Agent Based Modelling – Ants with pheromones

In this lesson we will develop a model where ants find a food source. Though each ant follows a set of simple rules, the colony as a whole acts in a sophisticated way. When an ant finds a piece of food, it carries the food back to the nest, dropping a chemical as it moves. When other ants "sniff" the chemical, they follow the chemical toward the food. As more ants carry food to the nest, they reinforce the chemical trail.

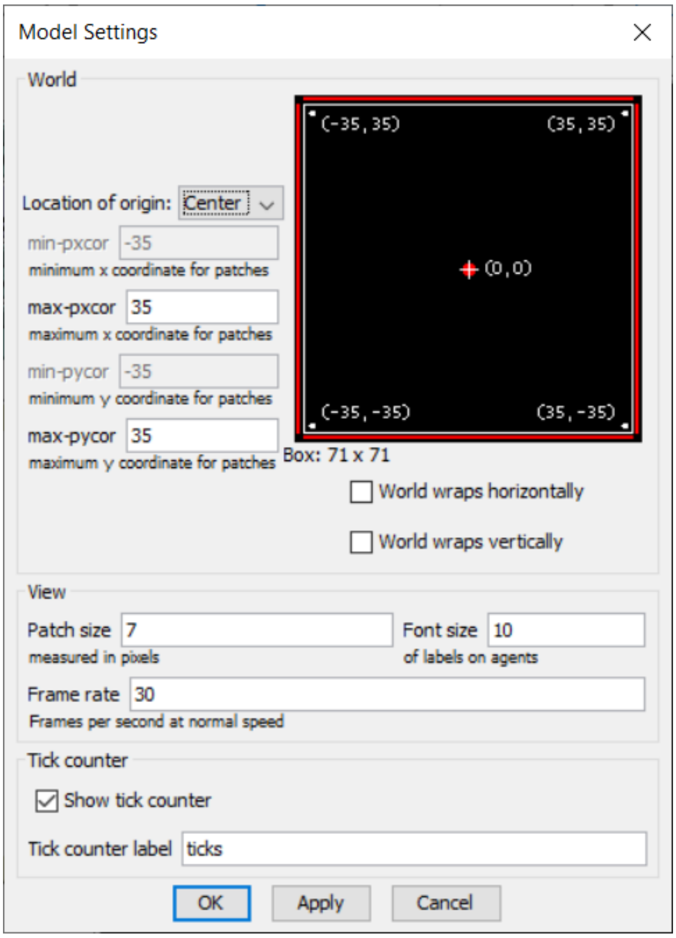
Before starting have a look at this video: https://youtu.be/vG-QZOTc5_Q

Reference

Wilensky, U. (1997). NetLogo Ants model. http://ccl.northwestern.edu/netlogo/models/Ants. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Step 1 - Setting up the environment

- Open NetLogo and save the file in an appropriate location (student drive)
- Click on the setting button and modify the environment to the following spesification:



Model Settings

World

Location of origin: Center

min-pxcor -35
minimum x coordinate for patches

max-pxcor 35
maximum x coordinate for patches

min-pycor -35
minimum y coordinate for patches

max-pycor 35
maximum y coordinate for patches

Box: 71 x 71

□ World wraps horizontally

□ World wraps vertically

View

Patch size 7
measured in pixels

Font size 10
of labels on agents

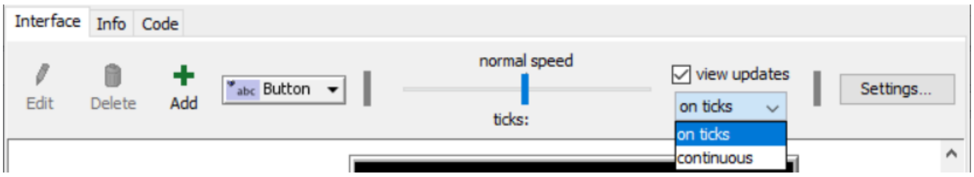
Frame rate 30
Frames per second at normal speed

Tick counter

☑ Show tick counter

Tick counter label ticks

Once this is done on the interface set the make sure view updates is ticked and ensure the world updates continuously rather than on ticks as shown below.



☑ view updates

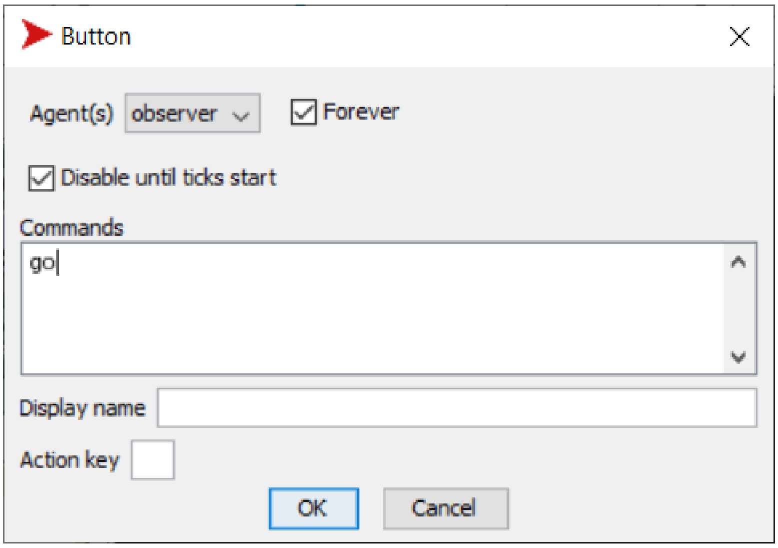
on ticks

Step 2 – Creating an interface

In this model we want to be able to easily change the parameters of the model, we will therefore create some features to facilitate this.

Step 2.1 – Creating command buttons

Select button from the option list then click the add button and the button dialogue box will appear where you can name the variable and sent the command for it to activate.



Agent(s) observer

☑ Forever

☑ Disable until ticks start

Commands

```
go
```

Display name

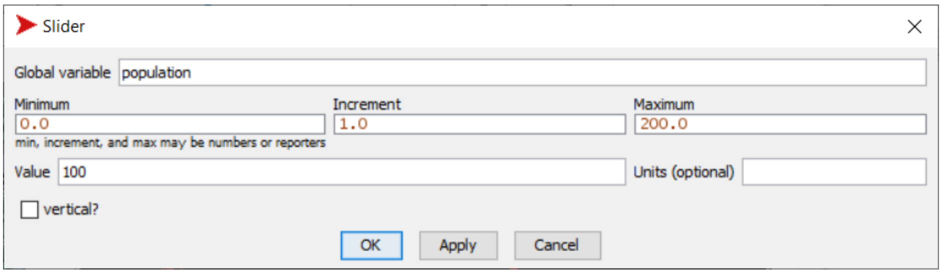
Action key

Create the following buttons with the stated parameters:

Name	Commands	Agent(s)	Forever	Disable until ticks start
go	go	Observer	Ticked	
setup	setup	Observer	Unticked	

Step 2.2 – Creating sliders

Select slider from the option list then add a slider with the following details.



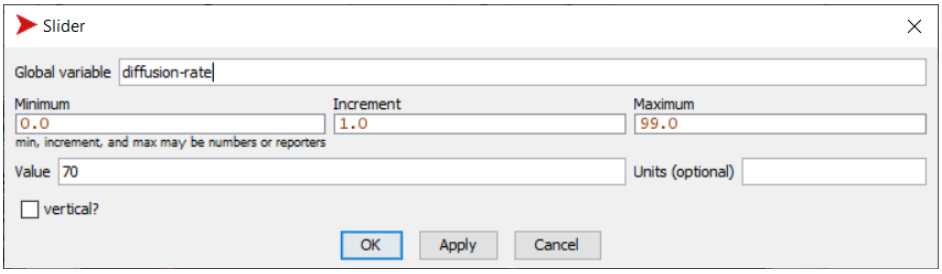
Global variable population

Minimum 0.0 Increment 1.0 Maximum 200.0

min, increment, and max may be numbers or reporters

Value 100 Units (optional)

□ vertical?



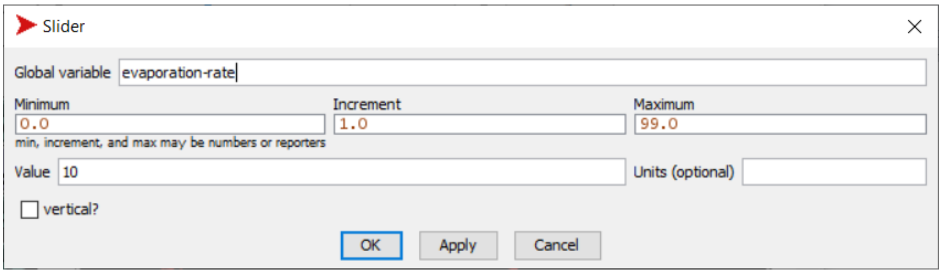
Global variable diffusion-rate

Minimum 0.0 Increment 1.0 Maximum 99.0

min, increment, and max may be numbers or reporters

Value 70 Units (optional)

□ vertical?



Global variable evaporation-rate

Minimum 0.0 Increment 1.0 Maximum 99.0

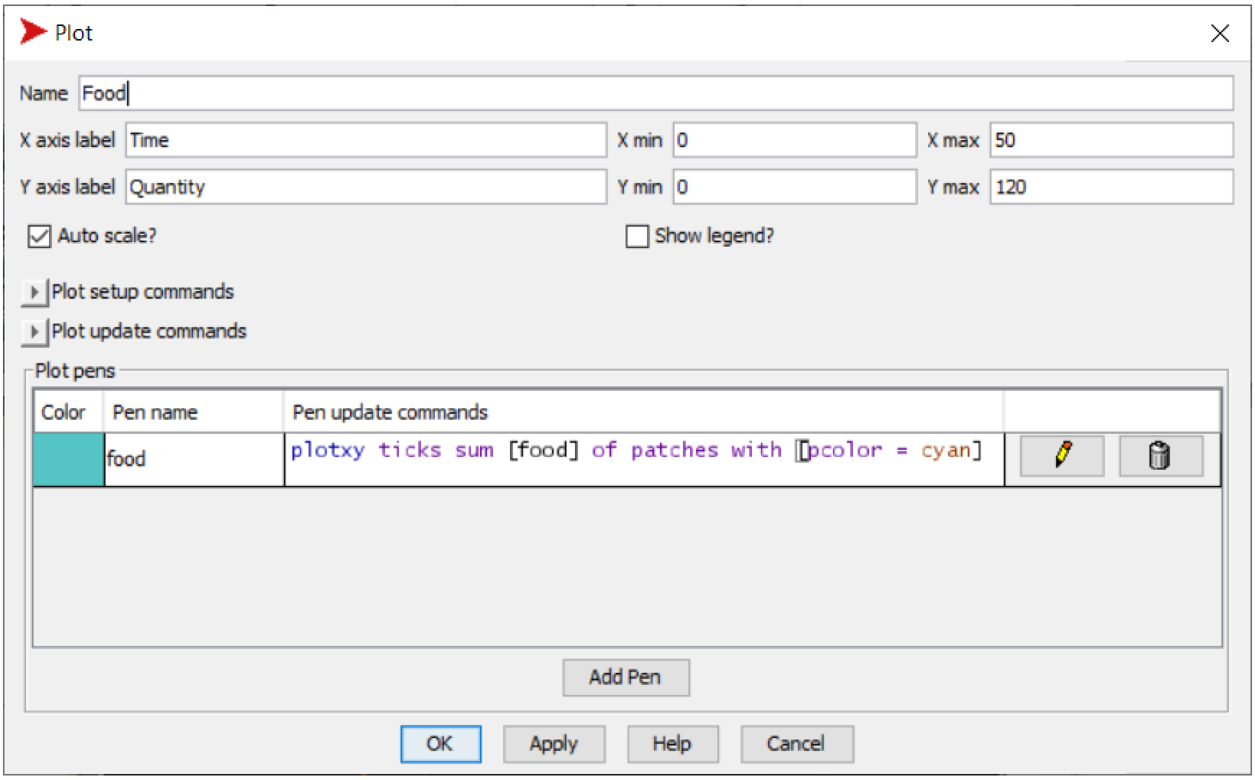
min, increment, and max may be numbers or reporters

Value 10 Units (optional)

□ vertical?

Step 2.3 – Creating a plot

Select plot from the option list then add a plot with the following details.



Name: Food

X axis label: Time — X min: 0 — X max: 50

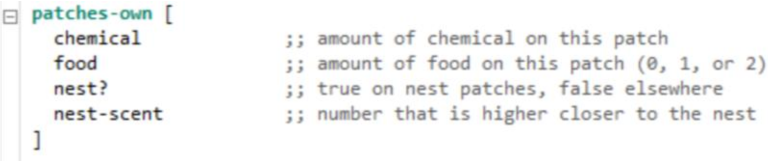
Y axis label: Quantity — Y min: 0 — Y max: 120

☑ Auto scale? ☐ Show legend?

Color	Pen name	Pen update commands
🟦	food	`plotxy ticks sum [food] of patches with [pcolor = cyan]`

Step 3 – Setup patch variables

Add the following patch variables to store world parameters:

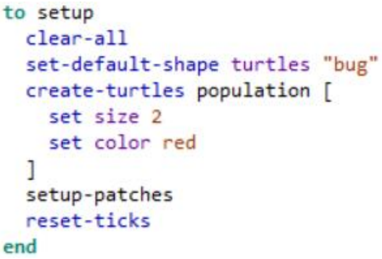


```
patches-own [
  chemical              ;; amount of chemical on this patch
  food                  ;; amount of food on this patch (0, 1, or 2)
  nest?                 ;; true on nest patches, false elsewhere
  nest-scent            ;; number that is higher closer to the nest
]
```

Step 4 – Write setup function

Write the setup function to create ants and call further function:

1) Clear everything in the world
2) Change the default shape of a turtle to a bug
3) Create x number of turtles based on the value of population
4) Set the size of the turtles created to 2 and the color to red
5) Call the setup-patches function (yet to be created)
6) Reset the ticks in our world

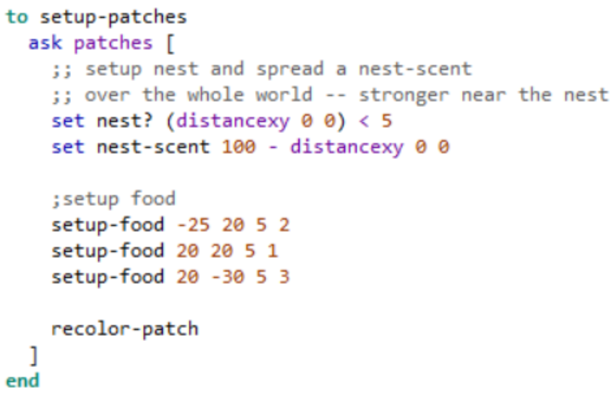


```
to setup
  clear-all
  set-default-shape turtles "bug"
  create-turtles population [
    set size 2
    set color red
  ]
  setup-patches
  reset-ticks
end
```

Step 5 – Setup patches

Write the following code to set the parameters of the nest and food sources:

1) Select all the patches and do the following:
2) Set the patches-own variable nest to be in the centre with a 5 patch radius from the centre
3) Set the value of the next scent to 100 right in the centre of the nest (again patches-own variable)
4) Call the setup-food function (yet to be created) setting the coordinates, radius and amount of the food source
5) Call the recolour-patch function (yet to be created)

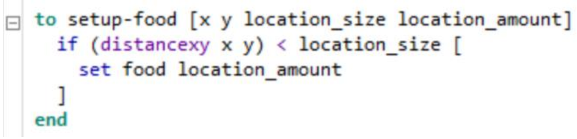


```
to setup-patches
  ask patches [
    ;; setup nest and spread a nest-scent
    ;; over the whole world -- stronger near the nest
    set nest? (distancexy 0 0) < 5
    set nest-scent 100 - distancexy 0 0

    ;setup food
    setup-food -25 20 5 2
    setup-food 20 20 5 1
    setup-food 20 -30 5 3

    recolor-patch
  ]
end
```

6 – Setup food sources

Write the following code to create a function that sets up the food sources:

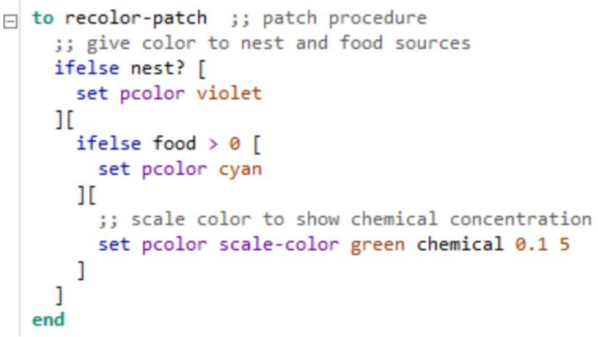
1) The input parameters are the x and y coordinates of the food source, the overall size of the food source and the amount of food in that source.
2) The if statement selects the patches within the given radius of the x and y coordinates.
3) The patches own variable food is set to the value passed in under the name location_amount.



```
to setup-food [x y location_size location_amount]
  if (distancexy x y) < location_size [
    set food location_amount
  ]
end
```

Step 7 – setup color patches for visualisations

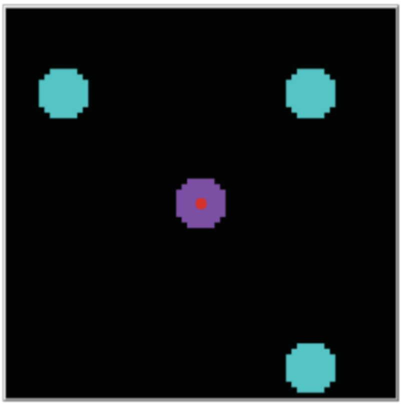
With the following code to update the patch colors based on the world parameters and chemicals deposited:



```
to recolor-patch  ;; patch procedure
  ;; give color to nest and food sources
  ifelse nest? [
    set pcolor violet
  ][
    ifelse food > 0 [
      set pcolor cyan
    ][
      ;; scale color to show chemical concentration
      set pcolor scale-color green chemical 0.1 5
    ]
  ]
end
```

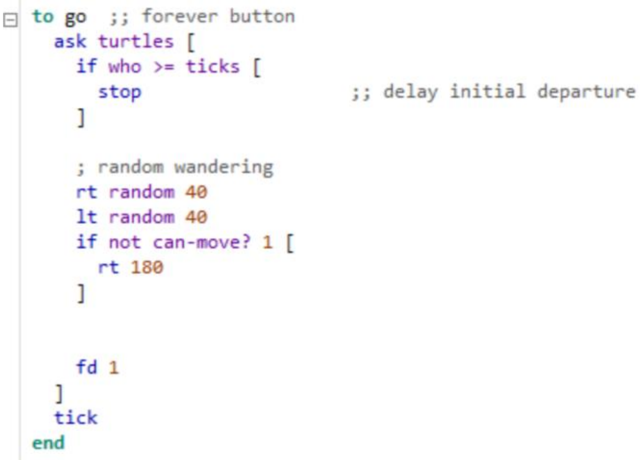
Step 8 – Test your model

It should look like this:



Step 9 – Create the go function

Write the following code to make your turtles move. 2 critical elements in this model are time and delaying the release of the ants. Time will be used for pheromone dissipation and the delay will increase the efficiency of the ants.



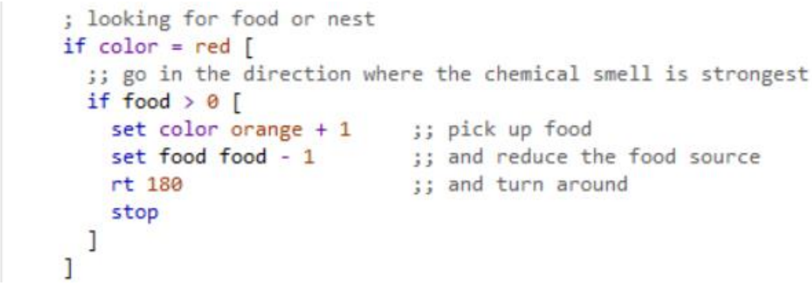
```
to go  ;; forever button
  ask turtles [
    if who >= ticks [
      stop                    ;; delay initial departure
    ]

    ; random wandering
    rt random 40
    lt random 40
    if not can-move? 1 [
      rt 180
    ]


    fd 1
  ]
  tick
end
```

Step 10 – Test your model

Your ants should wander around randomly

Step 11 – Collecting food

Add the following code within ask turtles so your ants can collect food and show this in their appearance:

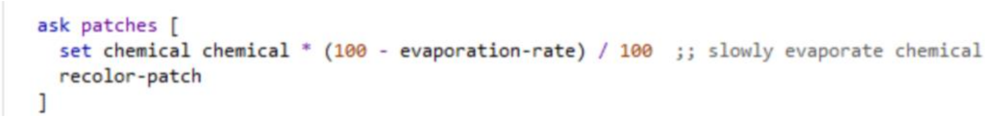


```
    ; looking for food or nest
    if color = red [
      ;; go in the direction where the chemical smell is strongest
      if food > 0 [
        set color orange + 1     ;; pick up food
        set food food - 1        ;; and reduce the food source
        rt 180                   ;; and turn around
        stop
      ]
    ]
```

Your ants should now collect food, turn orange and turn 180 degrees.

Step 12 – Updating visualisations

Add the following code in the go function but outside of the ask turtles section to update the patch colors to show the removal of food and update the chemical values of the patches:



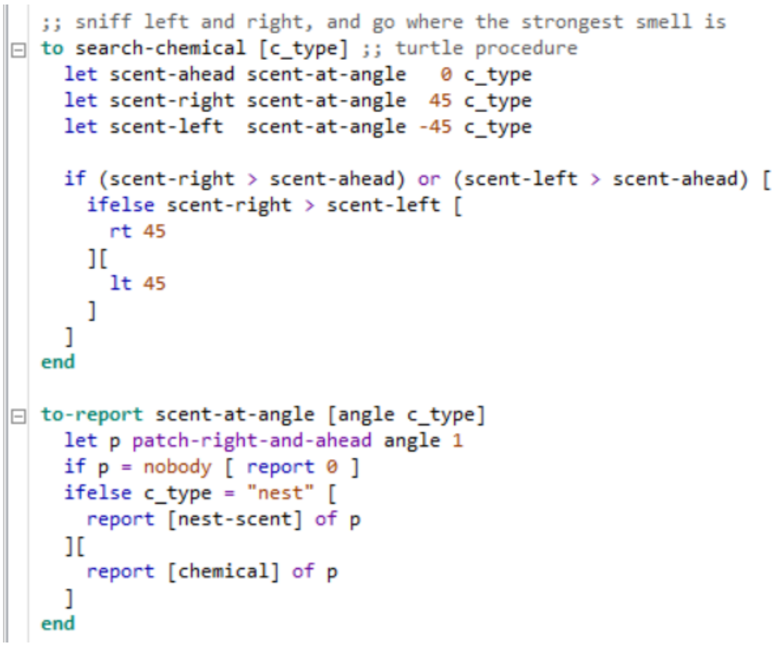
```
  ask patches [
    set chemical chemical * (100 - evaporation-rate) / 100  ;; slowly evaporate chemical
    recolor-patch
  ]
```

Step 13 – Test your model

The food sources should now appear to be reducing when the ants are picking it up.

Step 14 – searching for scents

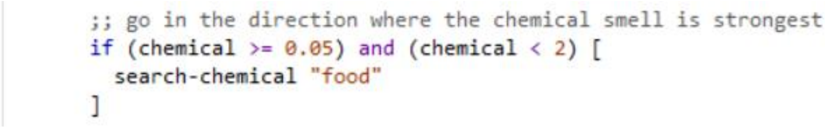
Write the following code to facilitate the search for either the nest or food based on the scent being searched for. This code is effectively getting the ant to sniff left and right before moving forward base on what it smells.



```
;; sniff left and right, and go where the strongest smell is
to search-chemical [c_type] ;; turtle procedure
  let scent-ahead scent-at-angle   0 c_type
  let scent-right scent-at-angle  45 c_type
  let scent-left  scent-at-angle -45 c_type

  if (scent-right > scent-ahead) or (scent-left > scent-ahead) [
    ifelse scent-right > scent-left [
      rt 45
    ][
      lt 45
    ]
  ]
end

to-report scent-at-angle [angle c_type]
  let p patch-right-and-ahead angle 1
  if p = nobody [ report 0 ]
  ifelse c_type = "nest" [
    report [nest-scent] of p
  ][
    report [chemical] of p
  ]
end
```

Step 15 – Calling the search chemical function to find food

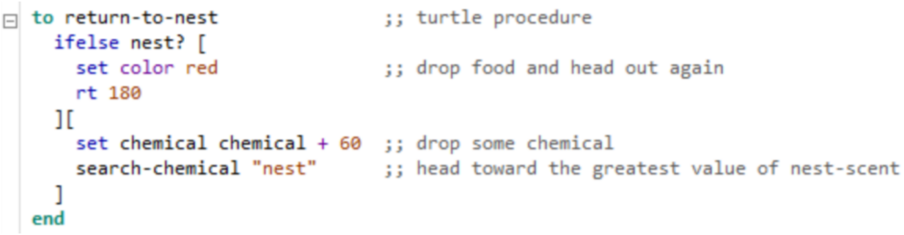
Add the following code within the if color = red if statement to search for food.



```
    ;; go in the direction where the chemical smell is strongest
    if (chemical >= 0.05) and (chemical < 2) [
      search-chemical "food"
    ]
```

Step 16 – Creating a return to nest function

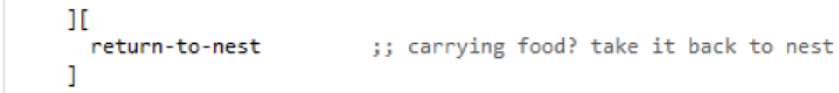
Write the following function to allow your ant to find their way back to the nest when they have food.



```
to return-to-nest                   ;; turtle procedure
  ifelse nest? [
    set color red                   ;; drop food and head out again
    rt 180
  ][
    set chemical chemical + 60      ;; drop some chemical
    search-chemical "nest"          ;; head toward the greatest value of nest-scent
  ]
end
```

Step 17 – calling the return to nest function

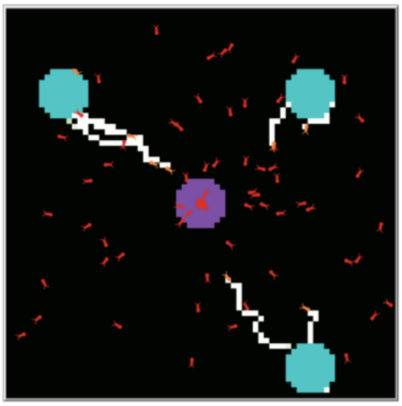
Change the if color = red to ifelse and the following code to call the newly created return to nest function:



```
  ][
    return-to-nest          ;; carrying food? take it back to nest
  ]
```

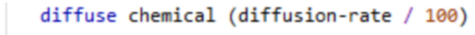
Step 18 – Test your model

It should look something like this:



Step 19 – Diffusing the chemical

In the go function add the following line of code to diffuse the chemical:



```
diffuse chemical (diffusion-rate / 100)
```

Step 20 – Play with your model

Experiment with the parameters and see how the sliders impact the model. It should look something like this: