SWARMATHON 4

ADVANCED DETERMINISTIC SEARCH (BFS)

1 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.

1.1 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.

1.2 FILE SETUP

As in Swarmathon 1 -3, we will be using Netlogo base code. Create a folder called *yourlastname_Swarmathon4.nlogo .* Place the file *[Sw4]AdvDetSearchstudentCode.nlogo* and the *parkingLot.jpg* picture in the folder.

2 CIRCLING TOWARDS SUCCESS

2.1 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
2) 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).
3) spiral-robots will look-for-rocks differently than DFS robots.
4) 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.

```
;;1) use two breeds of robots: spiral-robots and DFS-robots
breed [spiral-robots spiral-robot]
breed [DFS-robots DFS-robot]


;;2) spiral-robots need to know:
spiral-robots-own[
  ;;counts the current number of steps the robot has taken
  stepCount

  ;;the maximum number of steps a robot can take before it turns
  maxStepCount

  ;;is the robot searching?
  searching?

  ;;is the robot returning?
  returning?

]
```

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.

```
;Fill in the next sub procedure.
;-------------------------------------------------------------------------------
;; Create the number of spiral-robots equal to the value of the numberOfSpiralRobots slider.
;; Set their properties and their variables that you defined previously.
;; The create block for DFS-robots is identical to the code used for creating the
;; robots ins Swarmathon 3.
to make-robots

  ;;1) Create the number of spiral-robots based on the slider value.
  create-spiral-robots numberOfSpiralRobots[

    ;;Set their size to 5.
    set size 5

    ;;Set their shape to "robot".
    set shape "robot"

    ;;Set their color to a color other than blue.
    set color red

    ;;Set maxStepCount to 0.
    set maxStepCount 0

    ;;Set stepCount to 0.
    set stepCount 0

    ;;Set searching? to true.
    set searching? true

    ;;Set returning? to false.
    set returning? false

    ;;Set their heading to who * 90--who is an integer that represents the robot's number.
    ;;So robots will start at (1 * 90) = 90 degrees, (2 * 90) = 180 degrees...etc.
    ;;This prevents the spirals from overlapping as much.
    set heading who * 90
  ]
```

3 CONTROL IS KEY

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

AGENDA 2

2) 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.

```
;-------------------------------------------------------------------------------
 ;;;;;;;;;;;;;;;;;;;;;;;;
 ;;    ROBOT CONTROL    ;; : MAIN PROCEDURE
 ;;;;;;;;;;;;;;;;;;;;;;;;
 ;-------------------------------------------------------------------------------
 ;;1) Write the robot-control procedure. The different breeds of robots will perform
 ;; different behaviors.

to robot-control

  ;;ask the DFS-robots to DFS.
  ask DFS-robots[DFS]

  ;;ask the spiral-robots to spiral.
  ask spiral-robots[spiral]

  ;; We can use 'turtles' to ask *all* agents to do something.


  ;; Ask the turtles
  ask turtles [
    ;; Use an ifelse statement.
    ;;If the pen-down? switch is on, put the pen down
    ifelse pen-down?
    [pen-down]

    ;;Else take the pen up.
    [pen-up]
  ]
   tick ;;tick must be called from observer context, move into main procedure.
end
```

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!

4 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

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

4.1 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.

4.1.1 SPIRAL PART 1

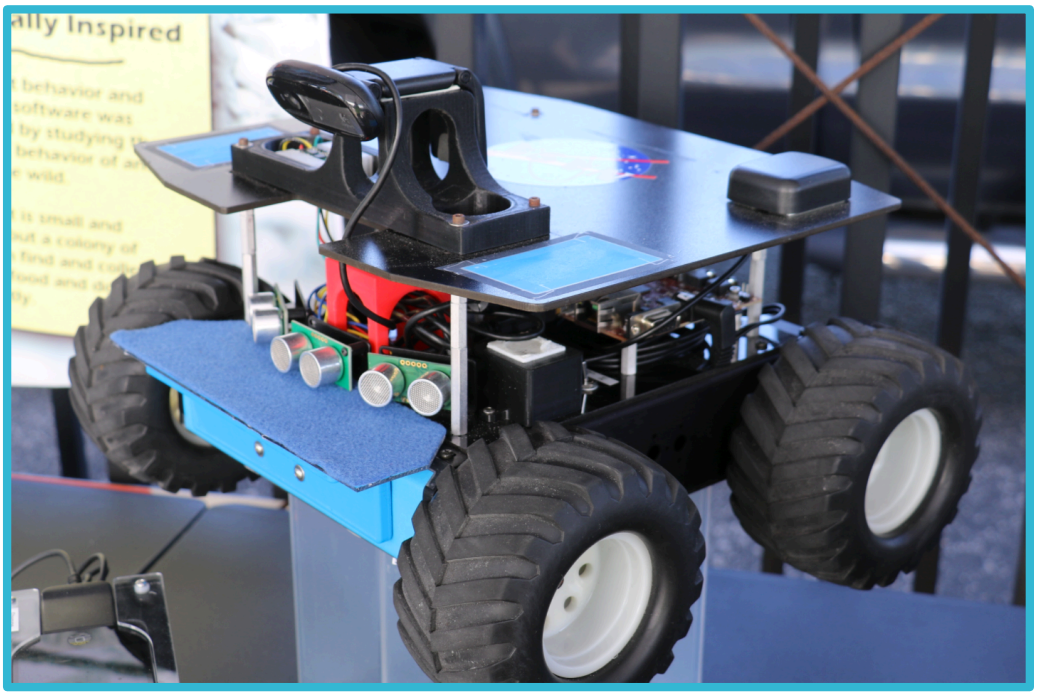
```
;;;;;;;;;;;;;;;;;;;
;;    spiral     ;; : MAIN PROCEDURE
;;;;;;;;;;;;;;;;;;;
;-------------------------------------------------------------------------------
;;1) Write the spiral procedure.
to spiral

  ;;If the robots can't move, they've hit the edge.
  ;;They need to go back to the base and start a new spiral.
  ;;Also reset their variables so they can start over.
if not can-move? 1 [

  ;;Set returning? to true to get them to go back to the base.
  set returning? true

  ;;Set stepCount and maxStepCount back to 0.
  set stepCount 0
  set maxStepCount 0
  ]

;;If they are returning? they should do the return-to-base-spiral procedure.
if returning? [return-to-base-spiral]

;;The following code makes a spiral.
;;The robot increases the distance it travels in a line
;;before making a left turn.

;;if the robot's stepCount is greater than 0,
ifelse stepCount > 0[

  ;;if the robot is searching?
  if searching?[

    ;;Go forward 1.
    fd 1

    ;;look-for rocks,
    look-for-rocks

    ;;then reduce stepCount by 1.
    set stepCount stepCount - 1
  ]
]
```

4.1.2 SPIRAL PART 2

```
;Else, no steps remain.
[

  ;; the robot should turn left based on the value of turnAngle,
  left turnAngle

  ;;increase the maxStepCount by 1,
  set maxStepCount maxStepCount + 1

  ;;then set the value of stepCount to the value of maxStepCount.
  set stepCount maxStepCount
  ]
end
```

4.2 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.


Ally Inspired

Now check your answers using the picture below:

```
;Fill in the next two sub procedures.
;-------------------------------------------------------------------------------
;;1) Write look-for-rocks the same way you did for Swarmathon 1 (before site fidelity).

to look-for-rocks
  ;;Ask the 8 patches around the robot (neighbors)
  ask neighbors[
    ;;if the patch color is yellow,
    if pcolor = yellow[

    ;;  Change the patch color back to its original color.
      set pcolor baseColor

      ;; The robot asks itself to:
      ask myself [

        ;; Turn off searching?,
        set searching? false

        ;; Turn on returning?,
        set returning? true

        ;; and set its shape to the one holding the rock.
        set shape "robot with rock"
        ]
    ]
  ]
end
```

4.3 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.

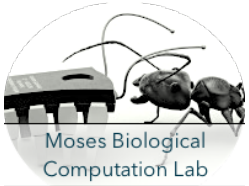
The `choose a cardinal direction for the robot` code may be difficult.

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

```
;-------------------------------------------------------------------------------
;;2) Write return-to-base-spiral.
;; We want to make a separate procedure for returning for the spiraling robots.
;; This is largely a design decision. We could add modify our existing DFS
;; return-to-base procedure to make it work for spiral robots, but this way we can keep the
;; code for both completely separate, whch makes it easier to read (and troubleshoot!).
;; return-to-base-spiral is much like return-to-base from Swarmathon 1,
;; but with the condition that the robot sets its heading to one of the cardinal
;; directions upon returning home.

to return-to-base-spiral

  ; If we've reached the origin, we're at the base.
  ifelse pxcor = 0 and pycor = 0[

    ;;set searching? to true,
    set searching? true

    ;set returning? to false,
    set returning? false

    ;set its shape to the robot without the rock
    set shape "robot"

    ;;choose a cardinal direction for the robot.
    set heading who * 90
  ]
  ;;Else we're not at the origin/base yet--face it.
  [facexy 0 0]

  ;;Go forward 1.
  fd 1
end
```

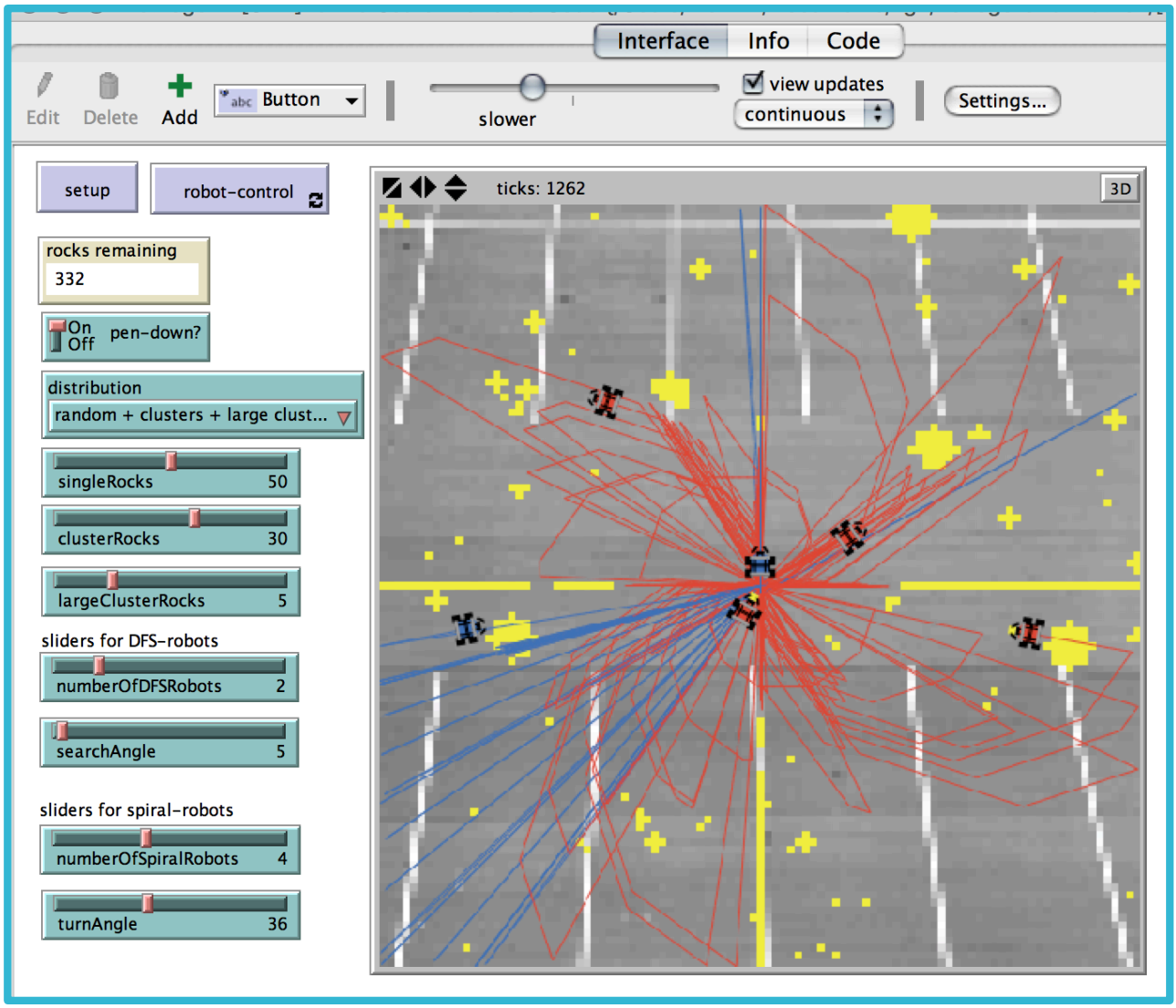
That completes Swarmathon 4.

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?



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!



GREAT JOB! You completed SWARMATHON 4.

BUG REPORT? FEATURE REQUEST?

email elizabeth@cs.unm.edu with the subject SW4 report

NEXT UP
SWARMATHON 5: competition

Moses Biological Computation Lab