Biomorphs

In this week's lab we will follow the example of Dawkin's biomorphs in the *Bind Watchmaker* to evolve digital creatures. The idea is to use NetLogo to build a geometric structure that is built following a simple set of rules. The rules then become a genetic code for the creatures which are subject to random mutation from one generation to the next. The selection pressure comes from you. You choose which mutated form to breed and which to let die. In the end you wander through biomorph land until a pleasing form emerges.

Making Branching Creatures

The basic form we will chose for our creatures is a branching structure similar to the branching trees we created in fall quarter. The main idea is to create a turtle that plays the role of a seed. It moves forward, with the pen down, hatches two new turtles that play the role of branches, and then dies. The branches then repeat the process a certain number of times until a tree-like structure is formed. First give yourself lots of space by changing the world-view to be 250 by 250 with patch size 1. So that we all start with the same code, I've included two procedures that work together to implement this idea. The first creates the "seed" turtle wherever the mouse is clicked and then calls the "branch" procedure, which contains the instructions to make the branching structure. You may want to add a `clear-screen` procedure to the interface too. For this code to run you will need to make sliders for all the variables except the `edge` and `angle` variables, which should be defined as `turtles-own` variables at the top of the procedures tab. When creating sliders choose sensible values for the max and increment. In particular don't allow the `number` variable to go larger than 10, as this will create a structure that has too many branches, and may crash NetLogo. Also let `edge-factor` range between 0.1 and 2 in steps and 0.1 and let `angle-factor` range between 1 and 5 in steps of 0.01.

```
to go
     if mouse-down? [
          create-custom-turtles 1 [       ; this is the seed turtle
               setxy mouse-xcor mouse-ycor
               pendown
               set color brown
               set edge start-edge    ;set the initial branch length
               set angle start-angle ;set the initial branching angle
               ]
          branch
          ]
end

to branch
     repeat number [                 ; number of branching levels
          ask turtles [
               fd edge
               set edge edge * edge-factor     ; make branches smaller
               set angle angle * angle-factor  ; change the angle
               hatch 1 [rt angle]              ; hatch right branch
               hatch 1 [lt angle]              ; hatch left branch
               die ]                           ; main trunk stops

          ]
      ask turtles[die]                         ; drawing is finished
end
```

You should get reasonable tree-like structures when the angle is not too large and the angle-factor is close to 1. However, when you let the angle-factor be larger you should see some structures that look nothing like a tree.

With the current set-up you a have five variables you can adjust to vary the trees. You can think of these variables as the genes of the creatures, and changing the variables corresponds to sweeping through biomorph space in five dimensions. Play around with the variables for a while to see what values lead to interesting trees, and what increments give rise to meaningful, but not too distant changes in form.

Before we start the next part of the lab let's try modifying our procedure so that it has a few more genes to vary. One thing we might consider doing is breaking the symmetry of our creatures.

Symmetry breaking

To break symmetry we will allow left branches and right branches to branch off at different angles. First replace the slider for `start-angle` with two new sliders, one for a left angle called `start-langle` and the other for a right angle called `start-rangle`. Similarly replace the angle-factor slider with two new sliders `rangle-factor` and `langle-factor`. Now in the procedures tab add new `turtles-own` variables called `right-angle` and `left-angle` (you can remove the `angle` variable, you won't need it.).

You must also modify the procedures. First, in your `go` procedure remove the line

```
set angle start-angle
```

and replace it with the two lines

```
set right-angle start-rangle
set left-angle start-langle
```

Similarly in the `branch` procedure replace the line

```
set angle angle * angle-factor
```

With

```
set right-angle right-angle * rangle-factor
set left-angle left-angle * langle-factor
```

And finally make the newly hatched branches turn in the appropriate amount

```
hatch 1 [rt right-angle]
hatch 1 [lt left-angle]
```

You now have a net increase of two genes for a total of seven. Try it out. You may not think these asymmetric creatures are very pretty. Most organisms retain bilateral symmetry so let's try to preserve this. We can do this making sure the main right branch is a mirror image of the main left branch (all the sub-branches are free to branch at strange angles).

To do this we need to treat the first level of branching a bit differently. When the seed turtle first

branches it should do so symmetrically – ie by the same angle on each side. At all subsequent levels of branching we can allow the branching angles to be different, but we must make sure that turtles on the main right branch do the opposite of what turtles on the main left branch do.

It will be helpful at this point to introduce the idea of `breeds`. We can distinguish between turtles which have different types of behaviour by giving them a `breed` name. In this case we have `seeds` and `branches`. To define these `breeds` we add the line

```
breeds [seeds branches]
```

to the start of the procedures tab. Then instead of defining `turtles-own` variables we define `seeds-own` and `branches-own` variables. Make those changes.

We also need to distinguish between the turtles on the main right branch and those on the main left branch. We do this with a new `branches-own` variable called `right?` which can take the values `true` or `false`. Add this variable to the line where you define the `branches-own` variable. The following code now implements the suggested changes

```
to branch
   ask seeds [     ;the seed turtle makes the main branches
      hatch-branches 1
              [ rt right-angle
                set right? true ]   ; main right branch
      hatch-branches 1
             [lt right-angle     ; make the main left branch the same
               set right? false ]; angle as the main right for symmetry
      repeat number [
            ask branches [
                  set right-angle right-angle * rangle-factor
                  set left-angle left-angle * langle-factor
                  set edge edge * edge-factor
                  fd edge
                  ifelse right?  ; distinguish left and right branches
                        [ hatch 1 [rt right-angle]  ; make the right
                          hatch 1 [lt left-angle] ] ; and left branches
                        [ hatch 1 [rt left-angle]   ; mirror images of
                          hatch 1 [lt right-angle]] ; each other
                  die
                  ]
            ]
       ask branches [die] ; remove the final branch turtles
        die ]              ; remove the initial seed turtle
end
```

Before this will work you also need to change the `create-custom-turtles` command in the go procedure to `create-custom-seeds` .

You should now be able to create creatures that have bilateral symmetry, but which are somewhat more diverse than before. We are almost ready to let these creatures evolve by human selection. First lets add a final gene that changes the `pen-size` by some factor at each level. Add a slider for `pen-factor` and then add a line in your `branch` procedure that sets `pen-size to pen-size * pen-factor`.

Homework
At the moment you select your creatures by changing sliders. This is not how evolution does it. The variables assigned by the sliders are the genes of our creatures. We want to have some mechanism for these genes to mutate when the creatures "reproduce". We will start by creating a litter of nine creatures, each of which is a slightly mutated version of its neighbour. Then we will select the "fittest" creature, and this one will be the one that will reproduce. Its offspring will all be displayed on the screen and then we select again. By a gradual process of mutation and selection we will see the creatures evolve.

1. First create a setup procedure which clears the screen and then creates 9 `seeds`, each with the heading set to 0. Within the `setup` procedure after creating the turtles you should also place the turtles evenly around the screen. It is probably best to write a new procedure called `plant-seeds` and call it from the `setup` procedure. Leave turtle 0 at the center of the screen and then In the `plant-seeds` procedure individually place the turtles 1 through 8 in a square around the center using the setxy command. Make good use of space. You may find it helpful to reference the global variables `screen-edge-x` and `screen-edge-y` when placing the turtles. Call the `plant-seeds` procedure in your `setup` procedure.

2. We want to give each of the seeds a genetic code. We can do this by defining a `seeds-own` variable called `genome`. This will be as a variable that is a list that contains the values you assigned to the different variable names using the sliders. As an example suppose I only have the variables `number`, `edge` and `edge-factor` as genes, and I want `number` to have the value 4, `edge` to have the value 10 and `edge-factor` to have the value 0.9, then I would define my `genome` variable as follows

   ```
   set genome [4 10 0.9]
   ```

   Add a line like this to the part of your setup procedure where you create the 9 seeds. You will have a longer list. At the moment each seed will have the same genetic-code, that will change after we mutate them. (Note, the order of the list is important so I would add a comment line in your code below reminding yourself which variable each value refers to.)

3. In order allow your seeds to mutate define a `globals` variable called `mutation`. This will be a list that defines the increments by which each of the genes can change in each step. The order of this list should be the same as the order for the genome list. For example if I want `number` to change by 1, and `edge` to change by 2 and `edge-factor` to change by 0.1 each step I would define mutation as follows

   ```
   set mutation [1 2 0.1]
   ```

   Add a line like this, with your complete list of mutation values, to your setup procedure in the line **before** you create any seeds. When the choosing the mutation rates, make sure there is scope for small, but still significant change. Use your original program to try out what values make sense.
4. Now you need a mutate procedure that mutates the genome of turtles1 through 8. (turtle 0 will be left un-mutated) . Since we have 8 genes lets have turtle 1 be the one with gene 1 mutated and turtle 2 have gene 2 mutate and so on. To mutate a gene we have to choose at random to add or subtract the corresponding value of `mutation` for the gene we want to mutate. The syntax for changing values in lists is a little awkward so I'll include the code for the `mutate` procedure below. The `mutate` procedure must be called by each one of the eight seeds who will mutate. I suggest calling the `mutate` procedure each time you place a

seed in the in the `plant-seeds` procedure. Here is the code:

```
to mutate
   let i who - 1
   ifelse (random 2) = 1
     [set  genome replace-item i genome ((item i genome) + (item i mutation))]
     [if (item i genome) > (item i mutation-rate)
       [set genome replace-item i genome ((item i genome) - (item i mutation))
        ]]
end
```

5. Now that your seeds are planted and mutated they are ready to grow. For the seeds to grow they need to know what the values of their genes are. Although you have given these values to the `genome` list you haven't yet assigned these values to the variable names you have use in your branch procedure. Define an `assign-values` procedure where you assign each variable the appropriate value in the `genome`. For example if the variable `number` corresponds to the first entry in the `genome` list and `edge` was the second value in the `genome` list then you would have the following lines in the `assign-values` procedure.

   ```
   set number (item 0 genome)
   set edge (item 1 genome)
   ```

   Note the typical programming convention that the first item in a list is called item 0 and the second item is called item 1. Complete this procedure, and then called it in the `setup` procedure immediately after the `plant-seeds` procedure. You should now delete all your sliders from the interface. You don't need them anymore.
6. Now the final line in your setup procedure should be to call your `branch` procedure. It should all work wonderfully now, once you remove any bugs. Actually it would be helpful if you also remove the last die command in your branch procedure, where you removed the seed. We will want to keep the seeds around for the final part of the program.
7. You are nearing the end of this lab. You now need to modify your go procedure so that you can use the mouse to select your favorite creature. First delete the lines in the go procedure where new seeds are created. We don't need to do this anymore since we create our seeds in the setup procedure. Now define a new `globals` variable called `fittest` at the top of the procedures tab. This variable is the creature we will select with the mouse. In the go procedure set `fittest` to be the seed closest to the mouse when it is pressed down. You'll need to use the `distancexy` reporter and the `min-one-of` reporter for this. I'll let you look these up in the manual.
8. Now in the go procedure have all seeds set their `genome` to the `genome` of the `fittest` seed, then clear the screen of all the drawings, mutate the turtles by calling the `plant-seeds` procedure, and then create your newly evolved creatures by calling `branch.`
9. You should now be able to select your favorite turtle with a mouse click. When you do, you should see your selection in the middle. If your mutation rates are about right you should be able to weave your way through biomorph space, selecting creatures without much care for the actual genes. If you want to keep track of the actual gene's you should add monitors showing each of the gene's for turtle 0, who will always be the latest selected.
10. Although we have no room left for genes, you might want to add color to your creatures by linking the color to one of the other genes. For example in any hatch command you add a line: `set color right-angle` and see what it does.
11. Now have some fun. Evolve three new creatures, and include the genetic code for these creatures in the information tab of your interface. Then Save your model, giving it the name "lastname_firstname_biomorph.nlogo". When you are finished the homework drop this file in the dropbox folder.