Swarmathon 1: Building a Modular Model

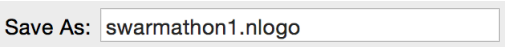
In the Swarmathon competition, teams will be programming real robots that will be searching for tags in an arena set up in a parking lot at Kennedy Space Center. In this module, you will be creating an agent based model that will simulate these real life robots searching for tags using a complete collection method!

Developing models and simulations is a useful tool for testing out different programming and logic ideas for real robots. These models will never be perfect, but if they are written well they can be used to gather useful data without having to reprogram or rebuild robots in between each test.

Part 1: Setting Up The Model

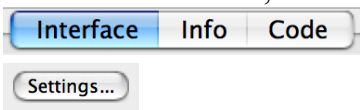
To begin, download the student code which will include the following files:
- parkinglot.jpg
- setup_tags.nls

Open NetLogo and begin with a new model. Save this model in the same directory where you saved the student code files. Save the model as swarmathon1.

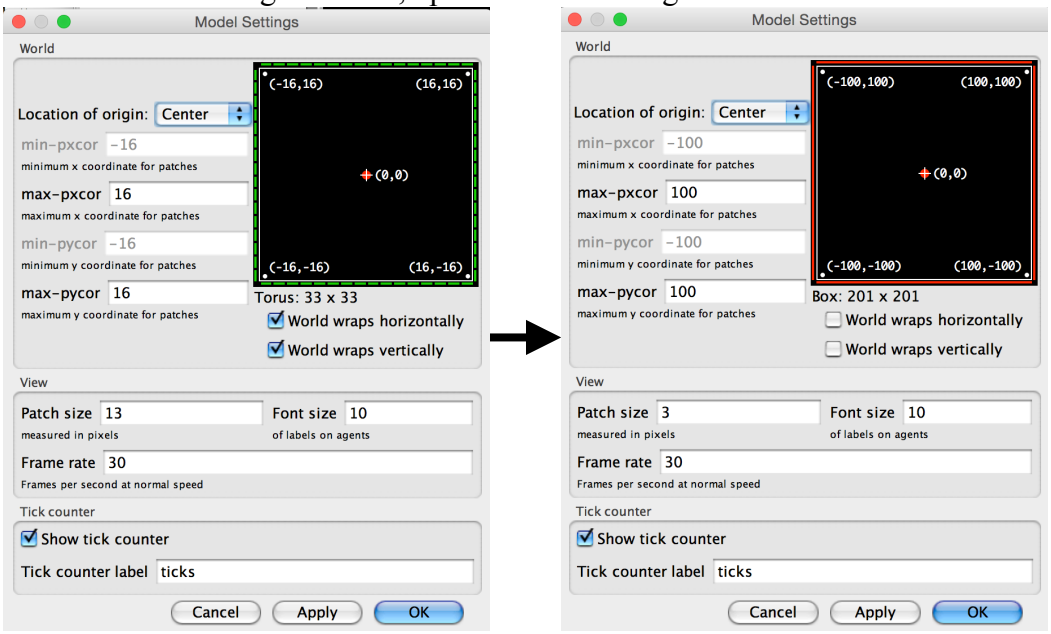


Before we begin writing code for the model, let us set up some important global settings.

1. In the interface window, left click the settings button.



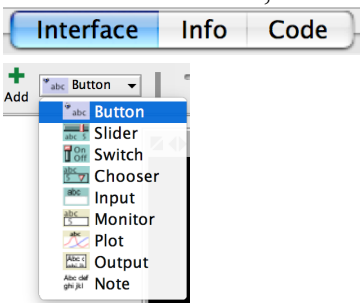
2. Within the model settings window, update the following values:



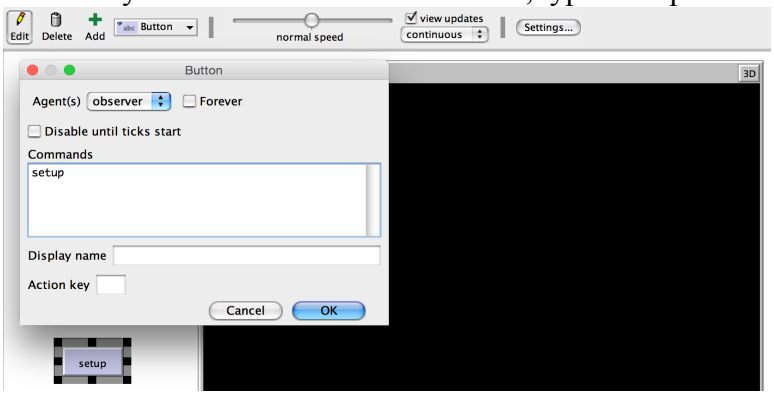
3. Click ok and save your model. Remember to save your work often!

Next, let's set up the buttons, sliders, and other elements that this model is going to need.

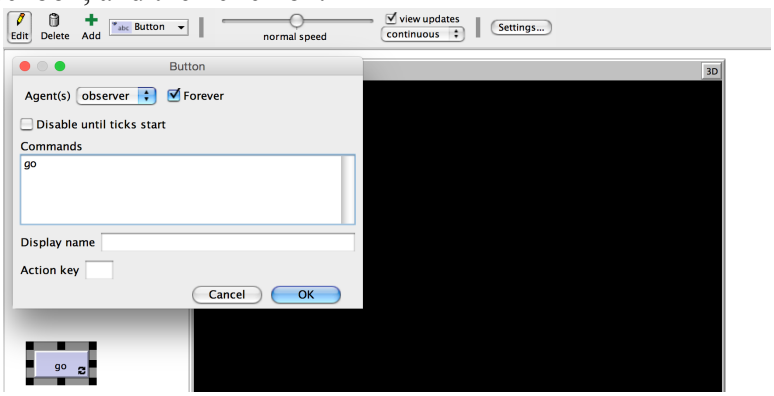
1. In the interface window, use the interface item chooser to add a new button.



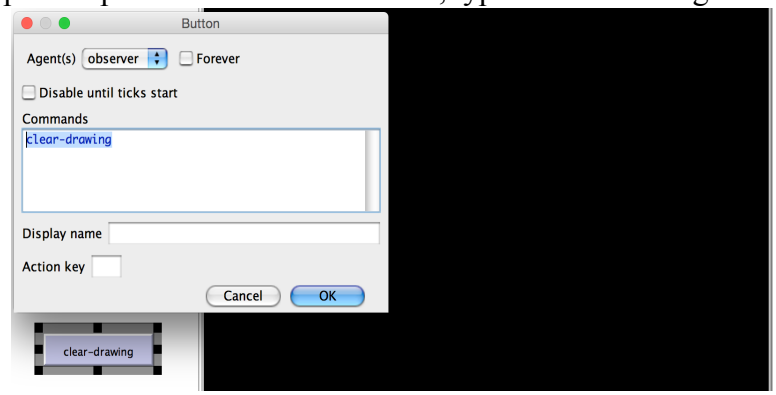
2. Left click anywhere in the interface window, type "setup" in the commands area, and then click ok.



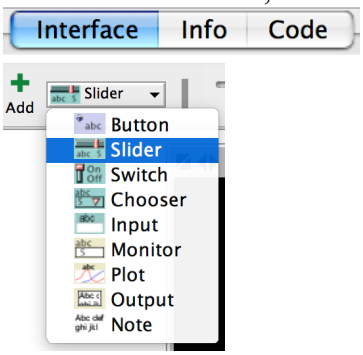
3. Repeat step 1 to create a second button, type "go" in the commands area, make sure to check the "forever" checkbox, and then click ok.



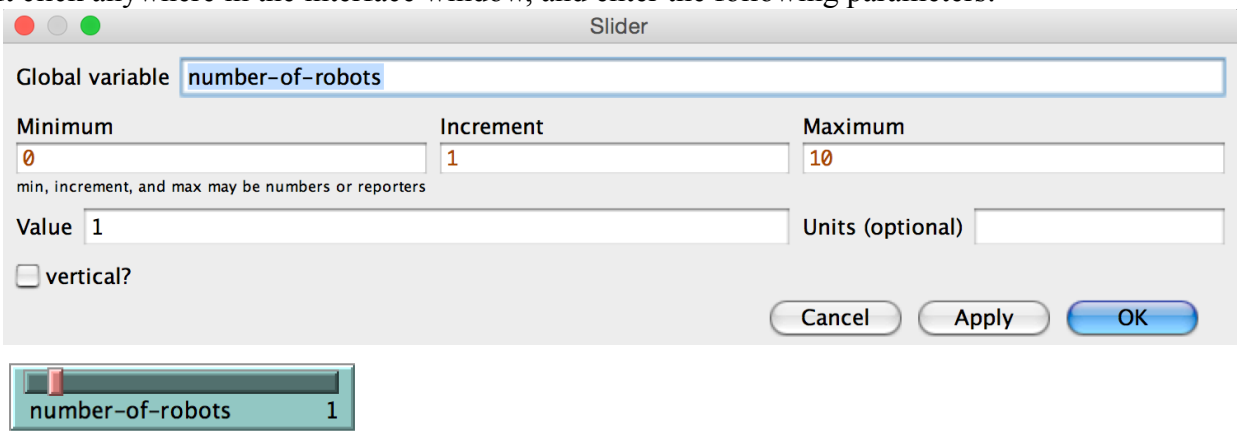
4. Repeat step 1 to create a third button, type "clear-drawing" in the commands area, and then click ok.



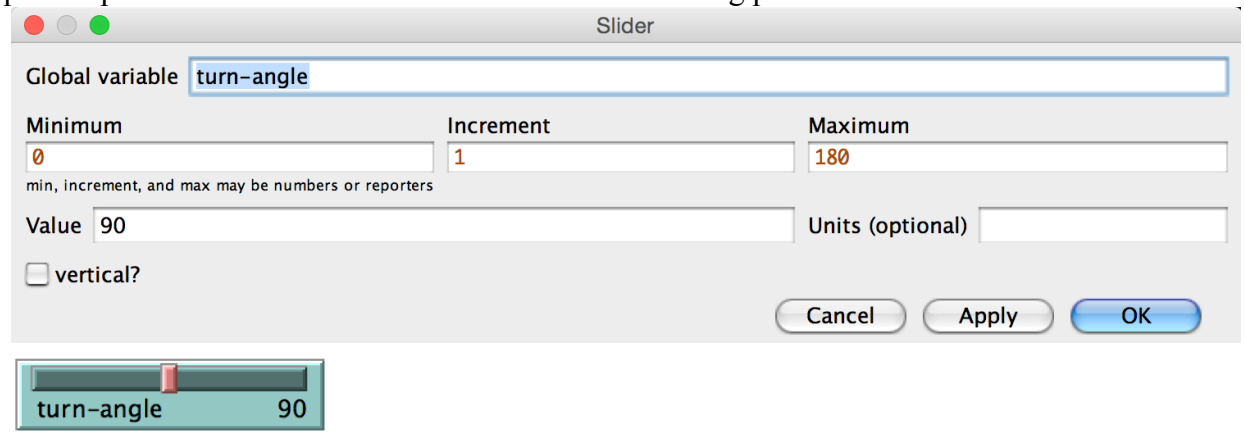
5. In the interface window, use the interface item chooser to add a new slider.

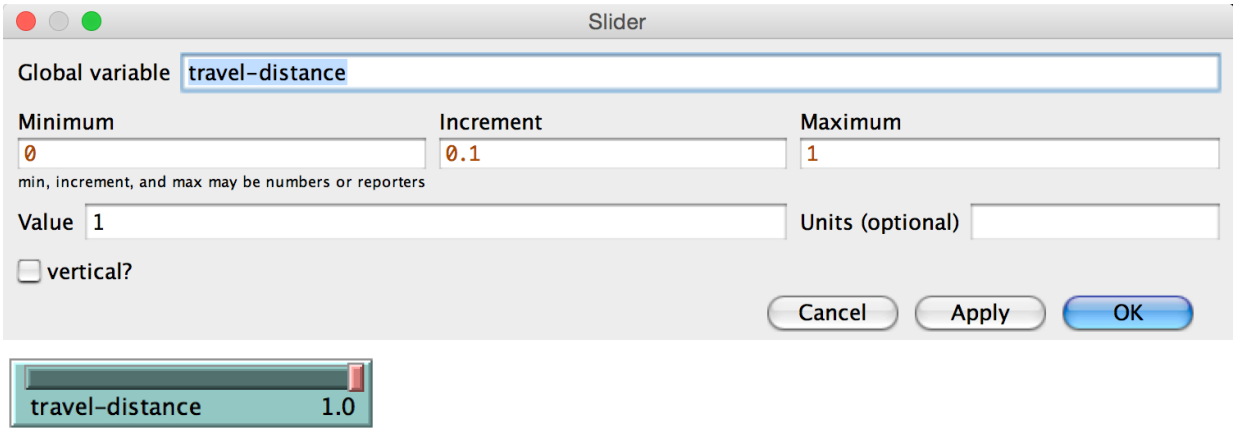


6. Left click anywhere in the interface window, and enter the following parameters:

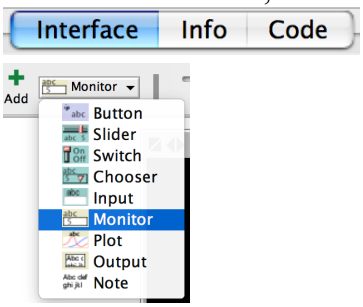


7. Repeat step 4 and create two more sliders with the following parameters:

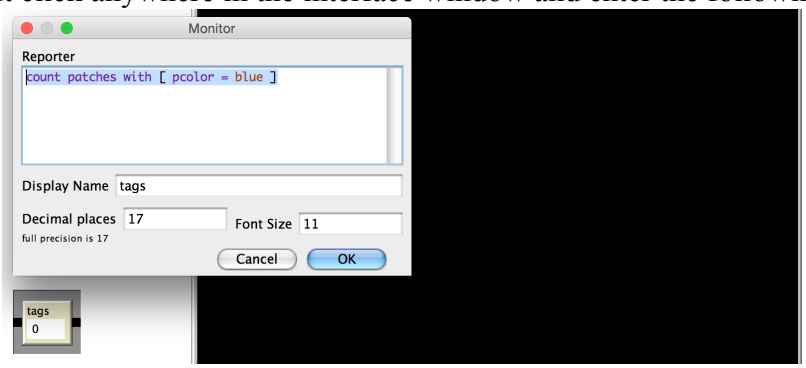




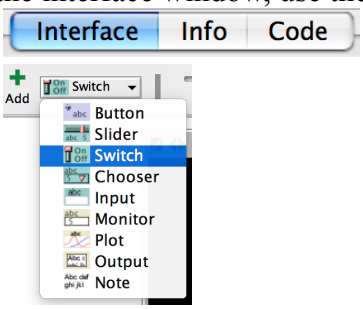
8. In the interface window, use the interface item chooser to add a new monitor.



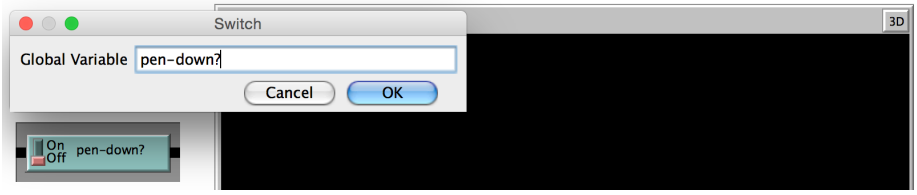
9. Left click anywhere in the interface window and enter the following parameters:



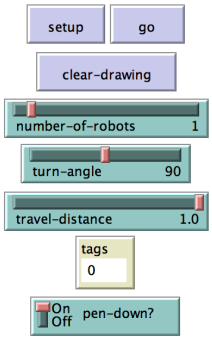
10. In the interface window, use the interface item chooser to add a new switch.



11. Left click anywhere in the interface window and enter the following parameters:



12. Your buttons and sliders should appear similarly to the example below. Don't forget to save your work!



Part 2: Using Multiple Files In One Model

This model is going to use additional files called "include" files. You will be using one include file provided to you, and you will also be writing two of your own files.

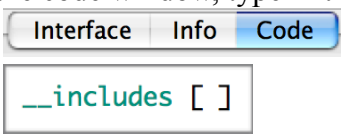
Why should a model have more than one file?

Large models can be split apart into logical pieces that are easier to work with.

In the real world of software development, programming projects can be extremely large. It is often necessary to have them split apart into smaller pieces that many different people can work on. The code will be more organized and easier to write and maintain.

Let's begin by setting up two files that you will use in your model: "setup.nls" and "go.nls"

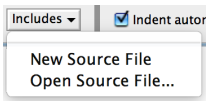
1. In the code window, type in the following code:



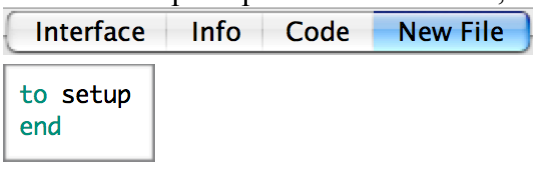
2. Click the check button. NetLogo may move you back to the interface tab. If that happens, simply click back into the code window for the next step.

3. A new menu named "includes" should appear at the top of the code window.

4. Click on the includes button and then click on "New Source File."

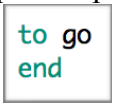


5. A new tab will open up called "New File", inside this tab write the following code:



6. Click file > save, and name the file "setup.nls"
   - Make sure to save the file in the same directory as your model and the student files.
   - It is ***required*** to name this file with a .nls extension so that NetLogo can use it properly.

7. Repeat steps 4, 5, and 6. However, you will name this second file "go.nls" with the following code instead:



We are now ready to begin writing the code for our model!

Part 3: The Main Code Tab

In parts I and II we laid down the groundwork for the model. Our next step will be to add the code that will bring the model to life! We will begin in the main code tab.

1. In the code tab, add the following lines of code:


```
; required for importing the parking lot image as the background
extensions  [ bitmap ]

; each patch should remember its original color from the parking lot image
; the tag color temporarily overwrites this base-pcolor on 256 patches
; once a robot picks up a tag, the patch returns to this base-pcolor
patches-own [ base-pcolor ]

turtles-own
[
  ; the maximum amount of steps a robot can take before it turns
  max-step-count

  ; a counter for the current number of steps a robot has taken
  step-count

  ; state variables for the robot
  ; only 1 of these will be "true", the other is "false", at any given time
  searching? ; the robot is searching for tags? true/false
  returning? ; the robot is returning to the base? true/false
]

__includes
[
  "setup.nls"
  "setup_tags.nls"
  "go.nls"
]
```

We added the empty __includes line previously, now we are going to fill it in with the names of the files we created and the file “setup_tags.nls” provided to you.

2. Click the check button. There should no longer be any errors, but if any remain double check the spelling and syntax of the code with the example.

3. Save your work. We are now ready to write our “setup” procedure.

Part 4: Writing The Setup Procedure

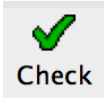
1. In the “setup.nls” file tab, enter the following code:

Interface | Info | Code | setup.nls | go.nls

```
to setup

  no-display
  clear-all
  reset-ticks

  bitmap:copy-to-pcolors bitmap:import "parkinglot.jpg" true
  ask patches [ set base-pcolor pcolor ]

  setup-tags

  create-turtles number-of-robots
  [
    set shape "robot"
    set size 8
    set heading (who * 90)
    set max-step-count 0
    set step-count 0
    set searching? true
    set returning? false
  ]

  display

end
```

The “parkinglot.jpg” file is one of the files included with the student starting code. This image file contains the floor image for our model.

2. Click the check button. There should no longer be any errors, but if any remain double check the spelling and syntax of the code with the example.

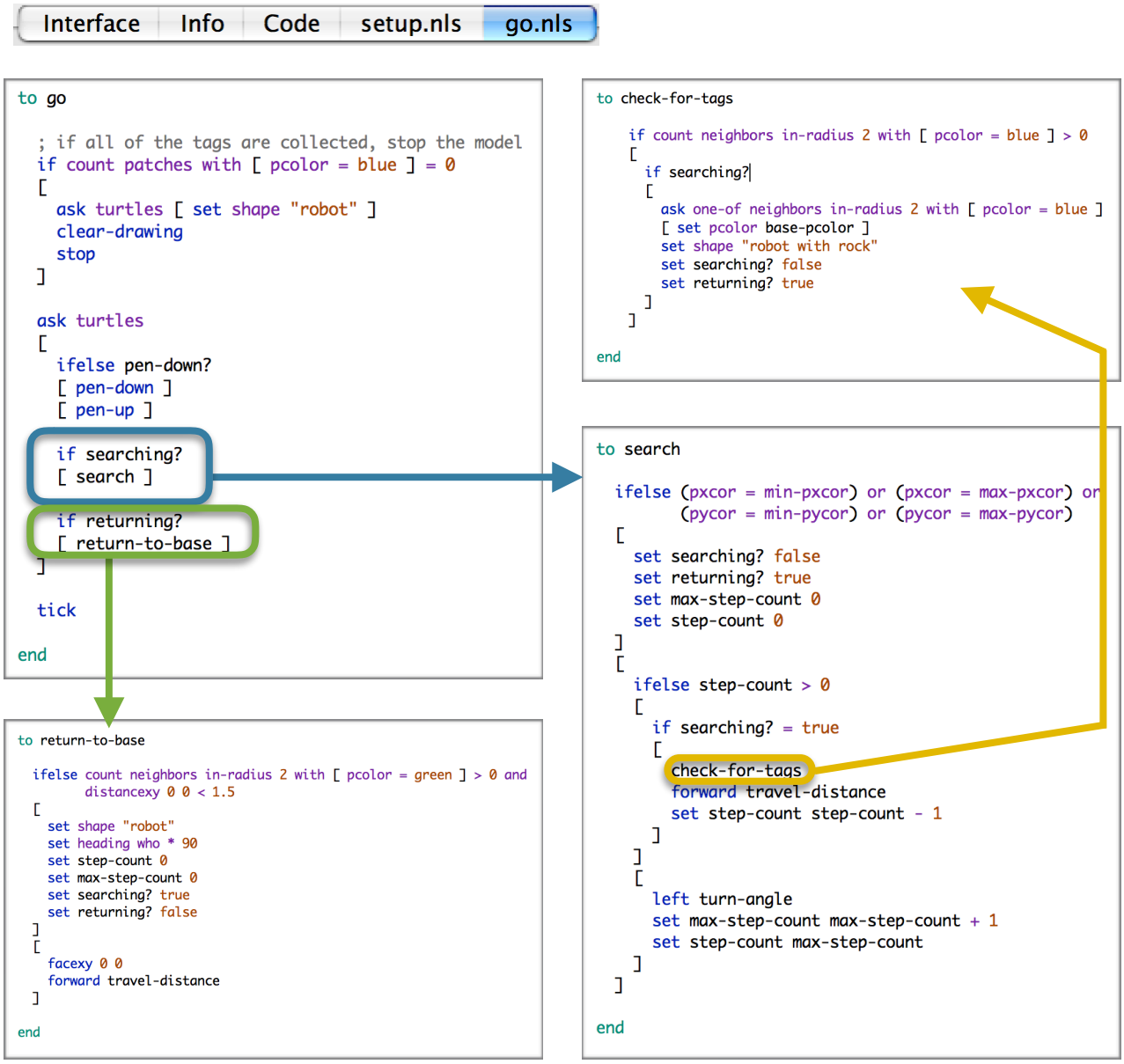


3. Save your work. We are now ready to write the “go” procedure.

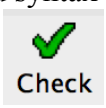
Part 5: Writing The Go Procedure

1. In the "go.nls" file tab, we will be entering code for four procedures:
   A. go
   B. check-for-tags
   C. search
   D. return-to-base

ALL of these procedures will be written in the go.nls file. However, the diagram display of the code below demonstrates how each of these procedures are connected to one another in the logical flow of the model.



2. Click the check button. There should no longer be any errors, but if any remain double check the spelling and syntax of the code with the example.



3. Save your work. We are now ready to test the model!

Part 6: Running And Testing The Model

You can run the model using the following steps:

1. Set the values of the switch and sliders:
   A. the number of robots to go foraging (default = 1 robot)
   B. the turning angle (default = 90 degrees)
   C. the travel distance for each step (default = 1.0)
   D. set the pen-down switch for drawing trails (default = on)
2. Click the “setup” button.
3. Click the “go” button.

As the simulation is running you can click “clear-drawing” to clear the trails being drawn by the robots. In addition, the switch “pen-down?” can be toggled to turn on/off trail drawing during an experiment.

The robot(s) will now be collecting tags as they spiral outwards in a square pattern. Assuming you used the default values, this method is a deterministic search that will explore every possible inch of the space and always collect all the tags. Try experimenting with different settings!

Some questions to consider as you change settings and run additional experiments with the model:

- What are the pros and cons of an exhaustive search?
- Is this the most efficient or fastest way to search? Why or why not?
- How would this compare to random search methods?

The final resulting model is demonstrated below: