Exploring Breeds

The focus of this week's lab will be using NetLogo to model the interaction of different organisms. Different types of turtle's in NetLogo can be specified by defining breeds. Each breed can have its own attributes and can be controlled with breed-specific commands. You can also specify different shapes and sizes for each breed.

The first model we will explore will be a case of amoeba feeding on randomly floating pieces of food. We will use NetLogo's Behavior Space to examine how a variety of parameters influence the growth curves for the amoeba and will make quantitative comparisons with the predictions of the Logistic Model. We will then gradually make additions and enhancements to this model, to reflect a more realistic interaction and see how these changes alter the growth of the amoeba.

Lab

Before starting your program set the graphics window to be 30 by 30 with patch size 8.
Now click on the procedures tab and at the top of the page type the command

```
breeds [amoeba food]
```

This defines two new types of turtles called amoeba and food. Now create a setup procedure which looks like the following:

```
to setup
   clear-all
   setup-amoeba
   setup-food
end
```

and then define the two setup commands as follows:

```
to setup-amoeba
  create-custom-amoeba amoeba-number
    [set color brown
     set shape "circle"
     setxy random screen-size-x random screen-size-y]
end

to setup-food
 create-custom-food food-number
    [set color green
     set shape "circle"
     setxy random screen-size-x random screen-size-y]
end
```

In order for the above code to be complete you need to specify
the range of values for the variables `amoeba-number` and `food-number`
using sliders on the interface page. Let the values range from 0 to 500 in steps of 1.

Shapes editor

In order to make the model more representative of what we might see under a microscope lets design our amoebas and food to have “realistic” shapes. Open up the Shapes Editor under the Tools menu. There are many predefined shapes and there are more that can be imported.

However, lets design our own shape. Click on “new” and you will find yourself with a rather primitive graphics editing window. Draw a basic amoeba shape – you can add a few pseudopods and organelles if you wish, but don't get too fancy. Save your shape with name amoeba, and then create a new, smaller shape called food. There is no need to get to elaborate – some irregular polygon will do. Save this one as food, and then return to your setup procedure and change the shapes from “circle” to “amoeba” and “food” as appropriate.

Movement
Now you are ready to get the amoeba and food to move. In this first model we will just have the breeds move randomly – with perhaps the amoeba moving faster than the food, since they have their own locomotion. Define a `go` command as follows:

```
to go
  ask food [ wiggle .5 ]
  ask amoeba [ wiggle 1]
end
```

where `wiggle` is a command which must be defined. As written this command takes one input which specifies how far the turtles move in each time step. To define commands with inputs you specify the input variable in square brackets after the command name. A suitable wiggle command is defined below. You can modify this as you wish.

```
to wiggle [amount]
      left random 40
      right random 40
      forward amount
end
```

Test your code thus far by adding setup and go buttons on the interface.

Eating and Reproduction
At the moment all that happens is amoeba and food move randomly across the environment. Now we need to program some mechanism for the amoeba to eat the food and then reproduce. We will have amoeba only eat food if they find themselves on the same patch as the food. After eating food they will gain some energy. When their energy surpasses some defined threshold they will divide in half. Add two new procedures to your `go` procedure:

```
ask amoeba [eat-food]
ask amoeba [reproduce]
```

where `eat-food` can be defined as follows

```
to eat-food
     without-interruption [
           if not (count food-here = 0) [
                 ask random-one-of food-here [die]
                 set energy  energy + food-energy
              ]
      ]
end
```

The `without-interruption` command makes sure that each amoeba runs all the commands before passing control over to the next amoeba. This way two amoeba do not try to eat the same piece of food at the same time.

In order for the `eat-food` code to work we need to make a slider for `food-energy` on the interface. Have this range from 1 to 10, with default 10. We also need to specify the amoeba energy variable. Add the following statement under the line defining the breeds

```
amoeba-own [energy]
```

and then in the setup-amoeba procedure add a line

```
set energy 10
```

This specifies the initial energy of the amoeba.

Turtles will reproduce if their energy reaches twice this amount. A suitable `reproduce` command might be

```
to reproduce
  if (energy >= 20) [
    set energy energy / 2
    hatch 1 [ wiggle 1 ]
    ]
end
```

Note: when when a turtle calls the hatch command, the new turtles have all the same attributes as the parent. Consequently, we do not need to specify these attributes, unless we want offspring to differ in some way from their parents.

Run your simulation to see if it behaves appropriately. Play around with different values of the parameters. Add a line at the bottom of the `go` command to get the program to stop when all the food is consumed. Something like this will do.

```
if (count food = 0 ) [stop]
```

Plotting your data

Add a labeled plot and a monitor to your interface showing the population of amoeba over time. You may want to glance at last week's tutorial to see how best to do this.

Behavior Space.

After playing with the parameters in the model, you should notice that the growth pattern seems to be logistic in nature. In order to do more with our model than watch it we need to be able to analyze the data that is generated and compare it with either data in the field or some analytical model that we have derived. Lets compare this data with the predictions of the logistic model $\Delta N = rN(1-N/K)$. To do this we will make use of NetLogo's Behavior Space. Behavior Space is a way to do multiple runs of your model and sweep through different values of your parameters. The data is then outputted to a file with comma separated values (csv) which can be read by a data analysis package such as Excel.

From the Tools menu select Behavior Space and choose Edit Experiment Setup. The first window specifies the values of the parameter you want to test. For this experiment lets allow the food-number to range from 100 to 500 in steps of 100, and keep the food-energy fixed at 10 and the initial number of amoeba constant at 1. This is what you should type in the first window

```
["food-number" [100 500 100]]
["food-energy" 10]
["amoeba-number" 1]
```

In the next window you specify the number of runs for each combination
Lets run each combination 3 times so that we can average runs to reduce statistical error.

The next window specifies the data you want to analyze. In this case it should be

```
count amoeba
```

The next two windows should be fine with their default values.
For the final two windows
set “Stop after this many steps” to 0 and then
set “Stop if this reporter becomes true” to

```
count food = 0
```

click “ok” and then run the experiment. You will find that it runs much more quickly if you uncheck “update graphics” and “update plots and monitors”.

Save your data file as Lab3Data.csv in your CAL MathOrigins student folder.

Analyzing the data

Open up the data file in Excel. The experimental runs should be neatly arranged in columns for your analysis. Our purpose here is to compare the data with the predictions of the Logistic model. In particular, for the logistic model we expect that the per-capita growth rate $\Delta N/N$ will decrease linearly, with the y-intercept being the intrinsic growth rate *r* and the x-intercept being the carrying capacity *K*. The data is grouped into three columns because we did three runs for each parameter choice. Therefore, next to each group of three runs insert a new column to find the average and then another column to calculate $\Delta N/N$. Plot $\Delta N/N$ vs N. Plot the data and do a linear fit to your data. If a linear fit is a good fit, then obtain estimates for *r* for each run. For which values of food-number is the linear fit best? Can you explain why?

Make a plot of intrinsic growth rate, *r*, vs food-number and see if you can find a quantitative power law relationship between intrinsic growth rate and quantity of food. Can you explain your observations?

Extension to the Model

In this lab you have been introduced to a number of new features of NetLogo in a model for the growth of amoeba. For homework you will be asked to implement enhancements to this model that will model more closely how a predator goes about catching its prey. You may continue to use the amoeba-food analogy, or you may choose some other predator prey system the more closely fits your choices of parameters

Homework Questions: Predator/Prey Pursuit Problems

1. Save your lab 3 model, giving it the name “lastname_firstname_week_3.nlogo”. When you are finished the homework drop this file in the dropbox folder.
2. The first change to the model will be to arrange for amoeba to die if they do not get enough food.
   (a) Make a slider called `metabolism` which ranges from 0 to 2 in steps of 0.1. Then change the model so that amoeba lose an amount of energy equal to `metabolism` at each time step.
   (b) Each time step run a death procedure which kills all amoeba whose energy is less than 0.
3. In order that energy is not slowly drained away from the system have a certain amount of new food appearing at random locations on the screen at a rate determined by a variable which you might call `food-growth-rate`. Choose an appropriate range so that amoeba can grow.
4. One way to make the model more realistic is to include a `infancy-period` during which time a new amoeba will not move, hunt for food or reproduce. You could implement this by defining an `amoeba-own` variable called `age`. Have `age` start at 0 and increment by one each time step. Only allow amoeba to move, hunt for food and reproduce when `age` is greater than `infancy-period`. Remember to set the age of new amoeba (and the amoeba who hatched) it to zero in the `reproduce` procedure.
5. Another enhancement would be to take into acount the fact that amoeba (or other predators) can sense their food and move towards it.
   (a) Make a slider called `sensing-radius` ranging from 0 to 10.
   (b) Write a `find-food` procedure for the amoeba which sets their heading towards the nearest piece of food within a radius of `sensing-distance` before taking a step. Hint: In the procedure first create an agentset of food within the sensing radius. Then check that there is some food in this agent set (use the `any?` primitive), finally set the heading towards the food in the agent set with the minimum distance to the amoeba (use `distance myself` to measure the distance from each piece of food to the amoeba).
6. Explore your model by changing the parameters. Find different combinations of parameter values that lead to approximately linear growth to the carrying capcaicty, to an oscillating population of amoeba, a chaotic population pattern (you may have to adjust your `food-energy` variable to have higher upper bound to see this.)
7. Now use Behaviour Space to explore how the model behaves with `metabolism` and `food-growth-rate` set to zero. Fix the food-number at a reasonable setting and then change the `sensing-radius` and `latent-period` to see what combination yields a growth curve which does not fit a logistic model well (ie the decrease in percapita growth rate is not linear.) Do this for a variety of food-number settings and see which combination shows the greatest departure from what you expect from the logistic model. Include your Excel analysis with your homework as lastname_firstname_week3.xls
8. Before submitting your NetLogo program write comments for all your procedures explaining what each “non obvious” line does.