants? Is there anyway to have the observer run the **`go`** procedure and still ensure that the agents always take turns in exactly the same order?

Imagine that 2 ants end their respective turns standing on the same unpainted patch. During the next turn, according to the protocol implicit in NetLogo's simulated

4 Actually, when we have a button that is addressed to agents other than the observer, the button is roughly equivalent to an **`ask-concurrent`**. Given that, it's inadvisable to assume anything about how NetLogo switches execution contexts among the agents – even if the agents are all executing the same commands, there's no guarantee that NetLogo is passing control from one agent to another at exactly the same point each time.

concurrency, one of the 2 ants will first flip the state of the patch from unpainted to painted; the other will then flip it back to unpainted. In other words, even though the 2 ants end a turn on an unpainted patch, that patch will still be unpainted at the end of the next turn. Is this appropriate? Can the model be modified to implement cooperation between 2 or more ants starting a turn on the same patch, so that the painted vs. unpainted state of the patch at the end of the turn will be the same as if only 1 ant were on the patch? Will this affect the outcome?

Color Awareness

In the models we've built so far, the ants only distinguish between painted and unpainted patches. Could the models (the single ant model and/or the multi-ant model) be modified so that the ants distinguish colors from an ordered sequence of N colors (including black/unpainted), with a corresponding rule sequence of N characters, each of which is either "L" or "R": the k^{th} character of the sequence would indicate whether a left or right turn should be taken when on the ant is on a square of color k; after turning, the ant would paint the patch with color $(k + 1)$. What would be the effect? (In this alternative, any ant can paint a patch with any of the N colors; thus, the color of the ants themselves are unimportant – indeed, it might be best in this case to hide the ants altogether.)

References

[1] A. Gajardo, "The Industrious Ant of Langton", Mar. 2002. [Online]. Available: http://www.ing-mat.udec.cl/~anahi/langton/. [Accessed: 26 Jan. 2013].

[2] "Christoper Langton", Wikipedia, Jan. 2013. [Online]. Available: http://en.wikipedia.org/wiki/Christopher_Langton. [Accessed: 26 Jan. 2013].

[3] NetLogo Dictionary, Oct. 2012. [Online]. Available: http://ccl.northwestern.edu/netlogo/docs/index2.html. [Accessed: 26 Jan. 2013].