Collecting Rocks on Mars 2

In Collecting Rocks on Mars, you created a simulation of one of the Moses Biological Computation Lab's Swarmie robots collecting rocks on Mars.

In this exercise, you will go one step further and create multiple robots that work cooperatively together. The robots will now have the ability to recruit each other to a location when multiple rocks are found. This will make them more efficient at gathering rocks.

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

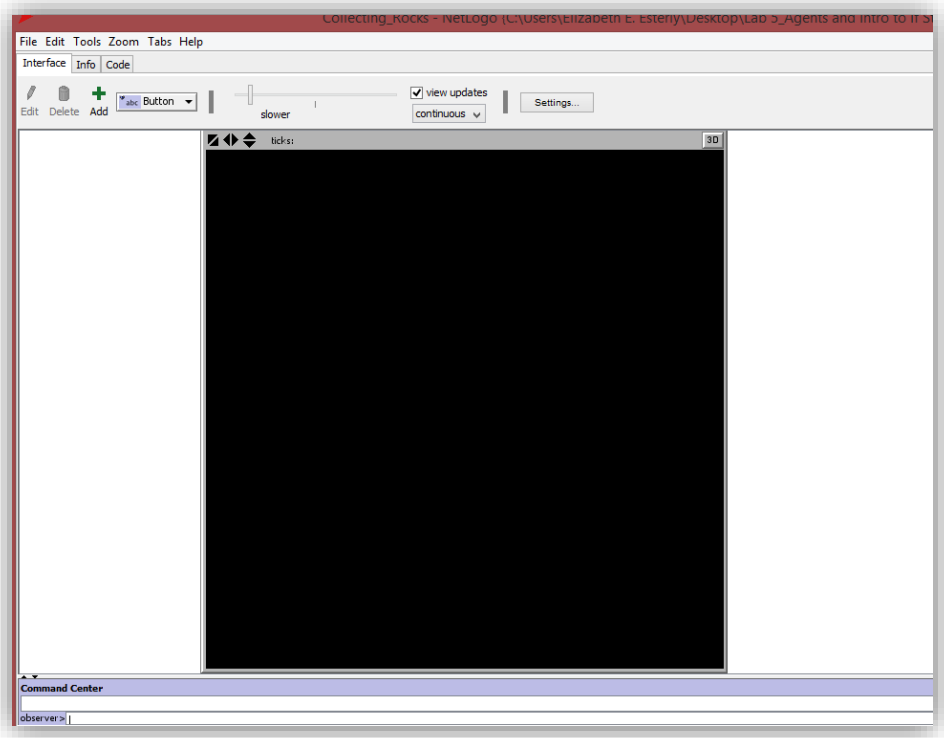
Some parts of the program are already done for you. While you are working**, don't change the code that's already been written, or the program may not run. Once you've finished the program,** you may experiment by changing the code.

1) Getting Started

1.1 Create a new folder called "CollectingRocks2" on the Desktop.

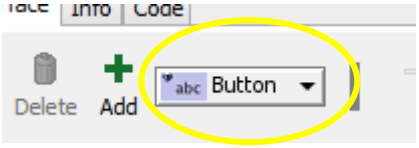
1.2 Download the .nlogo file and the mars.jpg file and put them in the folder.

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

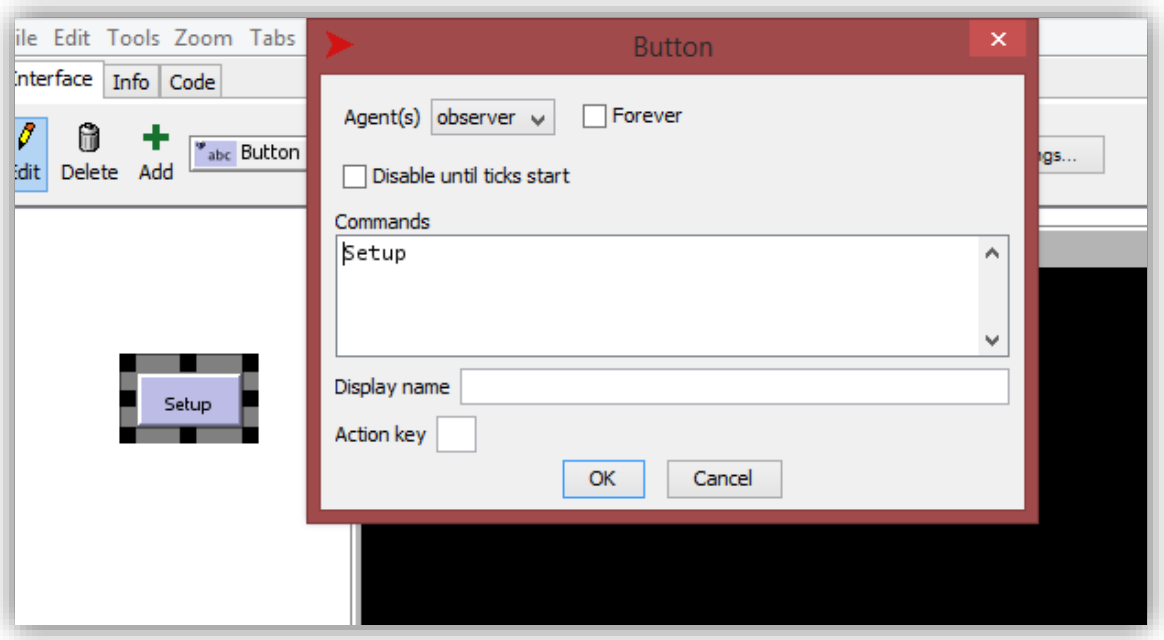


2) Setup the program

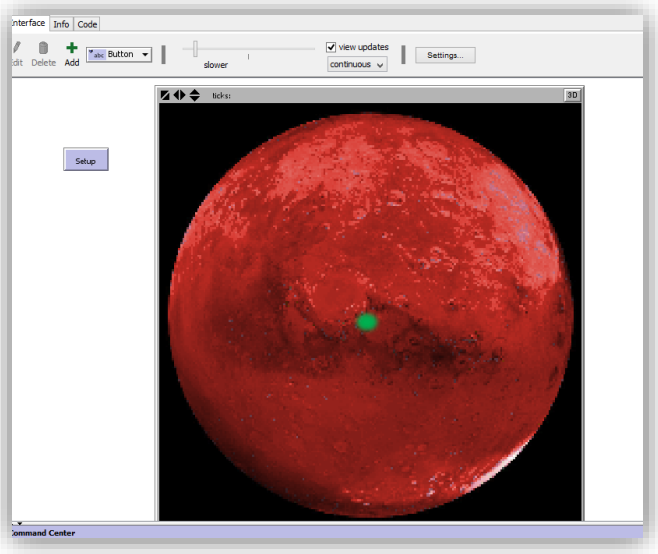
2.1 Create a new button by clicking the button next to the Add button and selecting Button from the drop-down menu (first option).



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.

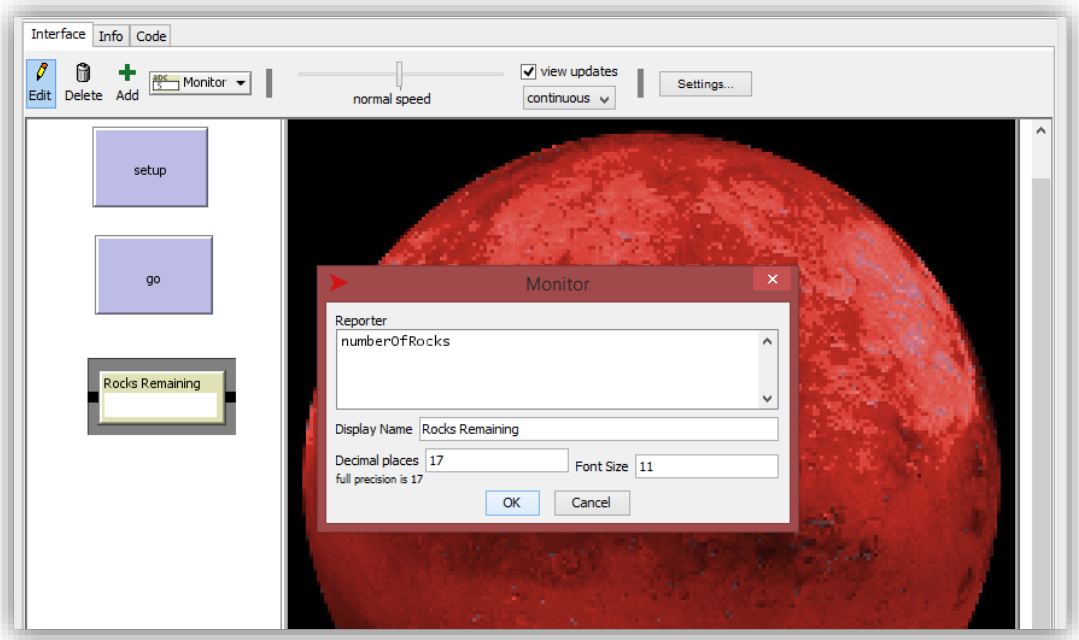


2.3 Click your new Setup button. You should see an image of Mars appear in the window after about 20 seconds. As with Mars Robot 1, the green circle in the middle is your base.



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, 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.



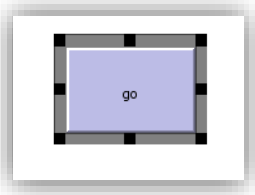
2.6 Click on the Code tab, scroll to the Setup section, and enter the following code:

```
;-------------------------------------------------------------------
;;1)Set numberOfRocks to 0 to start
set numberOfRocks 0
```

This completes setting up the monitor.

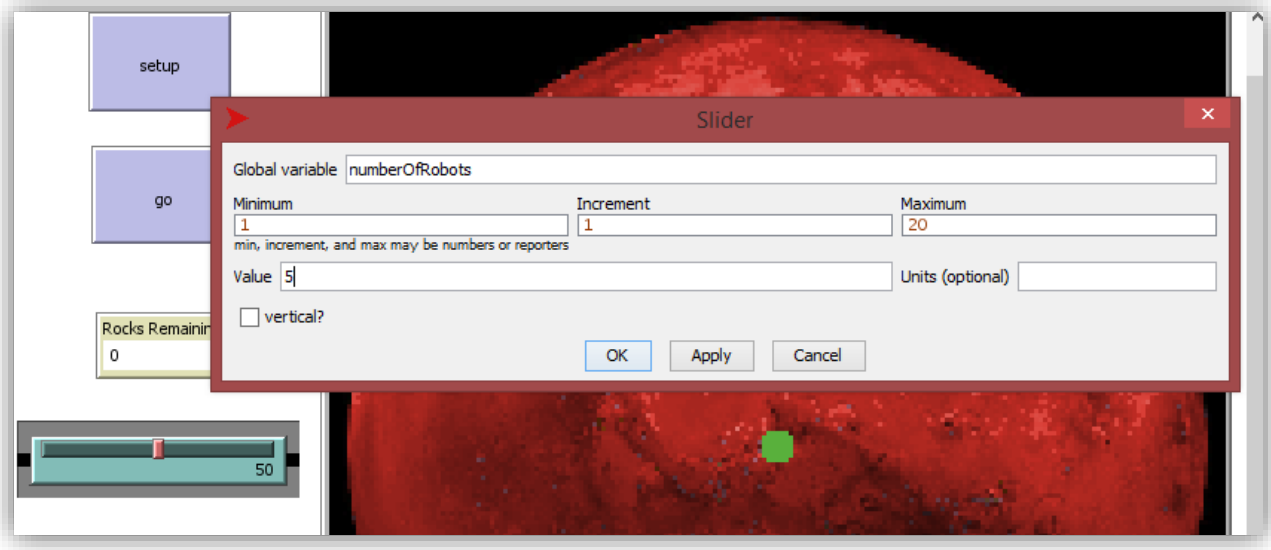
2.7 In this program, we will also use slider bars so that you may experiment with the number of robots, the number of clusters of rocks, the number of single rocks, and the radius a robot uses to recruit other robots. Let's set up these slider bars now.

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.

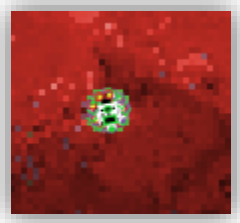
In the Global Variable box, enter numberOfRobots. Set the Minimum to 1 (there must be at least one robot) and the Maximum to 20. Leave the increment as 1. Set the value to 5. You can change the value by moving the slider bar after creating the slider. Click OK.



2.8 Let's test that our robot setup is working. Click on the Code tab, scroll to the Setup section, and enter the following code, just below where you set the numberOfRocks:

```
;;2)We'll use the slider value for numberOfRobots to determine how many
;; turtles (robots) to create.
;;Change their shape from a turtle to a robot.
;;Set their size to 8, so you can see them more clearly.
set-default-shape turtles "robot"
create-turtles numberOfRobots
[
  set size 8
]
```

2.9 Go back to the Interface tab. Click the Setup button. Five robots appear, clustered at the origin!



This completes setting up the numberOfRobots slider.

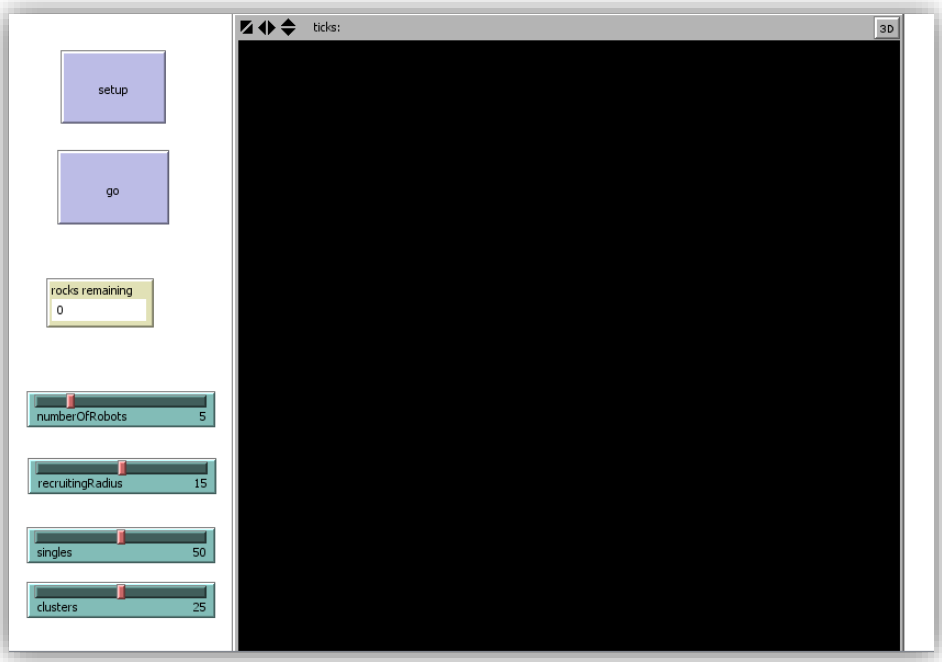
2.10 Save your program.

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

Global variable	Minimum	Increment	Maximum	Value
recruitingRadius	0	5	30	15
singles	0	5	100	50
clusters	0	5	50	25

Look at the hint from Section 2.7 if you need to move or resize your Interface buttons, monitor, or sliders. It's best to keep your Interface neat.

Your interface should now look like this:

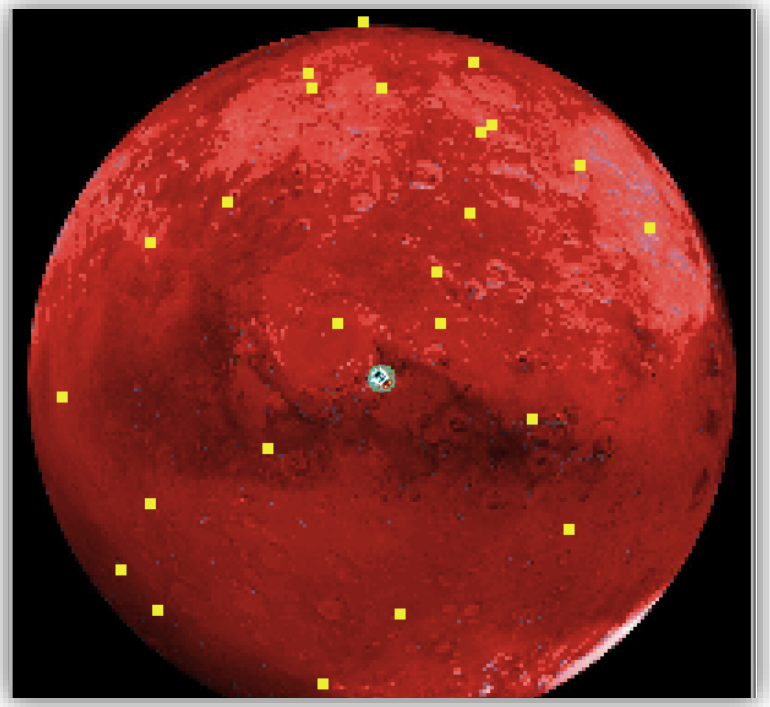


2.12 We'll complete setting up the recruitingRadius slider later in the program. Now, let's finish the singles and clusters sliders. The values of the slider bars tell us how many singles and clusters of rocks to place, but we don't have any code yet to tell the program how to place them. Let's start with clusters. Click on the Code tab and scroll to 4) in the Setup section. (We'll come back to 3). **Read the comments to understand what the code does. Do this every time you enter a code section.**

Enter the following code:

```
;; 4) Let's create some clusters of rocks. Each cluster has 9 rocks.
;; We'll use the slider value for clusters to determine how many clusters to create.
;; In Mars Robot 1, we asked 10 random patches to place clusters.
;; We didn't ask the patch its color first.
;; If the patch was black (off-planet or trench), we didn't place the cluster, but the attempt still counted.
;; In this way, we could get a different amount every time, because some black patches were usually tested.
;; In Mars Robot 2, we ask only the patches that are not black to place clusters.
;; In this way, we make sure the exact amount of clusters on our slider shows up in the
;; program.

  ask n-of clusters patches with [pcolor != black]
  [
    set pcolor yellow
    set numberOfRocks (numberOfRocks + 1)
    ask neighbors
    [
      set pcolor yellow
      set numberOfRocks (numberOfRocks + 1)
    ]
  ]
```

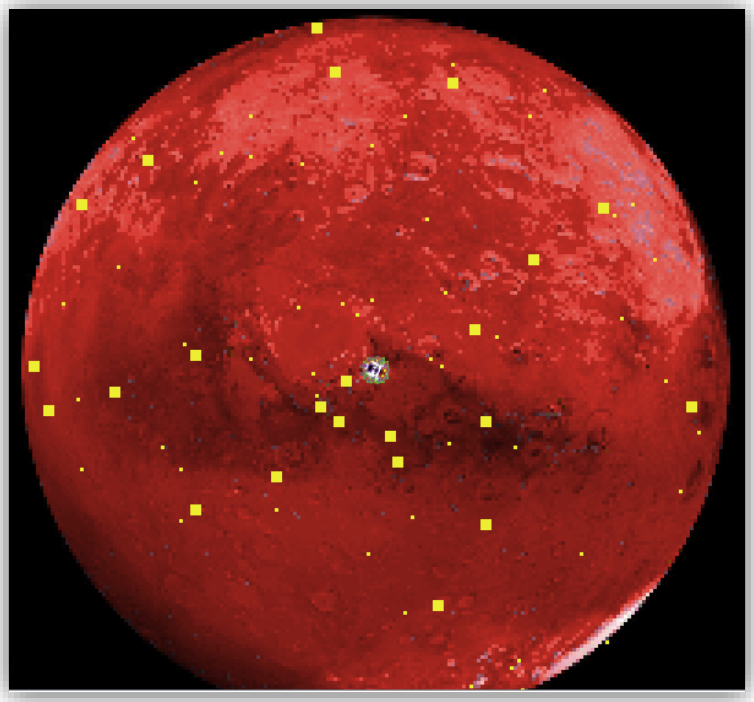
2.13 Go back to the Interface tab and click Setup. 25 clusters appear, and the Rocks Remaining monitor displays 225!



2.14 Add the following code to setup the single rocks:

```
;; 5) Now we'll place some single rocks.
;; We'll use the slider value for singles to determine how many single rocks to create.
;; We need to ask only patches that can hold a rock to place a rock, so the pcolor can't be black.
;; Since we already placed some clusters of rocks, we don't want to put a rock on top of another rock.
;; Make sure to check that the pcolor is not yellow too.

  ask n-of singles patches with [pcolor != black and pcolor != yellow]
  [
    set pcolor yellow
    set numberOfRocks (numberOfRocks + 1)
  ]
```

2.15 Go back to the Interface tab and click Setup. 50 single rocks appear along with the 25 clusters! The Rocks Remaining monitor displays 275.



This completes setting up the singles and clusters sliders.

2.15 Change your slider values for singles and clusters by adjusting the slider bars. Click Setup again to notice how this affects the simulation and the Monitor.

2.16 Save your program.

3) What does a Swarmie know?

3.1 In Mars Robot 1, our single robot knew if it was in search mode or not. In this program, we multiple modes. We also have multiple robots, all of whom must remember what mode they are in. Let's give each robot its own set of things to remember. Click the Code tab and scroll up to Globals. Enter the following code:

```
;;1) turtles (robots) each have their own list of modes and targetX and targetY.
turtles-own
[
 searching?
 returning?
 recruiting?
 recruited?
 targetX
 targetY
]
```

3.2 Now we'll return to the code block in Setup that we skipped in Section 2. Scroll down to Setup and enter the following code:

```
;;3) 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 recruited? false
  set recruiting? false
]
```

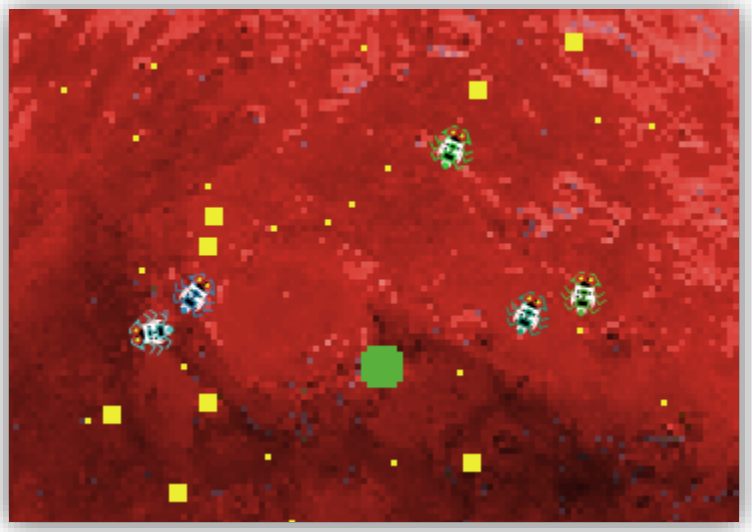
In NetLogo, each turtle (robot) already has some things that it remembers; for example, whether it is hidden or visible, or what its current coordinates are. We've now told the robots that they also have 4 modes and 1 additional set of coordinates to remember.

We'll use these 4 modes to build our Go procedure.

3.3 Save your program.

4) Let's Go!

4.1 The Wiggle procedure is identical to Mars Robot 1, so it is already written for you here. Click Setup and then Go. The 5 robots start moving!



4.2 Go to Tools → Halt to stop the program.

4.3 Change your slider values for numberOfRobots by adjusting the slider bar. Click Setup and go again. The number of robots in the simulation should match your slider value.

4.4 Now we'll tell the robot what to do based on what mode it is currently in. **A robot can only be in one mode at a time.** Scroll down to the Go procedure and enter the following code in your program:

```
;;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 recruited? it should move-to-friend
if recruited?
[move-to-friend]

;;4) if a robot is recruiting? it should call-friends
if recruiting?
[call-friends]
```

You may remember the [look-for-rocks] and [return-to-base] procedures from Mars Robot 1.

4.5 Save your program.

5) Controlling the Robots: Looking For Rocks

5.1 The [look-for-rocks] procedure is more complex than in Mars Robot 1. If a robot finds a rock, it must look around to see if there are other rocks in the area. If there are, it will set itself to recruiting? mode. On the next step, it will call-friends in the area of the recruitingRadius to move-to-location if they are not busy.

If there are no more rocks, it will set itself to returning? mode to return-to-base and drop off the rock.

This is why it is important that each robot has its own memory of its current state.

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

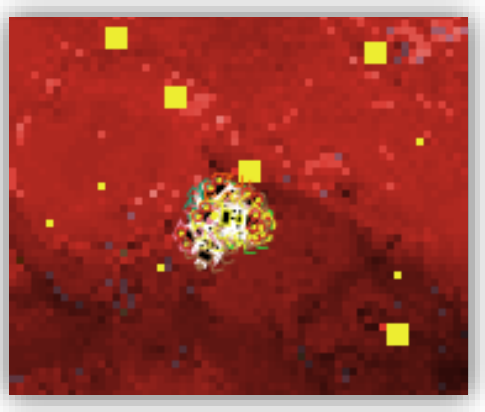
Scroll down to the look-for-rocks procedure and enter the following code into your program:

```
;;1)If the patch color of patch-ahead 1 is yellow,
if [pcolor] of patch-ahead 1 = 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"
  set pcolor red

  ;;3)Turn search mode off.
  set searching? false

  ;;4)Create 2 variables, turnOnRecruiting? and turnOnReturning?
  ;;and set them to false.
  ;;We have to do this because we need to check some things from a patch context.
  let turnOnRecruiting? false
  let turnOnReturning? false

  ;;5) Ask the patches 360 degrees around us
  ask patches in-radius 1
  [
    ;;6) Use an ifelse statement.
    ;;If the patch is yellow (a rock), we want to tell closeby robots
    ;;that there are rocks here.
    ;;turnOnRecruiting by setting its value to true.
    ifelse pcolor = yellow
    [ set turnOnRecruiting? true ]

    ;;7) else there's no more rocks here.
    ;;We don't need to recruit, we need to return.
    ;;turnOnReturning by setting its calue to true.
    [ set turnOnReturning? true ]
  ]

  ;;8) We're done testing patches.
  ;;Set the value of recruiting? to the value of
  ;;turnOnRecruiting? and
  ;;set the value of returning? to the value of
  ;;turnOnReturning?

  set recruiting? turnOnRecruiting?
  set returning? turnOnReturning?
]
```

5.2 Save your program.

5.3 Choose any number of robots you want by adjusting the numberOfRobots slider. Click Setup and then Go to run your program.

What's going on? The robots are stuck at the base after picking up a rock and returning it!



TROUBLESHOOTING!

Think about why the robots would behave this way.

HINTS

- We don't need to worry about recruiting?/call-friends or recruited?/move-to-friend.
- The robots that are searching? are successfully finding and picking up a rock (doing the look-for-rocks procedure). This means that searching? is working.
- look-for-rocks tells a robot to switch to returning? mode if it found a rock. The robots return-to-base and drop off the rock. But they are not switching back to searching? and get stuck in returning? mode!
- See if you can find the missing code and fill it in on your own! **There is a hint there to help you. Don't change any code that you've already written.**
- If you are successful, the robots will leave the base after dropping off the rock and begin searching again. Test your code by clicking Setup and Go.
- If you are stuck, don't get frustrated! It takes time to learn how to program. The answer is on the next page.

Remember that you can stop your program by going to Tools → Halt.

ANSWER TO TROUBLESHOOTING

Enter the following code into your return-to-base procedure.

```
;;!!!) Turn off returning mode, and turn on searching mode.
    set returning? false
    set searching? true
  ]
```

5.4 Ensure that your simulation is working properly by clicking Setup and Go. The robots should act just as they did in Mars Robot 1. It's important to check after every step. If there's an error, it needs to be fixed before moving on.

5.5 Save your program.

6) Controlling the Robots: Calling Friends and Moving to Friends

6.1 Our robots are successfully gathering rocks and dropping them off! Now we need to make them work together.

Recall from the last section that a robot who is recruiting? found more rocks in its area in look-for-rocks. In the Go procedure, we see that a recruiting? robot will call-friends.

The call-friends procedure simulates a robot transmitting its current coordinates (pxcor and pycor) to a satellite as rockLocationX and rockLocationY. It then checks in the recruitingRadius to see if there are any robots who are not busy.

A robot is busy if it is currently recruiting, returning, or recruited. Only robots who are currently searching? can be recruited?.

recruited? robots set their personal targetX and targetY coordinates to the coordinates uploaded to the satellite (rockLocationX and rockLocationY). On the next step, they will move-to-friend.

We'll finally finish setting up the recruitingRadius slider here.

Enter the following code into your program in the call-friends section:

```
;;1)Set the global location variables to the robot's current position
;;to tell recruited? robots where to go.
;;This is like uploading some coordinates to a satellite.
set rockLocationX pxcor
set rockLocationY pycor

;;2)Ask the robots (turtles) in the recruiting radius
ask turtles in-radius recruitingRadius
[
  ;;3) if they are searching?
  if searching?
  [
    ;;4) Turn off searching mode and turn on recruited mode,
    set searching? false
    set recruited? true

    ;;5) and set their turtles-own coords to the global rock location coords so they'll
    ;;head in that direction.
    ;;This is like getting the destination coordinates from a satellite.
    set targetX rockLocationX
    set targetY rockLocationY
  ]
]

;;6) The robot finished recruiting.
;;Turn off recruiting mode, and turn on returning mode
;;so it drops off the rock it is holding.
set recruiting? false
set returning? true
```

6.2 The call-friends and move-to-friend procedures work together. They must both be written to see any change in the program. If you choose, click Setup and Go to run your program now to verify that nothing has changed.

6.3 Save your program.

6.4 Enter the following code into your program in the move-to-friend procedure:

```
;;1) Make an ifelse statement.
;;If our target coordinate is our current coordinate,
ifelse targetX = pxcor and targetY = pycor
[
  ;;2) We reached our destination.
  ;;Turn off recruited mode.
  ;;Turn on searching mode so we detect the nearby rocks.
  set recruited? false
  set searching? true
]
;;3) else face the target coordinates
[ facexy targetX targetY ]
```

6.5 Save your program.

6.6 Set the numberOfRobots to 15, the recruitingRadius to 25, the number of singles to 60, and the number of clusters to 30. Click Setup and Go to run your program. The robots are working together like crazy! Notice that recruited robots have the label recruited flash on the screen while they're in that mode.

6.7 Speed up the simulation. Notice that robots tend to make patterns when recruitment levels are high, like with the settings we tried above. This phenomenon is called **emergent behavior**.

EXPERIMENT!

Run the simulation again 10 times with the values from **6.6**. Make a note each time of how many ticks it took the robots to gather all the rocks. Take the average of those scores.

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

Now run the simulation again 10 times, but with the recruiting radius turned down to 0.

Make a note each time of how many ticks it took the robots to gather all the rocks. Take the average of those scores.

Was the result what you expected? Why or why not?

Change the values to whatever you want!

Look for patterns. Which combinations are the most effective? Which are the least effective? Why?

What would this mean for real robots on Mars?

That's it! Thanks for participating.

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 !