

Collecting Rocks on Mars 3

In Collecting Rocks on Mars 2, you created a simulation of multiple Swarmie robots working cooperatively to collect rocks on Mars.

In this exercise, you will go one step further and develop biologically-based methods for robots to maximize both their individual and group efficiency in gathering rocks. The robots will now have the ability to make a return trip to a location where they found a rock, and lay a trail from this location so that other robots can find the location also.

Why Biology?

We can learn a lot about how to search effectively by looking at nature. Ants are arguably the best species at working cooperatively to gather resources. Here we implement two of the behaviors that ants use when searching for resources: **site fidelity** and **pheromone**.

Ants can remember the last location they found a resource. **Site fidelity** is the behavior where an ant returns to that location. Sometimes, an ant will also lay a **pheromone** trail when it finds a location with a lot of resources. The trail is made of chemicals and is invisible. Other ants can pick it up at the nest and follow it.

Follow the instructions step-by-step. **Don't skip ahead!**

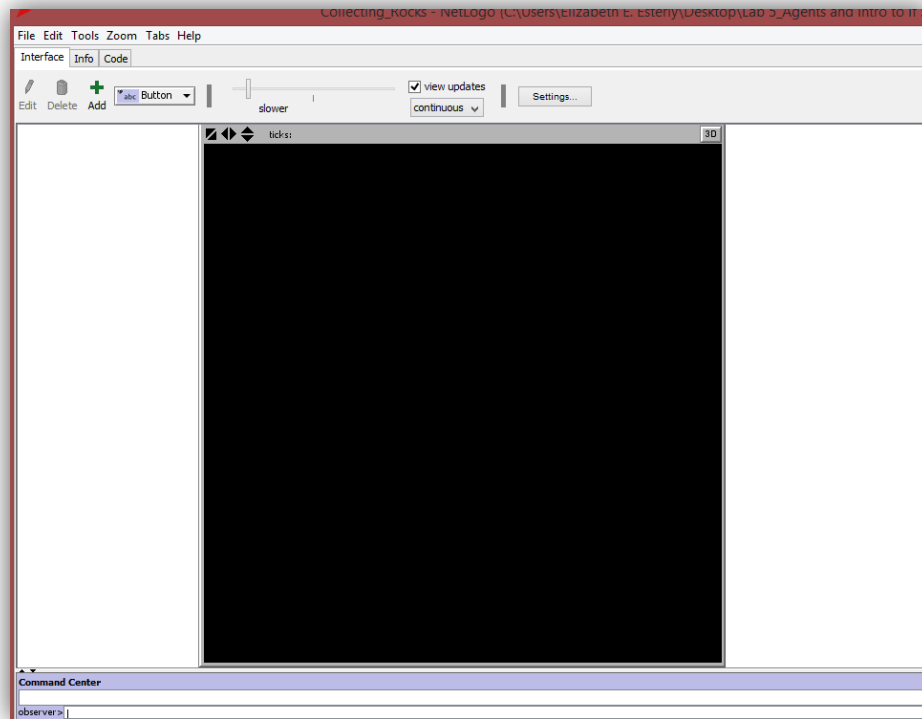
Some parts of the program are already done for you. While you are working, **don't change the code that's already been written, or the program may not run. Once you've finished the program, you may experiment by changing the code.**

1) Getting Started

1.1 Create a new folder on the Desktop and name it something you'll remember.

1.2 Download the zip file. Unzip it to the folder you just created. Make sure the student .nlogo file and the mars.jpg file are in the same directory.

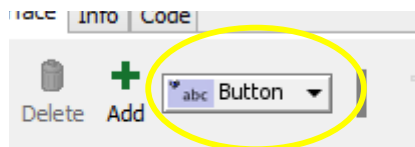
1.3 Double-click the student .nlogo file and it will launch in NetLogo.



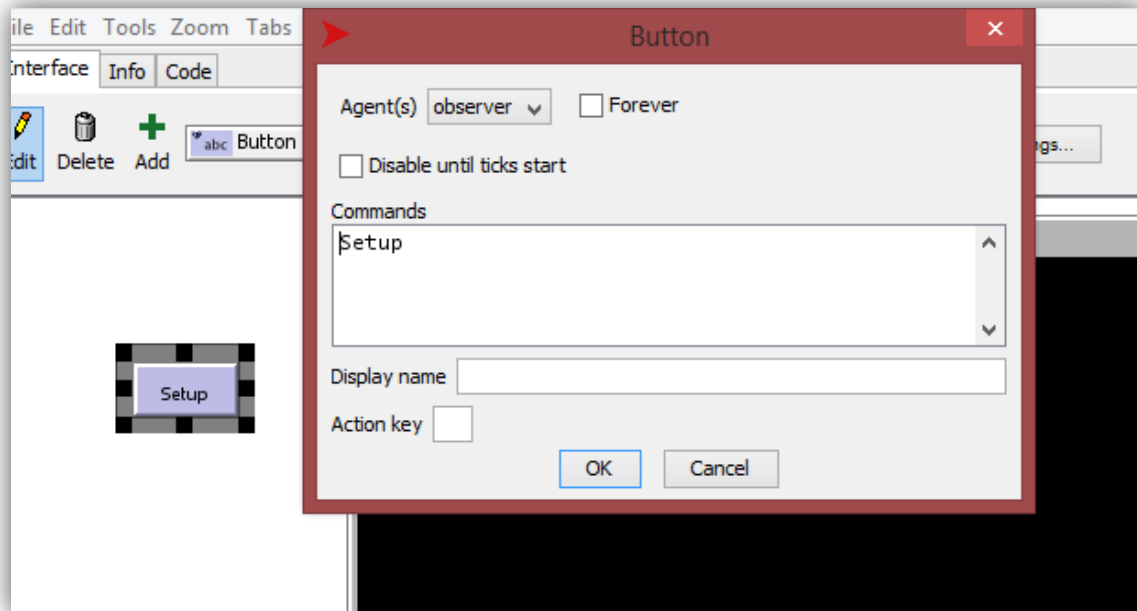
2) Setup the program

If you completed Mars Robot 1 and 2, this section will mostly be review. However, do not skip the section, as there are some important changes from Mars Robot 1 and 2.

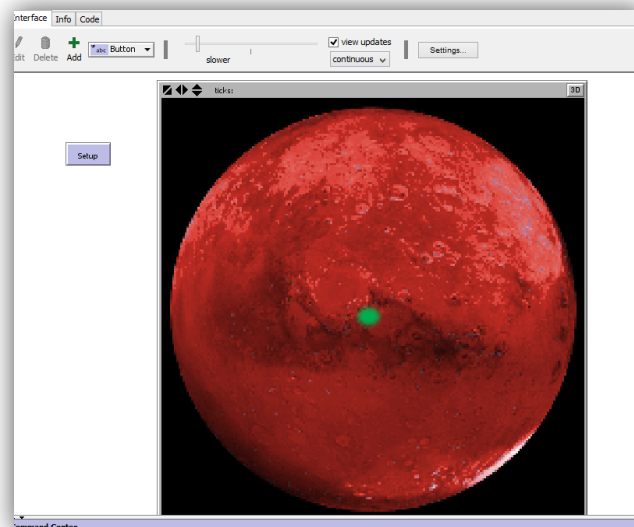
2.1 Create a new button by clicking the button next to the Add button and selecting Button from the drop-down menu (first option).



2.2 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.

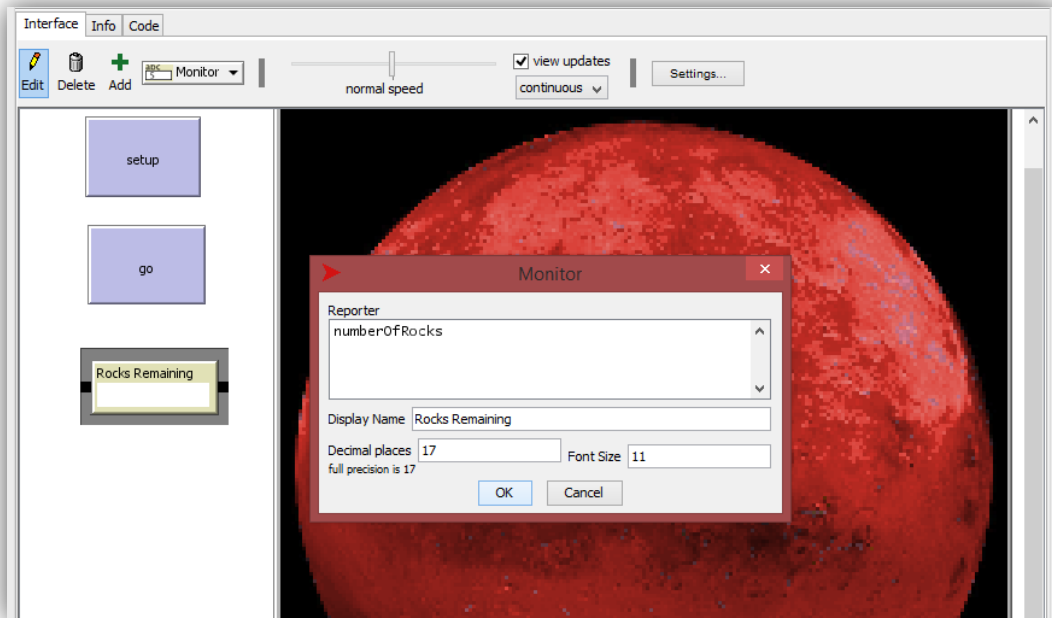


2.3 Click your new Setup button. You should see an image of Mars appear in the window after about 20 seconds. As with Mars Robot 1, the green circle in the middle is your base.



2.4 Use the same procedure to create a Go button directly below the Setup button. Type Go in the Commands window instead of Setup to do this.

2.5 Make a monitor to keep track of how many rocks remain for the robots to pick up. If you created a monitor in Mars Robot 1 or 2, this will be review. Select Monitor from the same drop-down menu. Click below the Go button to place the monitor. Type `numberOfRocks` in the Reporter box. You may type whatever you wish in the Display Name box. Try to choose a descriptive name. We chose Rocks Remaining.



2.6 Click on the Code tab, scroll to the Setup section, and enter the following code:

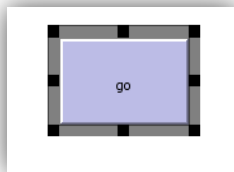
```
-----  
;;1) Set numberOfRocks to the value of the sliders.  
set numberOfRocks 9 * clusters + singles
```

Each cluster contains 9 rocks.

Because we did this, we don't need to add a rock to `numberOfRocks` every time we place one as in Mars Robot 1 and 2. This is why having a slider that changes the variables is convenient. We don't need to know the exact amount of clusters and singles to be able to place the correct amount every time.

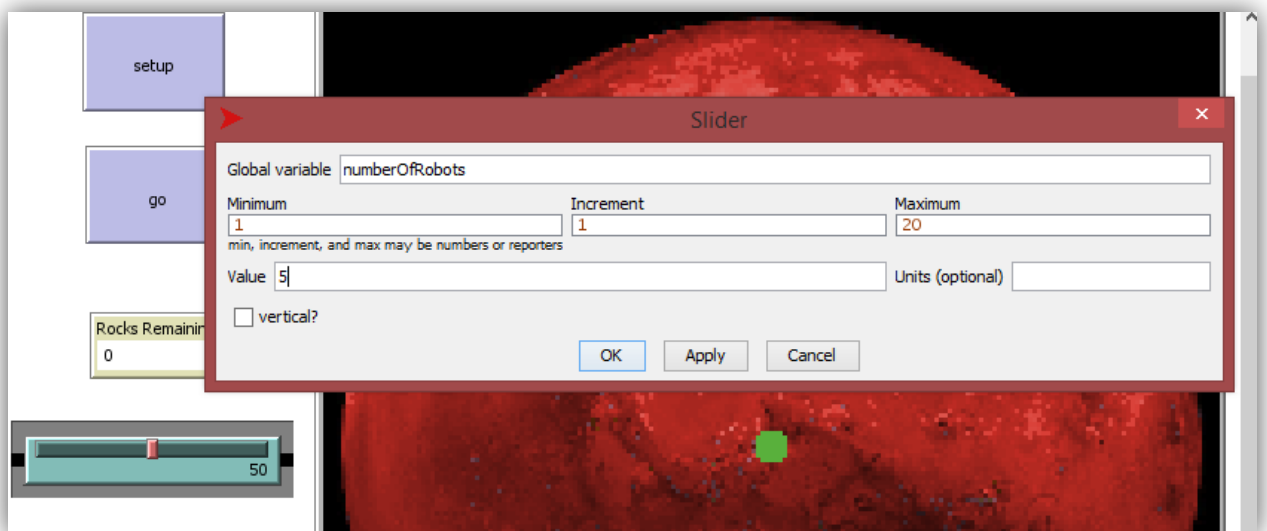
2.7 As in Mars Robot 2, we will also use slider bars so that you may experiment with the number of robots, the number of clusters of rocks, and the number of single rocks. We'll also add a new slider bar for `max pheromone`, which determines how long a pheromone trail is active before it evaporates. Let's set up these slider bars now.

We'll start by setting up the numberOfRobots bar. Select Slider from the same drop-down menu. Click below the monitor you just created to place the slider bar.



HINT: If you need to resize or rearrange your buttons, monitor, or slider bars, you can right-click the object you would like to modify and choose Select. Resize bounds appear and the object becomes scalable and moveable. Click the white background to deselect an object.

In the Global Variable box, enter numberOfRobots. Set the Minimum to 1 (there must be at least one robot) and the Maximum to 20. Leave the increment as 1. Set the value to 5. You can change the value by moving the slider bar after creating the slider. Click OK.

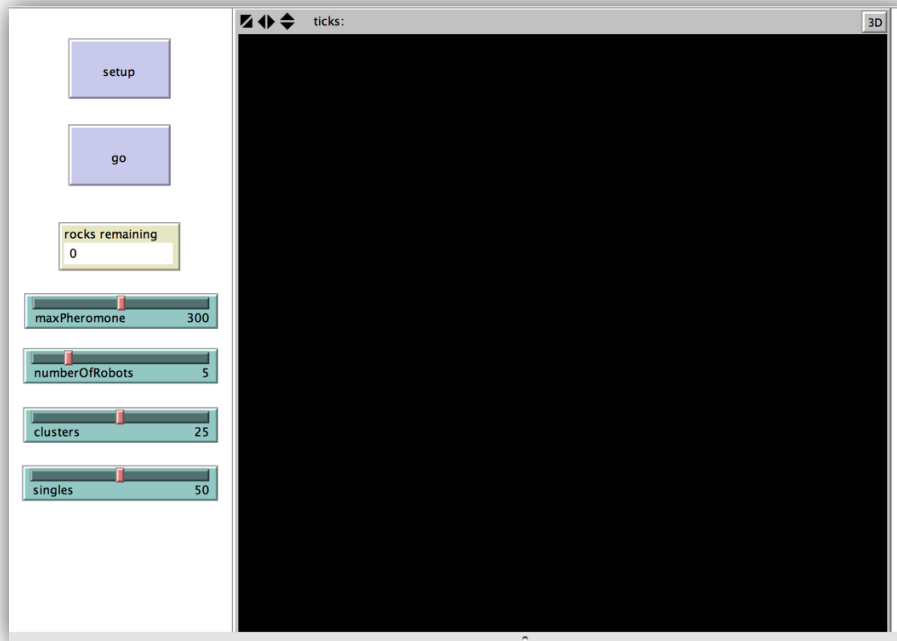


2.8 Save your program.

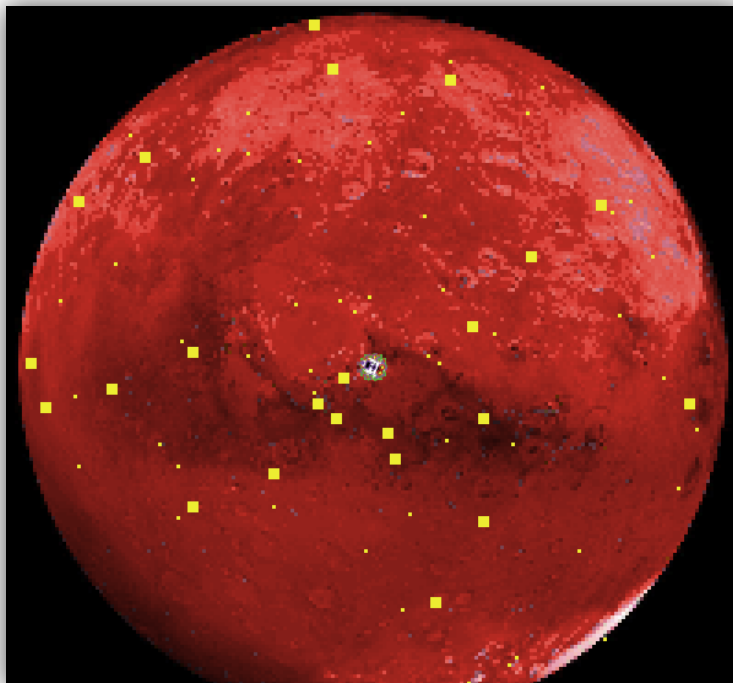
2.9 Create 3 additional sliders with the values shown in this table:

Global variable	Minimum	Increment	Maximum	Value
maxPheromone	0	100	600	300
singles	0	5	100	50
clusters	0	5	50	25

Your interface should now look like this:



2.10 Click Setup. You should see the following image. 50 single rocks appear along with the 25 clusters, the Rocks Remaining monitor displays 275, and 5 robots appear clustered at the base. If you do not see an image similar to this one or the monitor is not working correctly, please check your code



and try again. Remember that the rocks are placed in a random location every time, so your rock distribution may look different from the one pictured here.

2.11 Save your work.

3) Biologically-inspired machines

3.1 In Mars Robot 2, our robots knew if they were searching, returning, recruiting, or recruited. In this program, we also have multiple modes and multiple robots, but we are communicating information about resources in a way that is inspired by nature. We'll be using **site fidelity** and **pheromone**. Let's give each robot its own set of modes and also some coordinates to remember. Go to the Globals section and enter the following code:

```
;;1) Robots each have their own list of modes: searching?, returning?, usingSiteFidelity?,  
;; and usingPheromone?. They also remember the x and y coordinates for where they had previously found a rock  
;; if they are usingSiteFidelity. These coordinates are called siteFidelityX and siteFidelityY.  
turtles-own  
[  
  searching?  
  returning?  
  usingSiteFidelity?  
  siteFidelityX  
  siteFidelityY  
  usingPheromone?  
]
```

We'll use these 4 modes to build our Go procedure. We told the robots that they had these modes and variables, but we didn't give them a value yet. We need to do that before we can use them. Go to the Setup section and enter the following code:

```
;;1)Each robot has its own memory of what mode it is in.  
;;Set them to start in search mode. Set search mode to true, and set all other modes to false.  
ask turtles  
[  
  set searching? true  
  set returning? false  
  set usingSiteFidelity? false  
  set usingPheromone? false  
]
```

3.2 Let's expand further by giving the patches something to remember as well. Patches need to know how long pheromone is on them. Pheromone evaporates after an amount of time determined by your choice in the maxPheromone slider. Enter this code back in the Globals section.

```
;;2) Patches have a counter for how long pheromone has been placed on them called pheromoneCounter.  
patches-own  
[  
  pheromoneCounter  
]
```

4) Let's Go!

4.1 The Wiggle procedure is identical to Mars Robot 1, so it is already written for you here. Make sure your simulation is working correctly by clicking Setup and then Go. The robots should move, but not pick up any rocks.

4.2 Go to Tools → Halt to stop the program.

4.3 It's not just robots that must remember their mode in Mars Robot 3. We gave patches their own variable, `pheromoneCounter`, in the last section. If a patch has pheromone on it, we need to decrease the amount every time the clock ticks. If the pheromone lasted forever, it would not be effective.

Enter the following code in the Go section to make the patches with pheromone on them decrease the pheromone level by one every time the clock ticks.

```
;;1) When a robot finds multiple rocks at a site, it will lay a pheromone trail
;;so other robots can find that location. In our simulation, the pheromone trail is
;;colored cyan. The trail doesn't last forever, though, so on each tick, we'll decrease
;;the amount of pheromone on the patch by one until the trail evaporates.

;;ask the patches with cyan color
ask patches with [ pcolor = cyan ]
[
  ;;decrease pheromoneCounter by 1
  set pheromoneCounter pheromoneCounter - 1
  ;;if pheromoneCounter is 0, turn the patch red.
  if pheromoneCounter = 0[ set pcolor red ]
]
```

4.4 Now we'll tell the robot what to do based on what mode it is currently in. In Mars Robot 1 and 2, robots could only be in one mode at a time. In this activity, a robot can be using `SiteFidelity` and `returning` at the same time. You'll see why a little bit later on. Scroll down to the Go procedure and enter the following code in your program. You may remember the `[look-for-rocks]` and `[return-to-base]` procedures from Mars Robot 1 and 2.


```

;;1)if a robot is searching? it should look-for-rocks
if searching?
[look-for-rocks]

;;2)if a robot is returning? it should return-to-base
if returning?
[return-to-base]

;;3) if a robot is usingSiteFidelity? and not returning? it should go-back-to-rocks
if usingSiteFidelity? and not returning?
[go-back-to-rocks]

;;4) if a robot is usingPheromone? it should follow-the-trail
if usingPheromone?
[follow-the-trail]

```

4.5 Save your program.

5) Controlling the Robots: Looking For Rocks

5.1 The [look-for-rocks] procedure has some things in common with Mars Robot 2. If a robot finds a rock, it still must look around and count other rocks in the area. If there are at least 2 more, instead of calling other robots over as in Mars Robot 2, it will set itself to usingSiteFidelity? mode and remember the x and y coordinates of this location.

On the next step, it will begin to return-to-base, and will go-back-to-rocks after it drops off the current rock it is holding at the base. The robot will also leave a pheromone trail back to the base from this location. In this way, a robot who is at the base can pick up the pheromone trail and follow it.

If there are not at least 2 more rocks, the robot will set itself to returning? mode to return-to-base and drop off the rock.

Thus a robot can be usingSiteFidelity? and returning? at the same time.

Read the comments in this section carefully so that you understand what the code is doing.

Scroll down to the look-for-rocks procedure and enter the following code into your program, then save your program:

```


;;1)We can pick up any rock in an adjacent patch.
if[pcolor] of one-of neighbors = yellow
[
  ;;2)then pick up the rock (reduce the numberOfRocks by 1),
  ;;change the agent model to the robot holding the rock, and reset the patch to red
  set numberOfRocks (numberOfRocks - 1)
  set shape "robot with rock"
  ask one-of neighbors with [pcolor = yellow]
  [set pcolor red]

  ;;3)Turn search mode off, and return mode on.
  set searching? false
  set returning? true

  ;;Robots use site fidelity if there's 2 or more rocks remaining. This is efficient.
  ;;It's usually quicker just to go back here than to go search for another rock.
  ;;4)If the robot is not using site fidelity
  if not usingSiteFidelity?
  [
    ;;5) Here you'll combine an if statement, count, and in-radius 2 to check if there are
    ;;2 or more patches in-radius 2 that have a pcolor of yellow.
    if count neighbors in-radius 2 with [ pcolor = yellow ] > 1
    [
      ;;6) Then set usingSiteFidelity to true, and record the x and y coordinates of the patch
      ;;where the robot is at now. We do this so that the robot knows where to return to.
      set usingSiteFidelity? true
      set siteFidelityX pxcor
      set siteFidelityY pycor
    ];;end if
  ];;end if
];end if

```

5.3 Finally, let's add 2 lines of code to the return-to-base method that will allow us to test site fidelity in section 6.



```

;;1)Turn off returning mode.
set returning? false

;;1)If the robot is not already using site fidelity, it can check if there are any
;;pheromone trails to follow.

;;2)Use an if statement, count and in-radius again. This time, the radius will be 10. C
;;patches with the color cyan is greater than zero. The cyan colored patch indicates a


;;set usingPheromone to true.

;;3) We want to head towards the closest trail. Combine face, one-of neighbors, the s
;;with the correct pcolor, and the statement "with-min[distance-myself]" to make the
;;trail.

;;end if
;;end if

;;2) If we're not usingSiteFidelity and we're not usingPheromone, set searching to true.
if not usingSiteFidelity? and not usingPheromone?[set searching? true]

```



6) Controlling the Robots: Site Fidelity

As programs get more complicated, sometimes it is necessary to test 2 things at once. The robot must drop off the rock first before we can ensure it is correctly returning to the coordinates stored in `siteFidelityX` and `siteFidelityY`.

6.1 We'll need to do a little work in the `go-back-to-rocks` procedure before being able to test `site fidelity`. Some of the code in the `return-to-base` procedure that you wrote in Mars Robot 1 and 2 has already been filled in. We'll return to that procedure later.

Scroll down to the `go-back-to-rocks` procedure and enter the following code into your program:

```
to go-back-to-rocks

;;reset label
set label "site fidelity"

;;This section is similar to move-to-friend from Mars Robot 2.
;;1) Use an ifelse statement. Determine if we've arrived back at where we found rocks before by
;;checking if siteFidelityX and siteFidelityY are the same as thecurrent location.
ifelse siteFidelityX = pxcor and siteFidelityY = pycor
[
  ;;2)If so, set searching to true. Then set usingSiteFidelity to false. We don't need it anymore.
  set searching? true
  set usingSiteFidelity? false
]
;;3) else we're not at the destination. Face the destination.
[facexy siteFidelityX siteFidelityY]

end
```

6.2 Save your program.

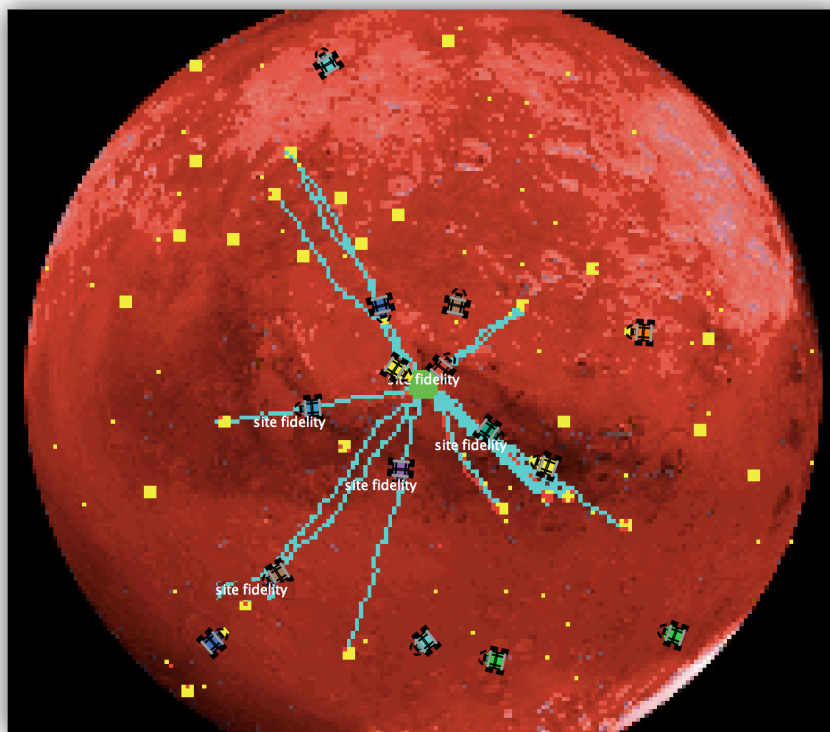


6.3 The robot in the picture above left a trail to the base. Let's add the code to make that happen in your simulation.

Go to `return-to-base` and enter the following code near the bottom of the method. Don't worry about the rest of the method at this time. We'll fill it in next when we handle pheromone.

```
;;1) If we're usingSiteFidelity or usingPheromone,
if usingSiteFidelity? or usingPheromone?
[
  ;;Ask the patch-here to
  ask patch-here
  [
    ;;2) Set its pheromoneCounter to the value we chose in the slider maxPheromone
    ;;to mark this patch as part of the trail. If the pcolor is not yellow, set the pcolor to cyan.
    ;;We have to check for a yellow patch because a yellow patch is a rock.
    ;;We don't want to knock out a rock.
    set pheromoneCounter maxPheromone
    if pcolor != yellow [set pcolor cyan]
  ]
]
```

6.5 Save your program.



6.6 Set the `numberOfRobots` to 15, the number of `singles` to 60, and the number of `clusters` to 30. Leave the `maxPheromone` at 300. Click `Setup` and `Go` to run your program. The robots should be using `site fidelity` a lot! Notice that when the trail evaporates, it leaves a red path. Let the simulation run for a while. What kind of patterns are left by the evaporated trails? Why do you think they're this shape?

7) Controlling the Robots: Pheromone

A robot that is not using **site fidelity** can follow the trail left by another robot. The trail can be picked up from the base. In nature, ants pick up trails from their nest.

7.1 Let's set up **pheromone** by finishing the return-to-base method, as we mentioned in 6.4.

Enter the following code in the return-to-base method:

```
;;1)If the robot is not already using site fidelity, it can check if there are any
;;pheromone trials to follow.
if not usingSiteFidelity?
[
  ;;2)Use an if statement, count and in-radius again. This time, the radius will be 10. Check if the number of
  ;;patches with the color cyan is greater than zero. The cyan colored patch indicates a trail.
  if count neighbors in-radius 10 with [ pcolor = cyan ] > 0
  [
    ;;3) set the robot's label to "pheromone"
    set label"pheromone"

    ;;4) set usingPheromone to true.
    set usingPheromone? true

    ;;5) We want to head towards the closest trail. Combine face, one-of neighbors, the same radius as in 1),
    ;;with the correct pcolor, and the statement "with-min[distance-myself]" to make the robot face the closest
    ;;trail.
    face one-of neighbors in-radius 10 with [pcolor = cyan] with-min[distance myself]
  ]
]
```

Lastly, we need to set up a way that the robot will continue following the trail on each tick. This seems easy, right? But what if the trail evaporates while the robot is following it? Then the robot will just run right back to the base and precious time was wasted.

It's good to think about these *edge cases*, or situations that don't happen often, but can break your program or make it inefficient. Try to think about what edge cases might occur every time you write a program.

7.2 Enter the following code into your program. **Be sure to read the explanation for each section carefully. This section's code is the most complicated so far.**

Notice that you can chain **with** statements. You can combine them like puzzle pieces.

- **with**
- **with-min**
- **with-max**

Also notice the **of** keyword.

- **of target**
- **of self**

There are many, many more combinations you can make.

```

to follow-the-trail
  set label "pheromone"

  ;;1) The robot located a trail when it was at the base. Now it have to follow it.
  ;;But the trail evaporates, so the robot must make sure the trail still exists.
  ;;Use the same statement as in return-to-base to make sure the trail is still there.
  ;;But this time, make it an ifelse statement.
  ifelse count neighbors in-radius 10 with [pcolor = cyan]> 0[

    ;2) The robot chooses a target patch that is on the trail. We have to choose a patch that's in-radius 10,
    ;;that has a pcolor of cyan, with-max distancexy from the origin, and with-min distancexy from myself.
    let target one-of neighbors in-radius 10 with [pcolor = cyan] with-max[distancexy 0 0] with-min[distance myself]

    ;;3) If the trail is already evaporating, the end of the trail might be closer to the origin than
    ;;the robot is. If this is the case, the robot shouldn't follow, as it will go right back to the base.
    ;;Use an ifelse statement. Check that the distancexy of the robot's target position from the origin
    ;; is greater than the distancexy from the origin of self.
    ifelse [distancexy 0 0] of target > [distancexy 0 0] of self
    ;;4)If so, face the target.
    [face target]
    ;;else, the robot should not follow the trail anymore.
    ;;Set usingPheromone to false and searching to true.
    [
      set usingPheromone? false
      set searching? true
    ]
  ];;end if
  ;;begin else
  ;;5) There wasn't a trail within 10 patches of our location. It must have evaporated.
  [
    ;;set usingPheromone to false and set searching to true.
    set usingPheromone? false
    set searching? true
  ]

```

Great job! You're done.

If you are planning on participating in the Swarmathon, the following experiment is highly recommended!

EXPERIMENT!

Run the simulation 10 times with a fixed value for clusters and singles, and also set maxPheromone at 300. Record how many ticks it takes for the robots to collect all the rocks. Now set maxPheromone to 0. Run the simulation 10 times again and record how many ticks it takes for the robots to gather all of the rocks.

You can find the tick counter at the top of the simulation window.

If you completed Mars Robot 2, compare the results with your data from that experiment, or just rerun and re-record the data. Which approach is the most efficient?

NetLogo is free to download, so I hope you'll continue programming at home!

If you have any questions, you can always email me at elizabethesterly@gmail.com !