

# SWARMATHON 1

## INTRO TO BIO-INSPIRED SEARCH

---

### 1 SWARM ROBOTS ON MARS



*nasa.gov*

#### 1.1 BACKGROUND

The Moses Biological Computation Lab at the University of New Mexico builds and does research with robots. The robots are *biologically-inspired*. They take inspiration from nature by simulating the behavior of swarms of ants, who are exceptionally good at finding and gathering resources as a group.

The goal of the research done by the Moses Biological Computation Lab is to develop robots that can be placed on Mars and other distant planets and asteroids to gather resources such as rocks and ice. We need these materials for building and creating a steady water supply. The robots will help us to colonize Mars! To this end, the lab is collaborating with NASA on the Swarmathon competition. By using the tools presented to you in this series of modules, you will be able to create your own submission to the high school NetLogo division of the Swarmathon competition—and help send robots to Mars!

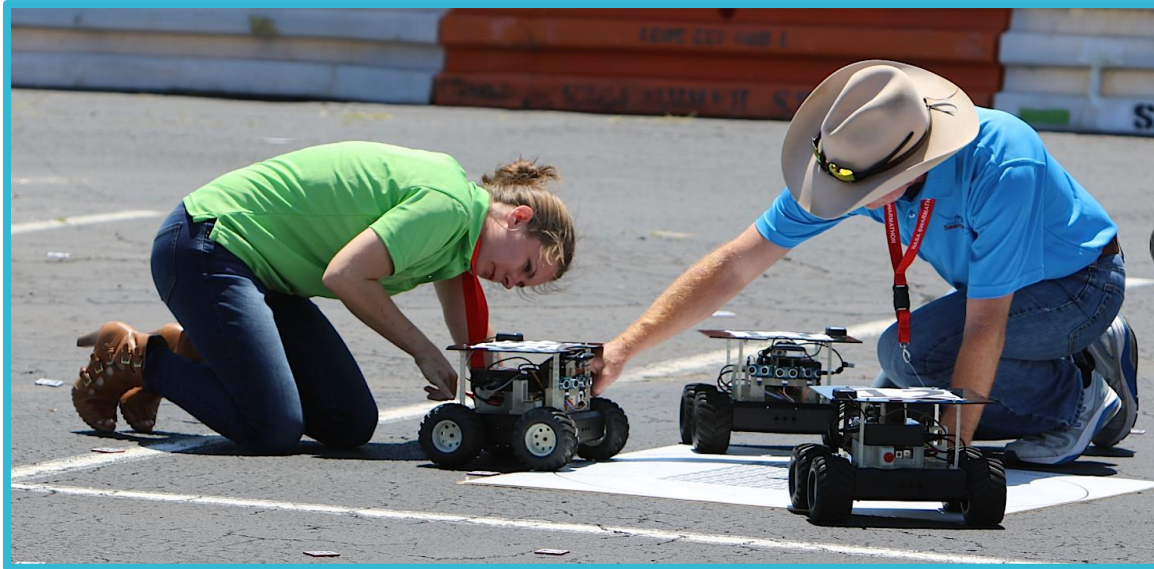
## 1.2 ACTIVITY DESCRIPTION

In this exercise, you will create a NetLogo model that simulates the Moses Biological Computation Lab's Swarmie robots collecting rocks on Mars. You will create robots that search for individual rocks and bring them back to a central base. Additionally, you will learn about and implement a *biologically-inspired* strategy that ants use to collect resources more efficiently: **site fidelity**.

Follow the instructions step-by-step. Don't skip ahead! Some parts of the program are already done for you. While you are working, **don't change the code that's already been written, or the program may not run**. Once you've finished the program, you may experiment by changing the code.



## 2 GETTING STARTED AND SETTING UP



*nasaswarmathon.com*

Let's setup our NetLogo file.

### 2.1 FIRST STEPS

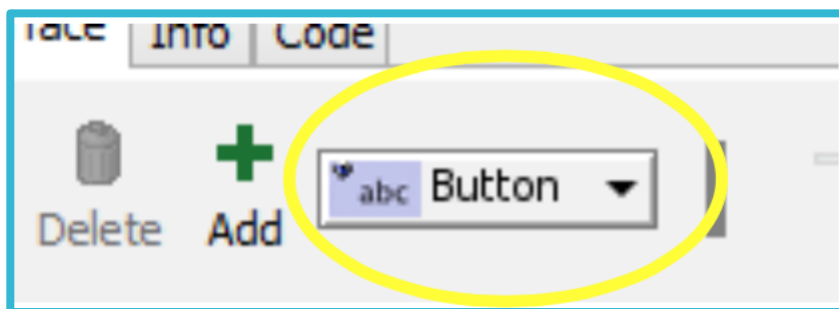
- Create a folder and name it *yourlastname\_Swarmathon1*. Your instructor will tell you how to access the .jpg and the .nlogo file.
- Place both the .jpg and the .nlogo file in the same folder.
- Double-click the .nlogo file. It should open in NetLogo. If not, open NetLogo directly, go to File->Open, and navigate to the .nlogo file.
- Click the Code tab at the top of the screen.

```
;;;;;;;;;;;;;;  
;;  Globals and Properties  ;;  
;;;;;;;;;;;;;;  
-----  
;;The default agent in NetLogo is a turtle. We want to use robots!  
breed [robots robot]  
  
;;Let's load the bitmap extension to use a Mars planet background.  
extensions[ bitmap ]  
  
;;We need to keep track of how many rocks are left to gather, and if our robot is looking for a rock.  
globals [ numberOfRocks ]      ;;total number of rocks to gather on the grid  
  
;;We need each robot to know some information about itself.  
robots-own [searching?]        ;;robots need to know if they are in the searching state.  
  
;;Each patch should know what color it is to begin with.  
patches-own[baseColor]
```

- The gray code is **comment code**. Comments are useful for explaining what a program does. NetLogo ignores comment code when it runs the program. You can create comments in your NetLogo code by adding 2 semicolons (;;) to the beginning of your text. The black and colorful code is used to run the program. Notice that some code in the Globals and Properties and Setup sections is already written for you.

## 2.2 SETUP THE PROGRAM

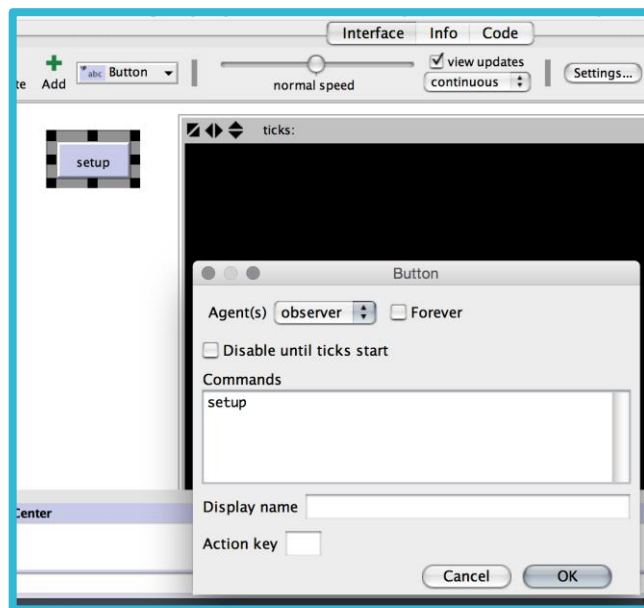
- Click the Interface tab.
- Create a new button by clicking the drop-down menu next to the Add button and selecting Button.



- The cursor changes to crosshairs. Click to the left of the black display panel. A button appears and a dialogue screen comes up. Type **setup** in the Commands box and click OK.

## BUTTONS IN NETLOGO

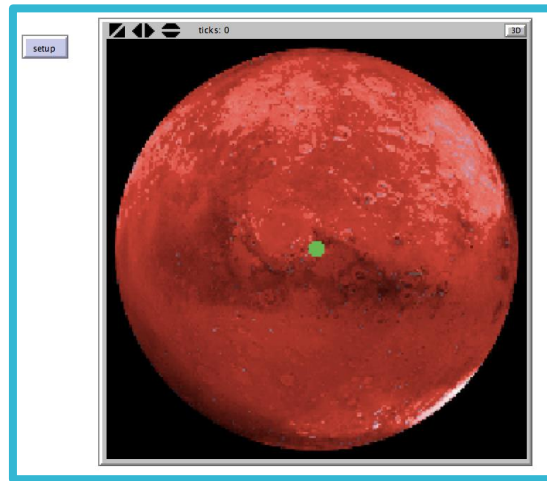
When you click a button that you made on the NetLogo interface, the code that corresponds to that button will run. We call that code a **procedure**.



- The **setup** procedure will run when the **setup** button is clicked.



- Click your new setup button now. An image of Mars should appear. The green circle in the center is your base.



- If the image does not appear, make sure that the .jpg image is in the same folder with the NetLogo file that you are working on.

### 2.3 WRITE SOME NETLOGO CODE TO CREATE ROBOTS!

Let's fill in part of the setup procedure.

- We need to set values for some **variables**. We created the **global variable** numberOfRocks and the **robots-own variable** searching? in the Globals and Properties section. Before we can use them, we need to give them a value.

Fill in sections 1), 2), and 3) in the setup procedure using the code from the following. The comments are there to guide you. Read the comments to understand what your code is doing.

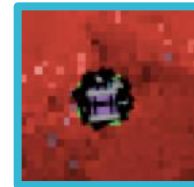
**Be careful to type the code exactly as it is written here, including dashes, quotes, and brackets, or it may not work.**

```
;;1) Set the global numberOfRocks to 0 to start
set numberOfRocks 0

;;2) Here we create some robots. NetLogo's default shape for an agent is a turtle,
;;so change their shape from a turtle to a robot.
;;Set its size to 8, so you can see it more clearly.
create-robots 6 [
  set shape "robot"
  set size 8
]

;;3) Set robots to start in search mode
ask robots [set searching? true]
```

- Go back to the interface and click your setup button. You should see 6 robots all grouped up at the base!
- **Save your program.** Do this often or you may lose work!

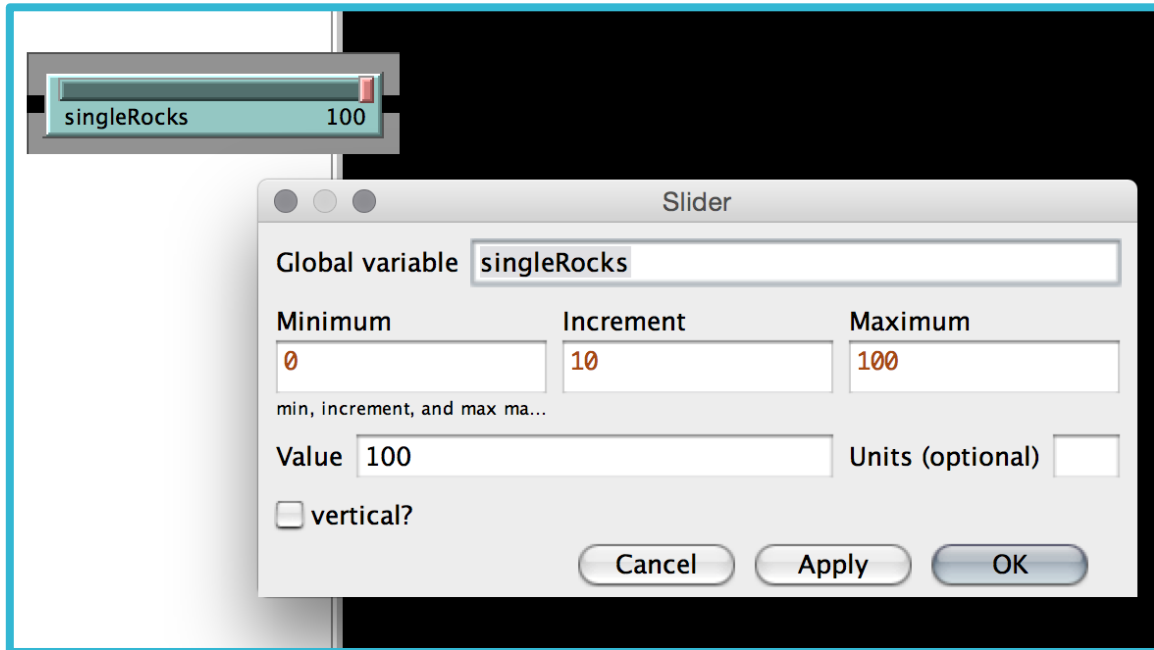


## 2.4 FINISH THE SETUP PROCEDURE

For the final touch, we'll add some rocks for the Swarmie robots to pick up.

- We'll add both **single rocks** and **clusters of rocks**.
- We'll determine how many single rocks and clusters of rocks to add by creating **sliders on the interface**.
- We will make the rocks by changing the color of some of the patches in the NetLogo world.

Go to the Interface tab and select **slider** from the same dropdown menu where you made a button. Fill in the values as in the picture below.



## SLIDERS IN NETLOGO

You can use the Global variable that you created in your slider in the Code tab.

- Is your slider overlapping the Interface? Do you want to adjust the size or position? Right-click it and choose Select. Now you can pull at the sides to resize it or click and drag to move it around.



Create a second slider for the number of clusters of rocks. Use the values in the table below.

Global variable	Minimum	Increment	Maximum	Value
clusterRocks	0	5	50	25

Now that we have our sliders, we can use their values to write code that places as many single rocks and clusters of rocks as we want. We'll also update the numberOfRocks variable so we know how many rocks we have.

As you did in [2.3](#), fill in sections of the code using the following pictures as a guide. Fill in sections 4) and 5) to finish the setup procedure code. Be sure to read the comments to understand what the code is doing.

- Click the **Check** button on the Code tab *after completing a section* to make sure there are no errors. If there is an error, **fix it** before adding more code.
- Click the setup button after completing 4). Do that again after completing 5).
- Don't forget to save your program!

```

;;4) Let's set some random patches to the color yellow to represent rocks.
;; We'll get the number of random patches from the slider singleRocks.
;; pcolor means patch color.
;; We don't want to make rocks that are off the planet, so we check if the random
;; patch selected is black. We won't put a rock there if it is. We also don't want to add
;; a rock right on top of another rock, so we'll check that too.
;; != means 'does not equal'.
;; Let's update our global variable numberOfRocks to keep track of
;; how many rocks we have. We do this after adding the rocks.
let targetPatches singleRocks
  while [targetPatches > 0][
    ask one-of patches[
      if pcolor != black and pcolor != yellow[
        set pcolor yellow
        set targetPatches targetPatches - 1
      ]
    ]
  ]
set numberOfRocks (numberOfRocks + singleRocks)

```

Notice that in 5) we **set** targetPatches instead of **let** targetPatches.

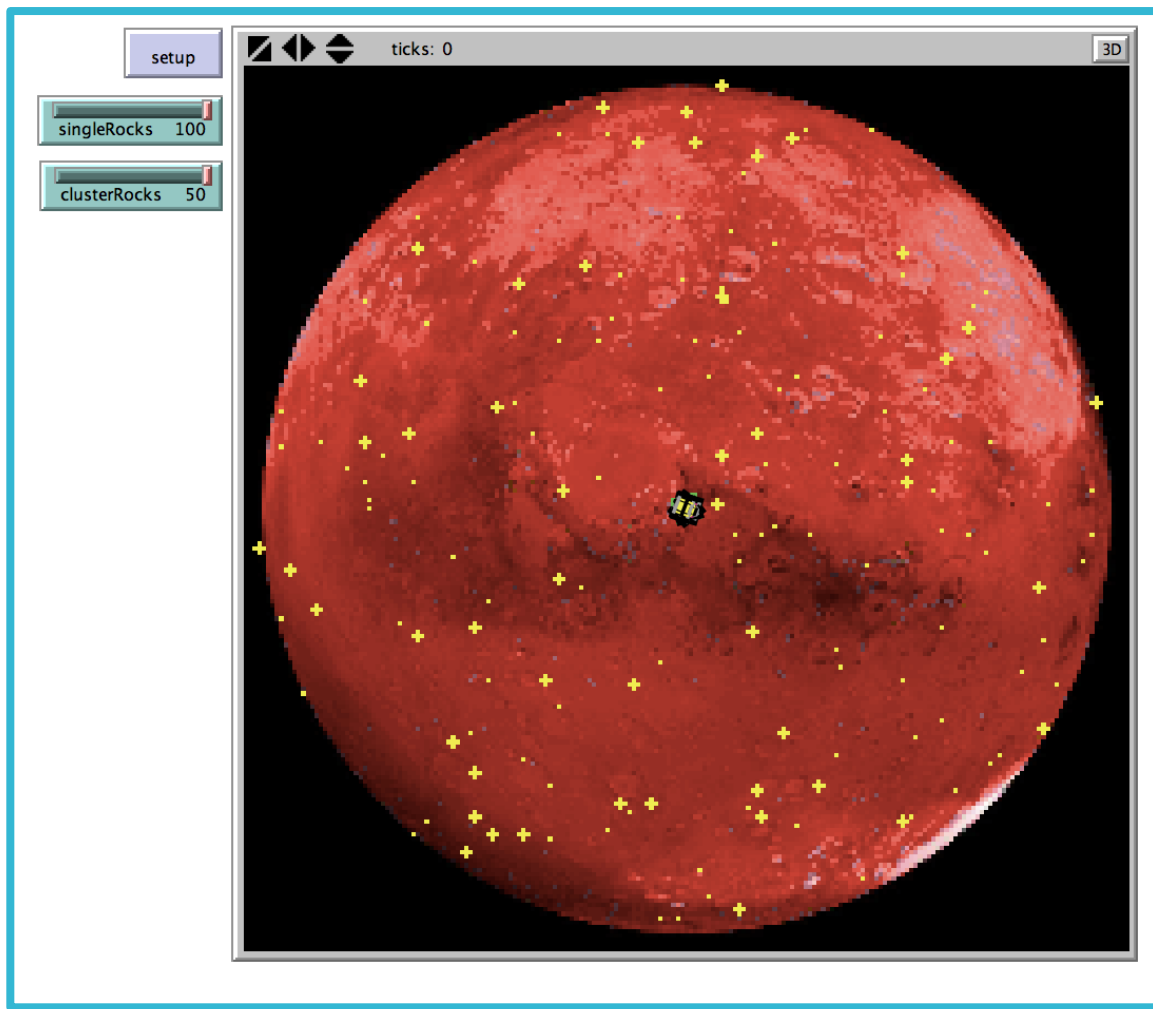
When we **let** a variable, we create it. When we **set** a variable, we reuse the name but change its value.

```

;;5) Now, let's make some clusters of rocks for the robot to pick up.
;; We'll get the number of clusters from the slider clusterRocks.
;; We don't want to make rocks in an illegal place, so we check for black and yellow patches like before.
;; This time we must also ask the patches to the North, South, East, and West of our target patch to become
;; rocks also and add but only if they are not off-world or already rocks.
;; Update numberOfRocks again.
set targetPatches (clusterRocks * 5)
  while [targetPatches > 0][
    ask one-of patches[
      if pcolor != black and pcolor != yellow
      and [pcolor] of neighbors4 != black and [pcolor] of neighbors4 != yellow[
        set pcolor yellow
        ask neighbors4[ set pcolor yellow ]
        set targetPatches targetPatches - 5
      ]
    ]
  ]
set numberOfRocks (numberOfRocks + (clusterRocks * 5))

```

GREAT JOB! You completed Section 2.



## END OF SECTION CHALLENGE:

Create a slider that controls the number of robots that are made when you create-robots.

## 3 GET THOSE ROBOTS RUNNING!

### 3.1 OVERVIEW

Now we have our Swarmie robots and rocks...but how do we make the robots move around? How do we make them pick up the rocks? How do they know where the base is to bring the rocks back? And how do robots know when there aren't any rocks left and they should stop searching?



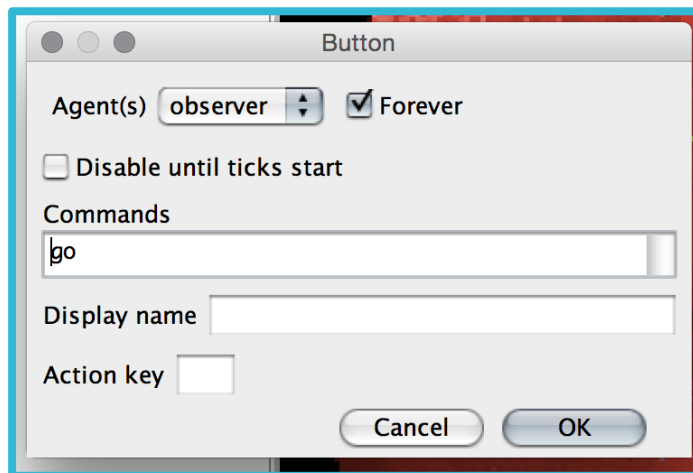
*Photo courtesy of NASA*

Scroll down below the setup procedure that you have just written. You will find a **go** procedure. In this section, we will complete this procedure.

- We'll add a button to the interface to run the procedure.
- The robots will move around in a biologically-inspired way.
- We'll make the robot recognize when it finds a rock and have the robot pick up the rock and drop it off at the base.
- The robots will stop searching for rocks when no more rocks remain.




### 3.2 THE WIGGLE WALK/CORRELATED RANDOM WALK

Make a **go** button on the interface now. **This time, make sure the forever box is checked.** (See the picture below.) When the forever box is checked, **the program will repeat the code in that procedure until you click the button again to stop it.**



Let's start with the basics. The robots need to move around. How should we have them move? Well, since the Swarmies are biologically-inspired, we should look to nature for inspiration.

Animals in nature move many different ways. Look at the table below to learn more about animal movement.

name	behavior	what is it used for	what does it look like
<b>random walk</b>	steps have no relationship to the previous step	tight search of an area	a squiggle 
<b>wiggle walk (correlated random walk)</b>	steps are within a given angle from the previous step	moderate search of an area	a slightly organized squiggle 
<b>ballistic motion</b>	straight line	fast travel to another area	a straight line 

Let's use a wiggle walk for the Swarmies.

In the **go** procedure, fill in section 1) (near the bottom of the procedure) like so:

```
;;1) make the robots move  
wiggle
```

By typing **wiggle** in the **go** procedure, we tell the **go** procedure to use the **wiggle** procedure. Scroll down in the code to see the **wiggle** procedure. The **wiggle** procedure controls the robots' movement. Note that the comments mention **maxAngle**. We need to define this variable.



We can create a slider to control the maximum angle in degrees a robot can turn from its previous step. Make a slider called maxAngle on the Interface using the values below.

Global variable	Minimum	Increment	Maximum	Value
maxAngle	0	5	90	45

Recall that we used the singleRocks and clusterRocks slider values in our code in the setup procedure. In the go procedure, we'll use the maxAngle slider value. We'll also add a basic command to make the robots turn around when they reach the edge of the world. We don't want them wandering off into space!

Fill in the **wiggle** procedure as in the picture below.

```

;-----
;::::::::::::::::::::::::::
;;   wiggle   ;;
;::::::::::::::::::::::::::

to wiggle

;; 1) turn right 0 - maxAngle degrees
right random maxAngle

;; 2) turn left 0 - maxAngle degrees
left random maxAngle

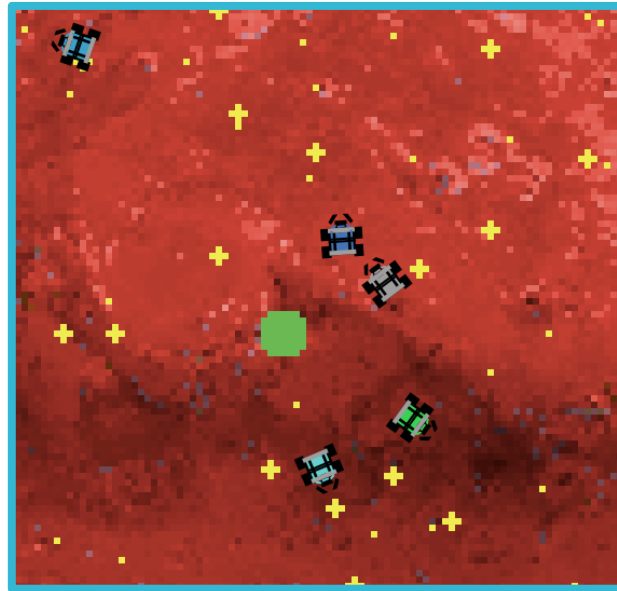
;; 3) turn around and face the origin if we hit the edge of the planet
;; (the patch color is black at the edge of the planet)
if pcolor = black [ facexy 0 0 ]

;; 4) go forward one patch
forward 1

end

```

- Hit the check button in the Code tab to make sure your code is correct. If not, go back and fix it before moving on.
- Save your program.
- Click setup, and then go. The robots move around!
- Experiment with changing the maxAngle slider and notice how it affects the robots' movement.



### 3.3 FINISH THE GO PROCEDURE WITH AN IFELSE STATEMENT

In the setup, we set each robot's mode to searching. We did that by setting the value of the **robots-own** (every robot has one) variable **searching?** to **true** in the setup procedure. If a robot is searching, it should look for rocks. Else, it found a rock and is no longer searching—it should return to base. Let's make that happen.

Scroll to the go procedure and complete it by adding this code to sections 2) and 3):

```
;;2) ask the robots if they are searching or not
ifelse searching?

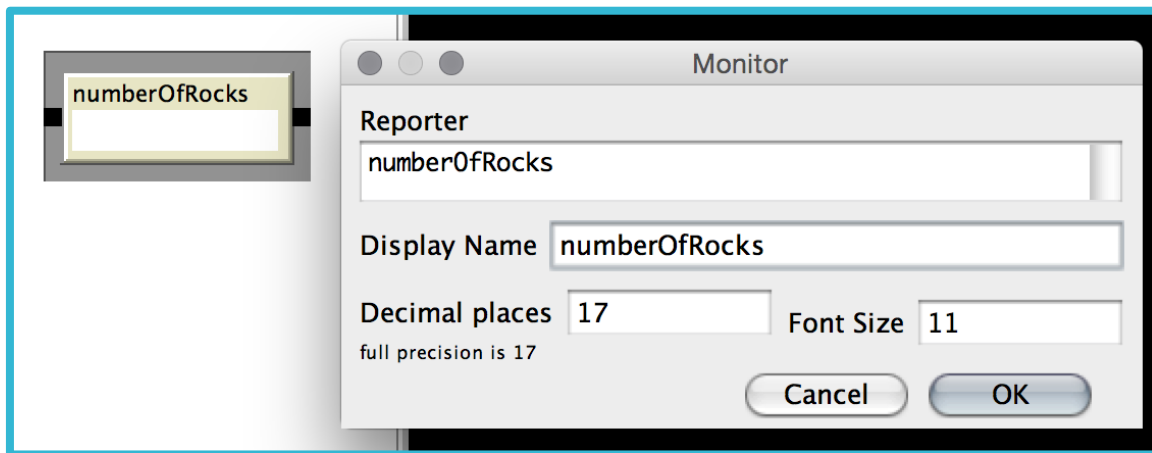
|

;;3) tell them what to do based on the answer
[look-for-rocks]
[return-to-base]
```

Use an **ifelse** statement to control the robots' behavior. We tell the go procedure in section 3) to use the **look-for-rocks** procedure **if** the robots-own searching? variable is set to true. **Else**, the variable is set to false (the robot is not searching), and the go procedure should use the **return-to-base** procedure. We'll write the code for look-for-rocks and return-to-base now.

### 3.4 LOOK-FOR-ROCKS

First, let's create a monitor on the Interface to keep track of how many rocks we have. We'll use the numberOfRocks variable. Use the drop-down menu and select monitor. Fill it out like this:



- Click the setup button and verify that your monitor shows the correct amount of rocks that you specified with your sliders. (Each cluster contains 5 rocks.)

Scroll down to the look-for-rocks procedure. If the robot is set to search mode, it will perform this procedure before **wiggle**. It will not perform the return-to-base procedure.

Fill in sections 1) and 2) in the look-for rocks procedure as in the picture below. Carefully read the comments in the code to understand what the code is doing.

```
,  
,,,,,,,,,,,,,,,,,,,,,  
;; look-for-rocks ;;  
,,,,,,,,,,,,,,,,,,,,,  
  
to look-for-rocks  
  
;;1) Ask the 8 patches around the robot if the patch color is yellow  
ask neighbors [  
    if pcolor = yellow [  
  
;;2) If it is, take one rock away, and set search mode to false.  
;;   Change the patch color to the original color where we removed the rock.  
;;   Have the robot ask itself to turn off searching and set its shape to  
;;   the one holding a rock.  
set pcolor baseColor  
set numberOfRocks (numberOfRocks - 1)  
ask myself [  
    set searching? false  
    set shape "robot with rock"  
]  
]  
  
end
```

### 3.5 RETURN-TO-BASE

Scroll down to the return-to-base procedure. If the robot is not set to search mode, it will perform this procedure before **wiggle**. It will not perform the look-for-rocks procedure.

We'll use another `ifelse` statement here. If the robot found the base, it will drop the rock and start searching again. Else, it should continue to head towards the base.

Fill in sections 1), 2), and 3) to complete the return-to-base procedure. Carefully read the code comments to understand what the code is doing.



```
-----  
;;  
;;  
;; return-to-base ;;  
;;  
;;  
  
to return-to-base  
  
;;1) If the patch color is green, we found the base.  
ifelse pcolor = green  
  
;;2) Change the robot's shape to the one without the rock,  
;; and start searching again.  
[  
  set shape "robot"  
  set searching? true  
]  
  
;;3) Else, we didn't find the base yet--face the base  
[facexy 0 0 ]  
  
end
```

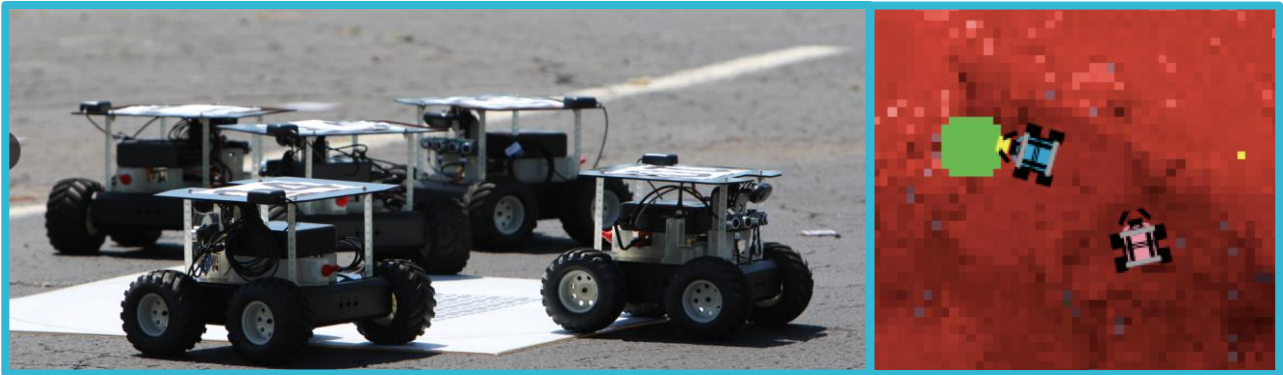
- Click setup and then go. Your robots should gather and drop off rocks until no more rocks remain.
- Try changing the wiggle angle and the number of clusterRocks and singleRocks.

GREAT JOB! You completed Section 3.

## END OF SECTION CHALLENGE:

Robots will automatically stop moving when no rocks remain (no more yellow patches) because of some code that is included at the end of the go procedure. Can you add some code inside there to make the robots go back to base then? HINT: **ask** the robots to 1) turn off their searching and **while** they are not at the base, they should return-to-base and move forward 1. Learn more about while statements in NetLogo here: <http://ccl.northwestern.edu/netlogo/docs/dict/while.html>

## 4 IF AN ANT CAN DO IT, SO CAN A ROBOT



### 4.1 SITE FIDELITY OVERVIEW

It's not just the Swarmies' wiggle walk that is biologically-inspired—the robots use other behaviors borrowed from nature too. In this final section, we will implement a behavior seen in some ants called *site fidelity*.



*Photo courtesy of Paul Stockton*



*Site fidelity* simply means that an ant remembers a location where it found food before and returns to that location. In our Swarmie NetLogo simulation, we'll go back to where we found a rock after dropping the rock off at the base. In this section, we will work with and add to the code we created in Sections 1 – 4 to implement site fidelity.

## 4.2 ANOTHER STATE OF MIND

Previously in this walkthrough, a robot was either searching or not searching. Now, we need to add an additional state to the robot: using site fidelity. The robot should know if it is doing this behavior, so we'll need to:

- add a **usingSiteFidelity?** variable to the **robots-own** section in the Globals and Properties section
- set the value of that variable to false initially in the setup procedure



*Photo courtesy of Paul Stockton*

But the robot needs to know one more thing—the coordinates where it found the rock before. If it can't remember where it found a rock, how would it know where to go back to? Call these coordinates `siteX` and `siteY`. We'll also need to:

- add these variables to `robots-own`, and add comments
- initialize both variables to 0 in the setup section.

Use the pictures to help you set up the site fidelity state for the robots. Be sure to include the comments in the pictures to document your code, or, better yet, add your own!

```
;;We need each robot to know some information about itself.
robots-own [
  searching? ;;robots need to know if they are in the searching state.
  usingSiteFidelity? ;;robots need to know if they are usingSiteFidelity or not.
  siteX ;;x-coordinate to use for site fidelity
  siteY ;;y-coordinate to use for site fidelity
]
```

```
;;3) Set robots to start in search mode
;; also set robots to start without site fidelity,
;; and set the site fidelity coordinates to the origin.
ask robots [
  set searching? true
  set usingSiteFidelity? false
  set siteX 0
  set siteY 0
]
```

### 4.3 ADDING THE SITE FIDELITY STATE TO GO

Now that the robots can remember where they found a rock before, let's change their behavior so that they return to that location once.

Our go procedure functions as a controller for the Swarmies' behavior based on their state. Now that we've added the site fidelity state, we need to check if it's active. Scroll to the go procedure and modify the code to include the new state:

```

;-----
;,,,,,,,,,,,,,,,,,,,,,
;;      Go      ;;
;,,,,,,,,,,,,,,,,,,,,,
to go

;;run the program until all rocks are collected
if (numberOfRocks > 0)
[
  ask robots
  [

    ;;2) ask the robots if they are using site fidelity and searching or not
    if searching? and not usingSiteFidelity? [look-for-rocks]
    if searching? and usingSiteFidelity? [do-site-fidelity]
    if not searching? [return-to-base]

    ;;1) make the robots move
    wiggle

  ]

]

;;advance the clock
tick

if not any? patches with [pcolor = yellow][
  set numberOfRocks 0
  stop
]

end

```

Great! But there are 2 major things we still need to tackle to get site fidelity working.

- 1) The `usingSiteFidelity?` variable isn't ever set to true.
- 2) We told the go procedure to use the `do-site-fidelity` procedure, but we haven't written the code for `do-site-fidelity`.

#### 4.4 SETTING THE SITE FIDELITY STATE TO TRUE

When a robot finds a rock, we want to turn site fidelity on. Recall that rock detection happens in the **look-for-rocks** procedure. So let's add a little code to the look-for-rocks procedure so that when the robot finds a rock, it also turns on the site fidelity state! Scroll to the look-for-rocks procedure and modify the code:

```

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

to look-for-rocks

;;1) Ask the 8 patches around the robot if the patch color is yellow
ask neighbors[
  if pcolor = yellow[
;;2) If it is, take one rock away, and set search mode to false.
;;   Change the patch color to the original color where we removed the rock.
;;   Have the robot ask itself to turn off searching and set its shape to
;;   the one holding a rock.
    set pcolor baseColor
    set numberOfRocks (numberOfRocks - 1)
    ask myself [
      set searching? false
      set shape "robot with rock"
      if not usingSiteFidelity? [ ;Turn on site fidelity if it's not on already.
        set usingSiteFidelity? true
        set label "sf" ;Put a label on robots that are using site fidelity
        set siteX pxcor ;Store the x and y coordinates of where we found
        set siteY pycor ;this rock.
      ]
    ]
  ]
]

end
```



Notice that we used `pxcor` and `pycor`. These variables describe the x and y coordinates of the patch the robot is currently standing on. Every patch has these variables. They are patches-own variables that you do not need to initialize in setup.

#### 4.5 SETTING THE SITE FIDELITY STATE TO FALSE

Now we finish up by writing the `do-site-fidelity` procedure, which will handle the robot's behavior when in the site fidelity state. We'll also need to turn the site fidelity state off when the robot arrives back at where it found the rock before, or it will get stuck in an infinite loop! This procedure also turns off the site fidelity label.

You will write the `do-site-fidelity` procedure on your own. Add it at the bottom of your file. Use the code in the picture below, but this time, **add your own comments** to describe what the code is doing.

```
;-----
;;;;;;;;;;;;;;;;;;
;; do-site-fidelity ;;
;;;;;;;;;;;;;;;;;;

to do-site-fidelity

  ifelse(pxcor = siteX and pycor = siteY)
  [
    set usingSiteFidelity? false
    set label ""
  ]
  [facexy siteX siteY]

end
```

## HINTS FOR COMMENTS:

- `pxcor` and `pycor` are explained at the end of 4.4.
- You used an `ifelse` statement and `facexy` in the `return-to-base` procedure.

GREAT JOB! You completed SWARMATHON 1.



## BUG REPORT? FEATURE REQUEST?

Email [sherbet@unm.edu](mailto:sherbet@unm.edu) with the subject SW1 Report

NEXT UP

SWARMATHON 2: Advanced Bio-Inspired Search