swarmathon 4

advanced deterministic search (BFS)

# spiral search

In Swarmathon 1 and 2, we examined biologically-inspired search techniques that employed randomness. In Swarmathon 3, we examined a type of deterministic search called Depth-First Search (DFS). In Swarmathon 4, the final module before you begin your competition submission, we will explore another type of deterministic search called Breadth-First Search (BFS). We will also create a *heterogenous* swarm in which robots can have different behaviors.

## what is BFS?

Breadth-First Search (BFS) is an algorithm used in computer science. A robot performing BFS will pick up the closest resources first. In our simulation, a robot moving in a spiral pattern will always collect the resources closest to the base first. Thus the spiral-robots are performing BFS.

## file setup

As in Swarmathon 1-3, we will be using NetLogo base code. Create a folder called *yourlastname\_Swarmathon4*.Place the *parkingLot.jpg* picture in the folder. Rename the NetLogo file *[Sw4]AdvDetSearchstudentCode.nlogo* to *yourlastname\_Swarmathon4.nlogo* and place the file in the same folder.

# circling towards success

## what do we need to add?

In Swarmathon 4, we want to create a *heterogeneous* swarm where different robots perform different behaviors. To that end, robots have been divided into two breeds: spiral-robots and DFS-robots. The breed of DFS-robots will perform the behaviors we created in Swarmathon 3. In Swarmathon 4, we will program the behaviors for the spiral-robots. To implement spiral search in the robots, we’ll need to code the following behaviors:

main agenda

1. spiral-robots need to know:

* their current stepCount
* their maxStepCount
* if they are in the searching? state
* if they are in the returning? state

1. We need to write a new control procedure that tells robots to perform different behaviors based on their breed (DFS-robots should DFS, spiral-robots should spiral).
2. spiral-robots will look-for-rocks differently than DFS robots.
3. spiral-robots will also need a different return-to-base procedure than the DFS robots.

Let’s begin by tackling agenda item 1:

agenda 1

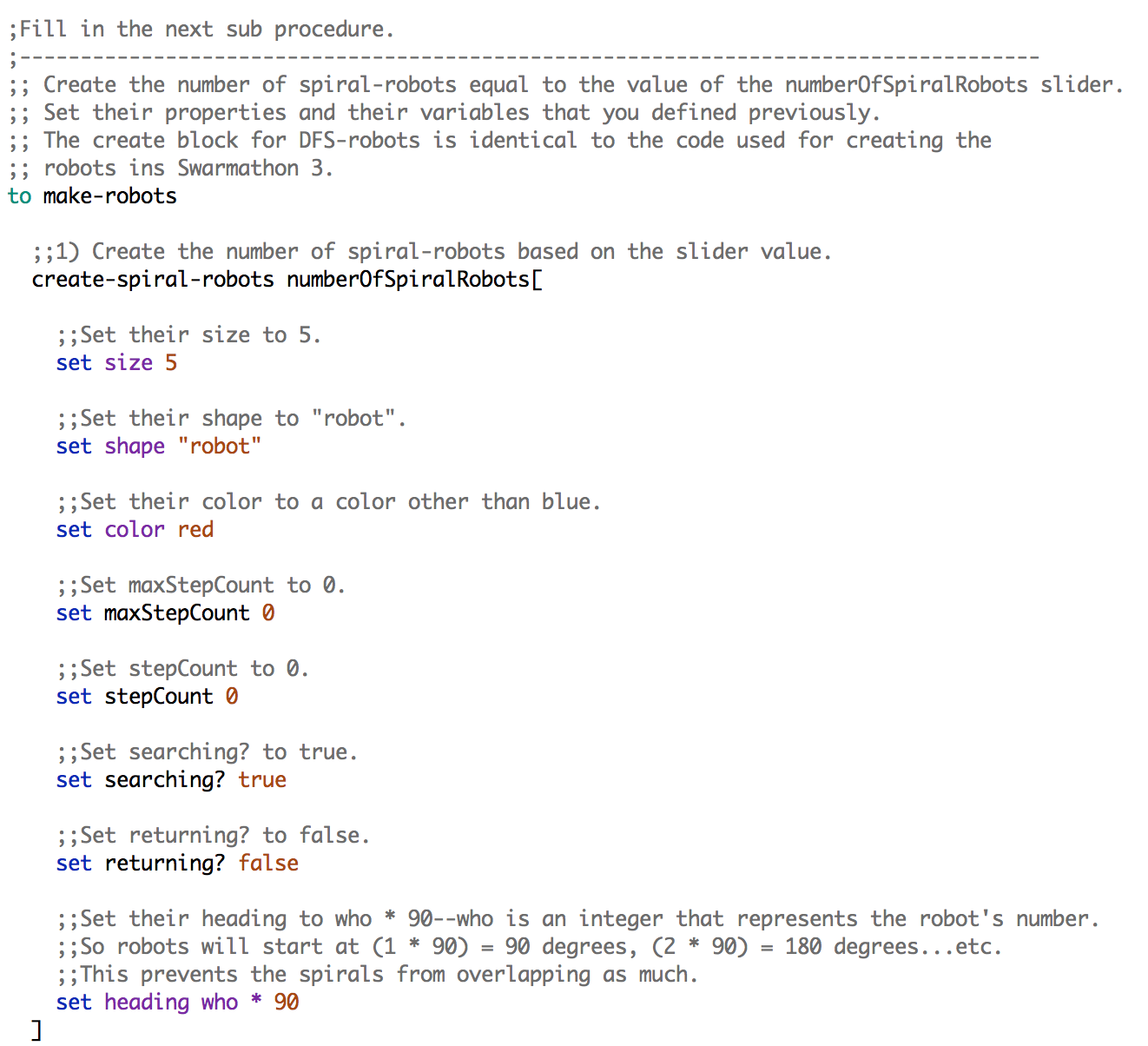
1. spiral-robots need to know:

* their current stepCount
* their maxStepCount
* if they are in the searching? state
* if they are in the returning? state

Scroll to the top of the file. Write sections 1) and 2) using the picture below to create the two breeds of robots and to tell the spiral-robots what they need to know. The DFS-robots behavior is carried over from Swarmathon 3.

Navigate to the Interface tab and notice that some new sliders have been implemented: numberOfSpiralRobots and numberOfDFSRobots. To complete section 2, let’s create a number of spiral robots equal to the value of the numberOfSpiralRobots slider. Let’s set their properties.

Recall that the code for the DFS robots is carried over from Swarmathon 3.



# control is key

Now that the robots are set up, let’s tackle Agenda item 2.

agenda 2

1. We need to write a new control procedure that tells robots to perform different behaviors based on their breed. DFS-robots should DFS, spiral-robots should spiral.

We will write a procedure that will control the robots based on their breed. Write the code in the robot-control procedure as in the picture below.

We have implemented two breeds in Swarmathon 4, but you can add additional breeds. As you write the robot-control procedure, think about other behaviors that we have explored in this series. How could you use breeds to add those behaviors to the swarm?

Notice that you can use the command ask turtles to ask all agents, regardless of what breed they are, to do something. This command allows you to implement both specialized and general behaviors for your robots. Keep it in mind for your Swarmathon competition submission!

# bfs robots

In this final section, we will write the spiral, look-for-rocks, and return-to-base-spiral procedures. These procedures describe the main behavior of the robots.

agenda 3-4

1. spiral-robots will look-for-rocks differently than DFS robots.
2. spiral-robots will also need a different return-to-base procedure than the DFS robots.

## spiral

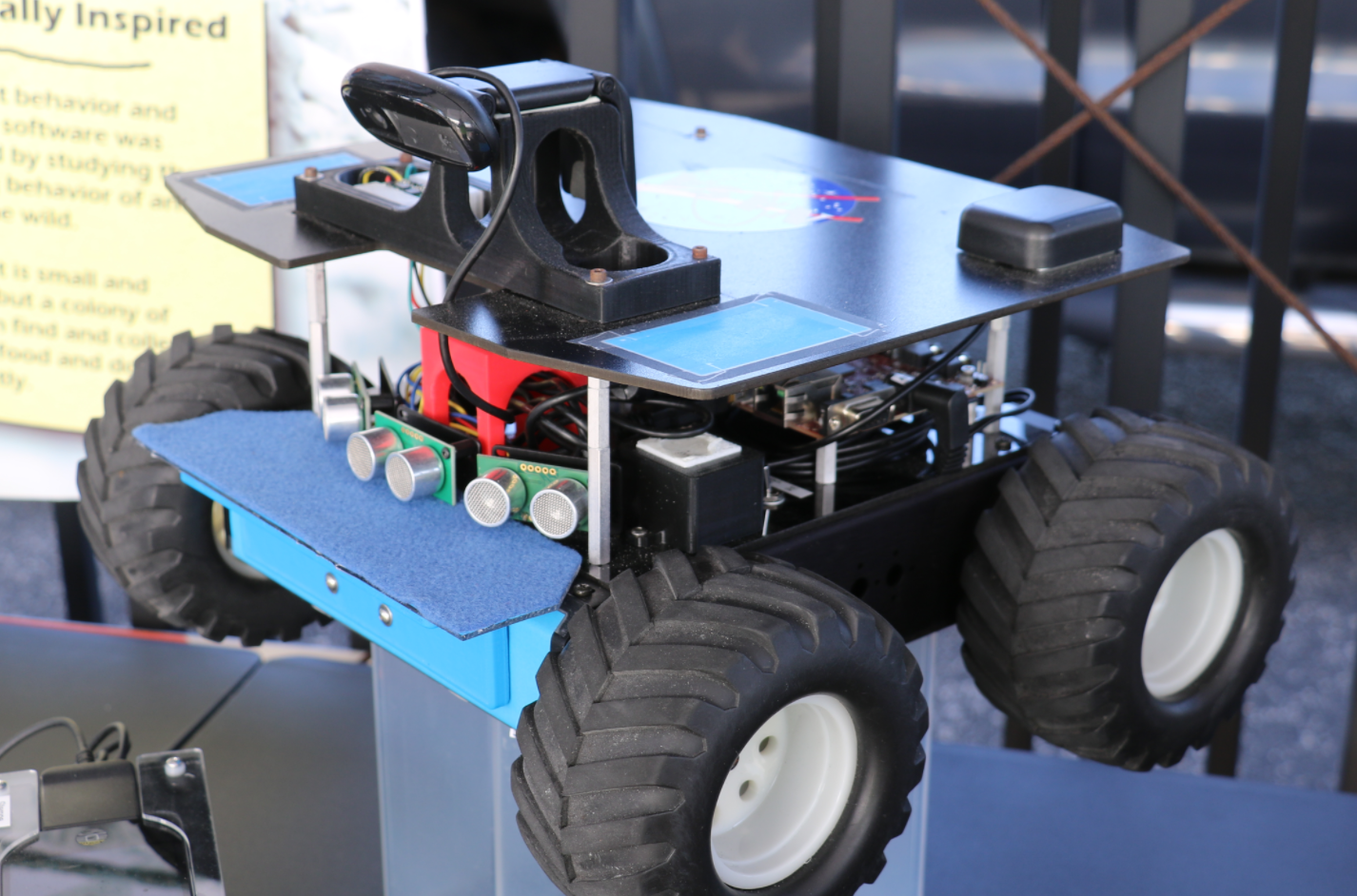
The main procedure, spiral, contains two subprocedures: look-for-rocks and return-to-base-spiral.

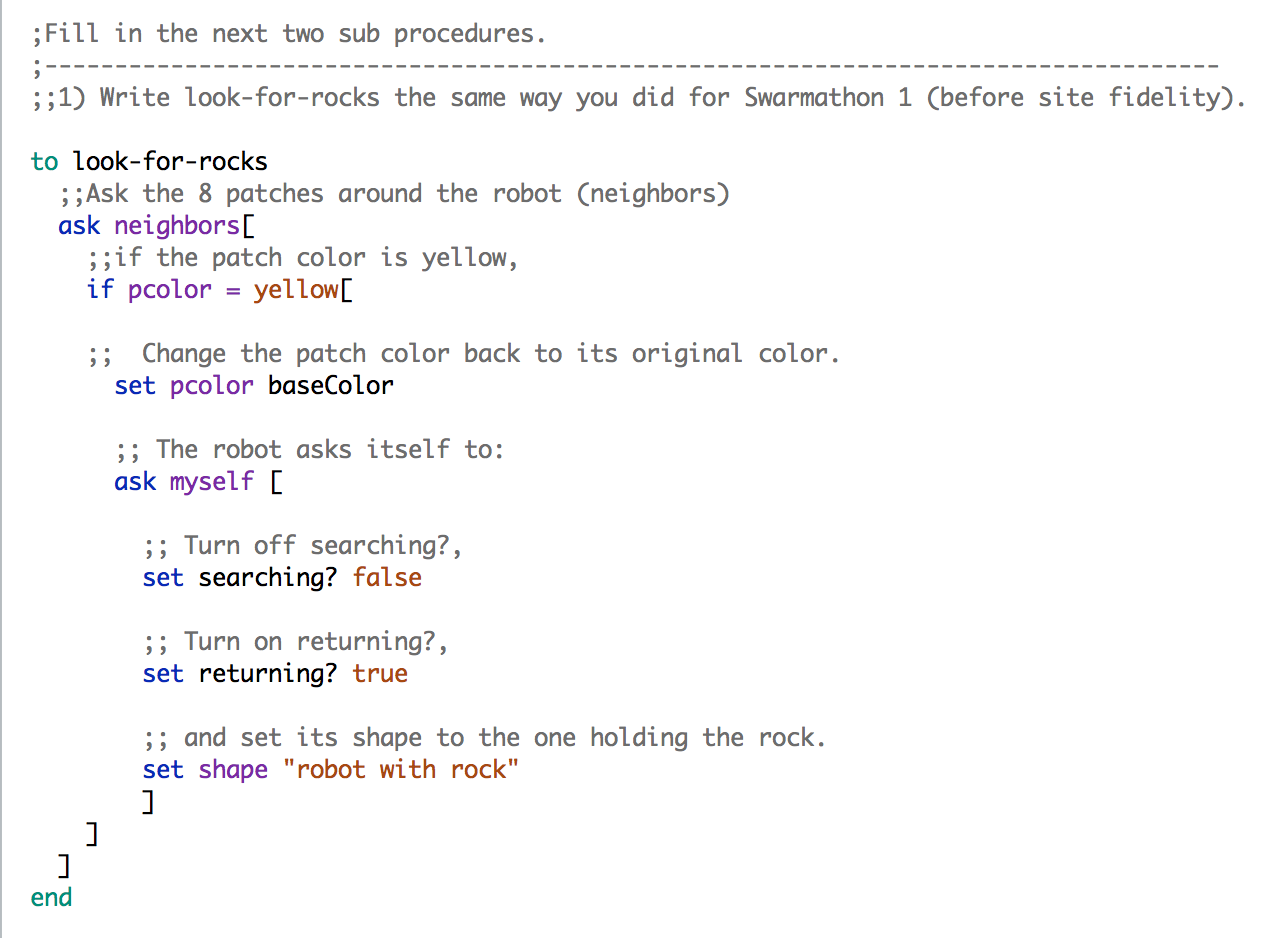
The spiral procedure is heavily commented to help you and is split into two pictures because of length. Write the procedure now.

### spiral part 1

### spiral part 2

## look-for-rocks

The look-for-rocks subprocedure is written the same way as in Swarmathon 1, before we added site fidelity. Since you’ve written this code before, use the comments in the procedure to write look-for-rocks without looking at the code solution.

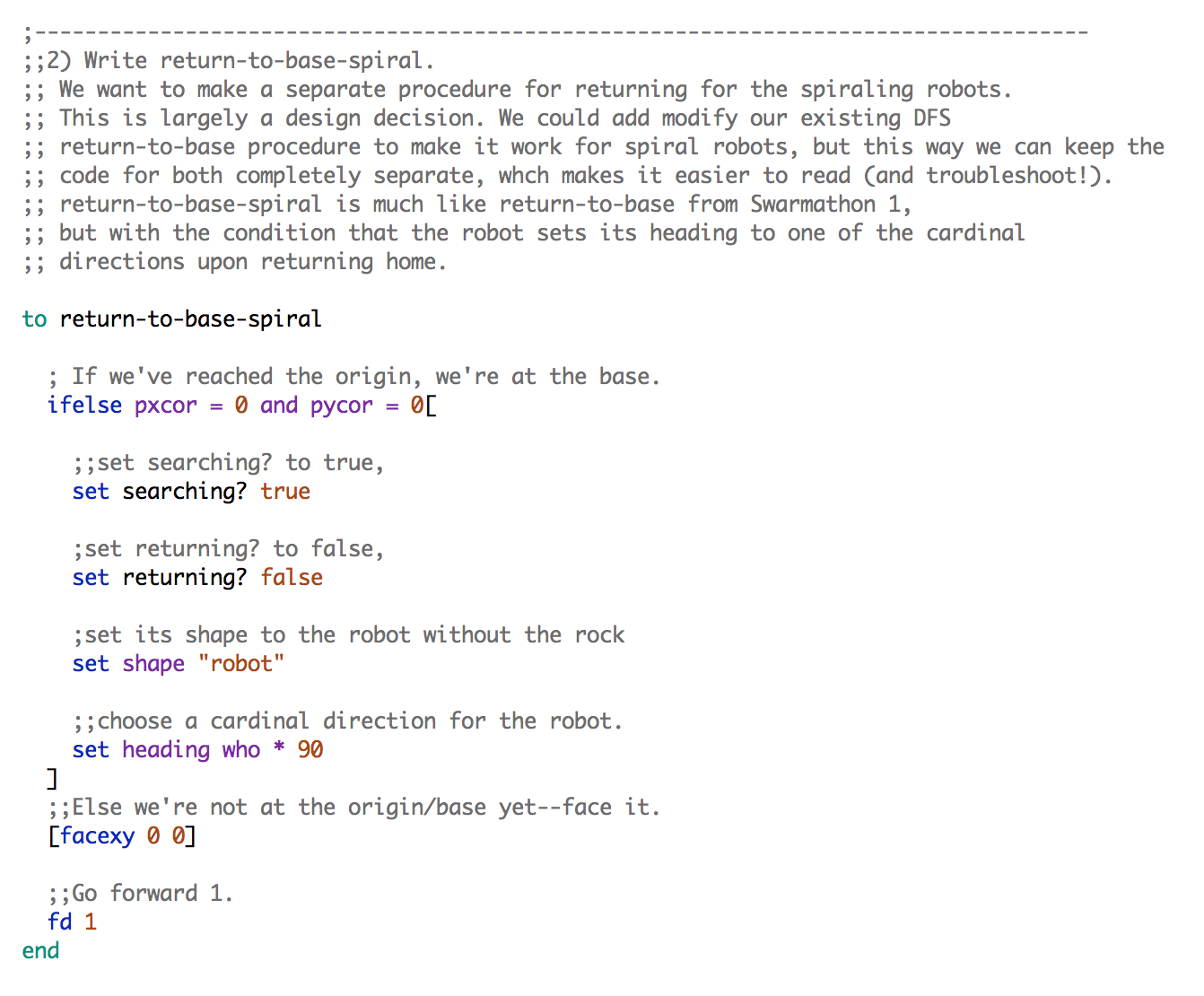
Now check your answers using the picture below:

## return-to-base-spiral

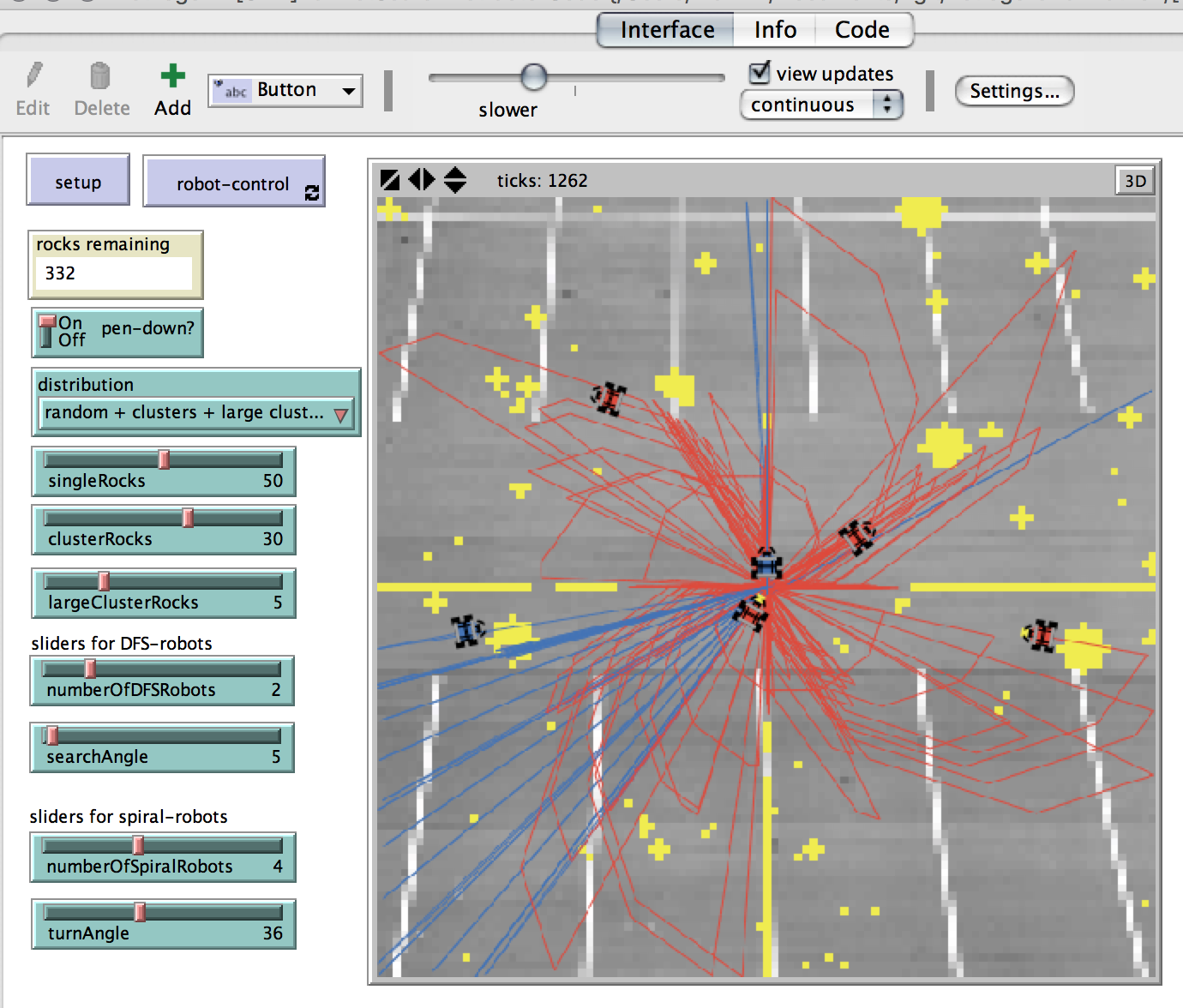
Let’s complete Swarmathon 4 by writing return-to-base-spiral. As you did in Section **4.2**, try to write the code first by using the comments to help you.

The choose a cardinal direction for the robot code may be difficult.

Try to write the rest of the code, then scroll to the next page to check your answers.

Navigate to the Interface tab and be sure that several robots of each breed will be created by setting the sliders for the numbers of each.

Now click setup, and robot-control. Look at them go!



That completes Swarmathon 4.

Notice that throughout this series of tutorials, we have built upon previous code to create more complex behaviors. Before beginning to write your Swarmathon competition submission, look over Swarmathon 1 and 2 again. How did we build from Swarmathon 1 up to Swarmathon 4? How can you modify some of the procedures and subprocedures to create new and interesting behavior in the robots?

GREAT JOB! You completed SWARMATHON 4.



BUG REPORT? FEATURE REQUEST?

Email sherbet@unm.edu with the subject SW4 Report

NEXT UP

SWARMATHON 5: Competition