Turtle Genome

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 which follows a simple set of rules. The rules then become a genome 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 phenotype space until a pleasing form emerges.

Recursion in Logo

We will start with the basic structure: Load the file classiclogo.nlogo in the handouts directory. Once you press setup the interface will have the settings of the Classic Logo, with one turtle at the origin on a white screen ready to do your bidding. The turtle is hidden, and the pen is down, so this like a command driven drawing tool. Lets try a few simple forms before coming to the logo-creatures. With the command center in turtle mode type:

```
repeat 4 [fd 30 lt 90]
```

Wow! A square! Ok so that shape isn't going to have much scope for evolving. How about a spiral? For that write the procedure:

```
to spiral
      let n 20
      let step 100
      while [n > 0] [
            fd step
            lt 45
            set step step * 0.8
            set n n - 1
            ]
end
```

The procedure has a while loop, which repeatedly executes a series of commands until the condition `n > 0` is no longer true. See if you can understand what the program does. Press setup to reset the screen then type spiral in the command center. What you get is called an equiangular spiral. This spiral is not our creature, but you may be able to see that the spiral has several parameters that influence its form. One is n which controls how many steps are taken, step is the length of the initial step, 45 is the angle rotated after each step, and 0.8 is a the scale factor by which the step length decreases each time through the loop. These parameters can be considered to be the spiral's genome. You could make sliders for all of these parameters and see how this alters the spiral. Don't. Instead lets look at a recursive way of drawing a spiral. In recursion we avoid using a while loop and instead have the procedure call itself. Modify the above procedure as follows:

```
to spiral [ step n  ]
      ifelse n > 0  [
            fd step
            lt 45
            spiral (step * 0.8)  (n -1)
            [stop]
end
```

At first sight this recursive definition looks a bit funny. Here is how to understand it. The procedure `spiral` now takes two arguments: `step` and `n`. You must specify values for these when you run the procedure. As with the previous procedure the basic element of the spiral is to move forward and then turn. Then you repeat the process with a slightly larger step. By calling itself with new arguments the spiral repeats those two simple commands. Notice that n is reduced by one each time. Eventually it will be reduced to zero and which point the procedure stops.

Press `setup` again and then type:

```
spiral 100 20
```

and confirm that this procedure does the same thing as previously. Ok, We are almost ready to make our creatures. But first let's work on the interface.

It will be handy to use the mouse to place our creature where we want them so make a `go` procedure as follows

```
to go
      if mouse-down? [
                  ask turtles [
                  setxy mouse-xcor mouse-ycor
                  set heading 0
                  spiral step-length step-number
            ]
      ]
end
```

Make sliders for `step-length` and `step-number`. Don't let n be much larger than 20. Actually you could also replace the angle 45 with a variable parameter and similarly for the scale-factor 1.2 and then add sliders for them. In this way you can quickly try out different parameter values in between mouse clicks. Now let's make our creatures.

Branching Creatures

The basic form we will chose for our creatures is a branching structure that will be recursively defined, in a similar way to the spiral. As a first stab try

```
to branch [ step n ]
            ifelse n > 0  [
                  fd step
                  lt angle
                  branch (step * scale)  (n - 1)
                  rt (angle + angle)
                  branch (step * scale)  (n - 1)
                  lt angle
                  bk step ]
                  [stop]
end
```

The main part of this procedure is to step forward, turn left and draw a branch, turn right and draw a branch, turn back to the original heading and then step back to where you started from. This last part is important. In order for recursion to work, you need to return the turtle to its

starting point before ending the procedure. What makes this interesting is that the branch that is drawn is identical to the main tree but scaled down, and one level of complexity less. The parameter n is the recursive level of this procedure and as you will see, Logo chokes if n is much larger than 15. This is because Logo has to draw 2^n branches, which grows very fast with n. In fact a more reasonable range would be between 1 and 10. This is especially important for the next change we will make. Replace the spiral command with the new `branch` command in your `go` procedure. Try it out.

As it is this branching structure, with only 4 parameter values, doesn't have a large enough genetic code to make it interesting for growing creatures. We can add another parameter quite easily. We could also allow our tree to have segments, by steping forward a number of times to repeat the pattern. Add a slider for `segment-number`, ranging from 1 to 6. Here is a modification that should work.

```
to branch [ step  n ]
          ifelse n > 0  [
               repeat segment-number [
                    fd step
                    lt angle
                    branch (step * scale)  (n - 1)
                    rt (angle + angle)
                    branch (step * scale)   (n - 1)
                    lt angle
                    ]
               bk (step * segment-number) ]
               [stop]
end
```

Notice that we now step back "`segment-number`" of times to return to our starting point. We now have 5 genes in our genome. See what kinds of creatures you can create. I bet most of these look like trees. No surprise here! In fact our creatures have far too much symmetry. In all developing creatures, symmetry is broken, so we must make one last change to our genetic code.

Symmetry breaking

To break symmetry we will allow left branches and right branches to branch off at different angles. As a first try let's define two new angles a left angle called `langle` and a right angle called `rangle`. Add sliders for these ranging from 0 to 180. Then modify your branching program as follows

```
to branch [ step  n ]
          ifelse n > 0  [
               repeat segment-number [
                    fd step
                    lt langle
                    branch (step * scale)  (n - 1)
                    rt (langle + rangle)
                    branch (step * scale)   (n - 1)
                    lt rangle
                    ]
               bk (step * segment-number) ]
               [stop]
end
```

Make sure you understand the changes and try out your new 6 code genome. You can quickly get a garbled mess. Most organisms retain bilateral symmetry so let's try to preserve this. We will still allow left and right branches to differ, but in a way that retains reflection symmetry.

Change your branch program back to the symmetric version, except have it call two new procedures `lbranch` and `rbranch`

```
to branch [ step  n ]
           ifelse n > 0  [
                repeat segment-number [
                     fd step
                     lt angle
                     lbranch (step * scale)  (n - 1)
                     rt (angle + angle)
                     rbranch (step * scale)   (n - 1)
                     lt angle
                     ]
                bk (step * segment-number) ]
                [stop]
end
```

The new procedures will be almost identical to the main branch procedure accept, the angles that are turned will now different

```
to lbranch [ step  n ]
           ifelse n > 0  [
                repeat segment-number [
                     fd step
                     lt langle
                     lbranch (step * scale)  (n - 1)
                     rt (langle + rangle)
                     rbranch (step * scale)   (n - 1)
                     lt rangle
                     ]
                bk (step * segment-number) ]
                [stop]
end

to rbranch [ step  n ]
           ifelse n > 0  [
                repeat segment-number [
                     fd step
                     lt rangle
                     lbranch (step * scale)  (n - 1)
                     rt (langle + rangle)
                     rbranch (step * scale)   (n - 1)
                     lt langle
                     ]
                bk (step * segment-number) ]
                [stop]
end
```

The only difference is that in the left branch we turn left first and in the right branch we turn right first.

We now have seven genes for our genome. These are: `step-number`, `segment-number`, `step-length` `scale`, `angle`, `langle` and `rangle`. There are some more genes we can add to our genome, but we'll save that for homework..

Homework

At the moment you select your creatures by changing sliders. This is not how evolution does it. Instead we start with a creature with a particular genome. It reproduces with random mutations, and one of the mutants is then selected as the "fittest" creature, it reproduces and so on. After several steps the ideal creature emerges. We want to do this. I recommend saving your program with a new name for these changes, as you may want to refer to the old one later for reference.

1. First create a `turtles-own` variable called `genome`. This will be a list that contains the values you assign to the different gene-types via the sliders. (The values are the genes, and the variable names are the genotype.) At this point you will need to remove all your sliders from the interface. Also define a `globals` variable called `mutation-rate`. This will be a list that defines how by how much you will mutate each gene during each time step.
2. Modify your `setup` procedure to assign the appropriate values to the `genome` and `mutation-rate` lists. A list is made by listing the numbers in order in square brackets. For example:
   `set primes [2 3 5 7].`
   The order is important. Note the genome must be defined in the `create-custom-turtles` command, while the `mutation-rate` list should be defined outside of this command. I will leave you to choose the values for these two lists, however, as a rule of thumb, let your initial genome be a simple shape (eg. `segment-number` 1 and `step-number` 2). Also, when choosing mutation rates, make sure there is scope for small, but still significant change. Use your original program to try out what mutation rates make sense. Make sure the gene order in your `mutation-rate` list corresponds to the gene order in your `genome`. Add a comment to your program to let people know what each number refers to.
3. Change your `setup` procedure to create 9 turtles. Initially each will be setup up with the same `genome`. Now write a `place-turtles` procedure that places each turtle in its own location around the screen. I suggest leaving turtle 0 at the center and then arranging turtles 1 through 8 in a square around it. Make good use of space. Before growing your turtles, you need to mutate all but turtle 0.
4. Write a `mutate` procedure which chooses a random gene from a turtles `genome` and changes it by adding or subtracting the corresponding `mutation-rate` for that gene (you need to be able to mutate in both directions. You may worry about what happens if a gene has a value zero or negative. If you are worried about this include an `ifelse` condition in your procedure to prevent this from happening). To refer to an item in a list use the `item` reporter, and to change an item in a list use the replace-item command. Be aware that the first item in a list is actually referred to as item 0. For example

   ```
   item 1 [2 3 5 7]
   ```

   returns the value 3 since 3 is item 1 in the list. If I want to define a new list with the 3 replaced by a 4 for some reason I'd write

   ```
   set new-list replace-item 1 [2 3 5 7] 4
   ```

   My new list would be [2 4 5 7]
5. Now you can use your `mutate` procedure. In the `setup` procedure ask all but turtle 0 to mutate.

6. Now you are ready to grow your creatures. Define a `grow` procedure which first assigns all the gene-values to their genotype names. Then ask the turtles to branch as you do in your `go` procedure. Use your `grow` procedure after your `mutate` procedure in the `setup`. You should see nine simple branching creatures on the screen, each with a slight variation from turtle 0 which is in the middle.
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 define a new `globals` variable called `fittest`. Then set `fittest` to be the turtle closest to the mouse when it is pressed down. You'll need to use the `distancexy` reporter for this.
8. Now set all turtles to have the same `genome` as the fittest turtle, paint all patches white to erase the screen, `mutate` all but turtle 0 and then `grow` your turtles again.
9. You should now be able to select your favorite turtle with a mouse click. When you do, you should see new mutants grow. Provided you have set your mutation rates about right you should be able to weave your way through phenotype space, selecting creatures without much care for the actual genes. If you want to keep track of the actual gene's you could add monitors showing each of the gene's for turtle 0, who will always be the latest selected.
10. Now that you've played around with that. Let's extend the genome a bit. First of all there is no rule that says you have to scale right and left branches the same amount. Add `rscale` and a `lscale` genes, and modify your `rbranch` and `lbranch` procedures appropriately so that you retain bilateral symmetry. You may want to try this out first in your original program, so that you can make sure you get it right.
11. You can also scale the angles, so that they change as you go to a deeper level in your tree. This is a bit harder to do. Create `rangle-scale` and `langle-scale` genes. In order to use them you will need to give your `lbranch` and `rbranch` procedures two new arguments, `langle` and `rangle`. Then when you use these commands include these two new arguments scaled appropriately. You will need to add these to all places where you refer to these procedures. One consequence of doing this is that now Netlogo will complain about having defined `langle` and `rangle` already as part of the genome. You will have to rename these variables in the `turtles-own` line and when you assigned values to them in the `grow` procedure. Perhaps you could call them `start-langle` and `start-rangle`.
12. Now think about how you might add a color gene to your creatures. Perhaps you could change color according to some color-factor at each level, or perhaps color, is not controlled by a new gene, but is just linked to another gene – maybe the step size or the angle. Try to liven up your creatures with some color-scale in some way.
13. Now have some fun. Evolve three new creatures, and include the genetic code for these creatures in the information tab of your interface.
14. Save your model, giving it the name “lastname_firstname_week_5.nlogo”. When you are finished the homework drop this file in the dropbox folder. Before submitting your NetLogo program write comments for all your procedures explaining what each “non obvious” line does.