# Lab 11: Recursion and the Fractal Tree

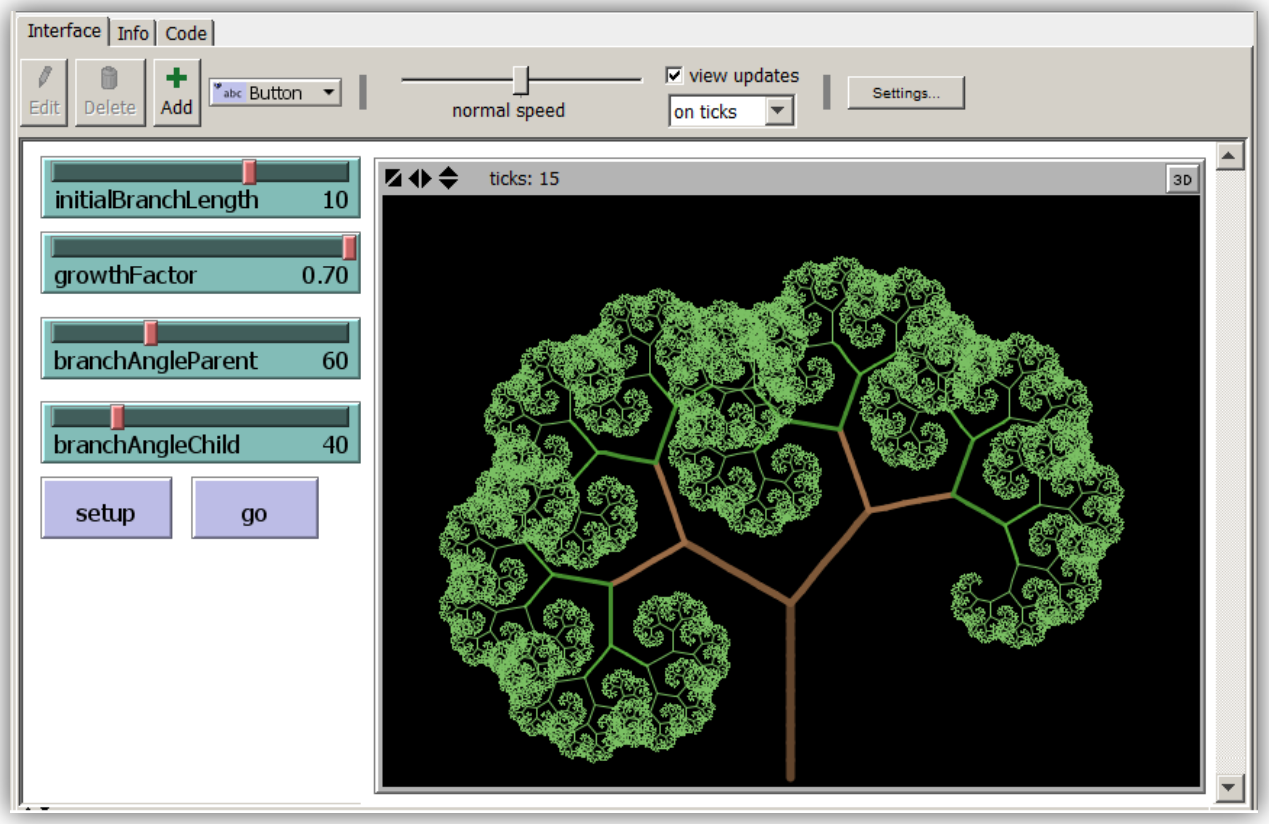



Fig 1: Program in its final stage of development.
This version continues to call itself recursively until the branch length is less than 0.05 patches.

## Model Overview:

In this Netlogo model, the goal is to author a ***recursive program*** that draws a tree-like ***fractal pattern***. In this project, we define a recursive program as one that contains a procedure that calls itself, but that somehow does so without creating an infinite loop. The two images below are famous examples of recursion that occurs outside of computer science: a mirror reflecting itself, and M.C. Escher image of a pair of hands drawing each other. For more details on recursion, see this week's videos.

Fig2: "*Holding infinity in the palm of my hand*", by Samizdat – the title being a reference to a line in the poem "To See a World..." by William Blake.

Fig3: "*Drawing Hands*" is a lithograph by the Dutch artist M. C. Escher first printed in January 1948.

## Getting Started:

Before trying to create the complete model, first start a new Netlogo program with the sliders and buttons shown. Use the world settings shown below with the origin set to the bottom edge, the min-pxcor set to -23, the max-pxcor set to 23 and the max-pycor set to 33.

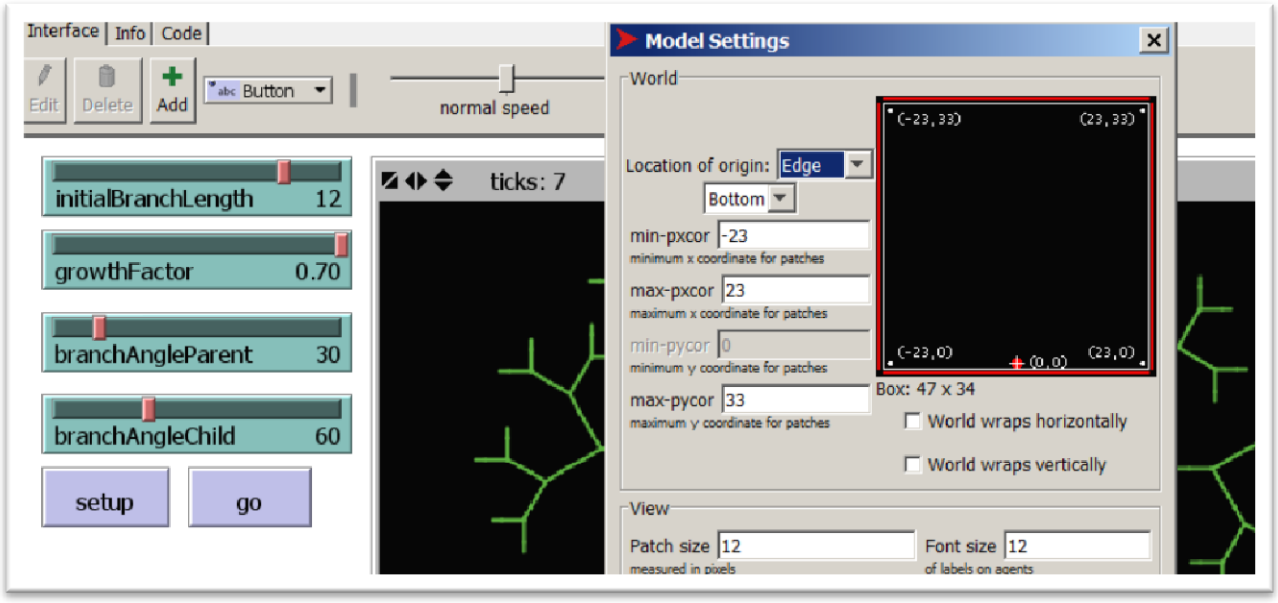



Fig 4

THE UNIVERSITY of NEW MEXICO

## Running with the Basic Structure:

In my version, I have three procedures. The two for which buttons are shown on the interface: "setup" and "go", and a third that I called "grow". In my version, this "grow" procedure is the recursive procedure. That is, someplace within "grow", there is a call to "grow". The basic structure of the program is shown below:

```
 1) to setup
 2)    ;; clear-all, create turtle, ...
 3) end
 4)
 5) to go
 6)   grow initialBranchLength
 7) end
 8)
 9)
10) to grow [branchLength]
12)  ;; do stuff (may be multiple lines of code)
13)
14)   if (branchLength >= 1)  ;;end recursion when <1
15)   [
17)     ask turtles
18)     [
19)        ;; do stuff (may be multiple lines of code)
20)     ]
21)
22)     tick ;; update the display before recersive call to "grow"
23)
24)     grow (branchLength * growthFactor)
25)   ]
26) end
```

This structure contains each of the key parts found in most recursive programs:

1) **Something that gets the recursion started:** In the example above, this is done with the call to "grow" in "go" where "grow" is given the starting input value from the slider. Each call to "go" draws one branch that has a length equal to this input value.

2) **The recursive call:** On line 24, grow calls itself. In general, recursive calls avoid creating infinite loops by passing input that is different from the input given.

3) **The End Condition:** Recursive programs need to have some way of ending. In this case, each time grow is called, it is called with a smaller and smaller input branch length. The recursion only continues when the branch length is greater than or equal to 1 patch (line 14).

THE UNIVERSITY of NEW MEXICO

## "Do Stuff":

The screen capture below shows a run of the model before it is fully complete. In my version of "setup", I create just one turtle, but with the "do stuff", within the "**ask turtles**" of "**grow**", I use the Netlogo "hatch" command to create a new turtle. Then, when "grow" is called on line 24, "**ask turtles**" gets called with both the parent I created in setup and with the child hatched in the last call to "grow". Since "hatch" is in "**ask turtles**", both the parent and the child hatch a net turtle. Thus, one becomes 2, 2 becomes 4, 4 become 8, 8 becomes 16, 16 becomes 32, ..... Of course, if this does not stop, then it will not be long before there are too many turtles for Netlogo to handle.

Since there is a picture being drawn, "do stuff" needs to do more than make baby turtles. In particular, it must draw a branch with length equal to the input value of "grow" and it must make a turn or two.

To understand how to make this work, think about the basic structure of Netlogo's "hatch":

```
;;Any changes to a turtle (color, heading, location, etc) made before "hatch" are
;;   inherited by (copied into) the new turtle.

hatch 1
[
   ;;Any changes to a turtle made within a hatch block only effect the
   ;;      new turtle that was just hatched.
]
;;Any changes to a turtle made after a hatch block only effect the parent, not the
;;     turtle just hatched.
```

In the example shown below, the turtle that starts at the bottom center and moves straight up the center to near the top is the turtle that was created in "setup". Observe the following about this grandparent of all turtles:

1) It never makes a turn.
2) In each call to "grow" it moves forward one branch length.
3) In each call to "grow" its branch length is shorter than it was in the previous call.
4) In each call to "grow" it has one child who branches off to its right.
5) It stops when its branch length is smaller than one patch (which in my version is 12 pixels).

Observe that facts true for the great grandparent of all turtles are also true for each of the turtles in each of the following generations!!!

Each child turtle starts its life with a heading equal to the slider value of "branchAngleChild" to the right of the heading of its parent, and each child's first branch has a length that is the same as the length its parent will branch on its next call to "grow". Otherwise, the life of a child is the same as the live of its parent.

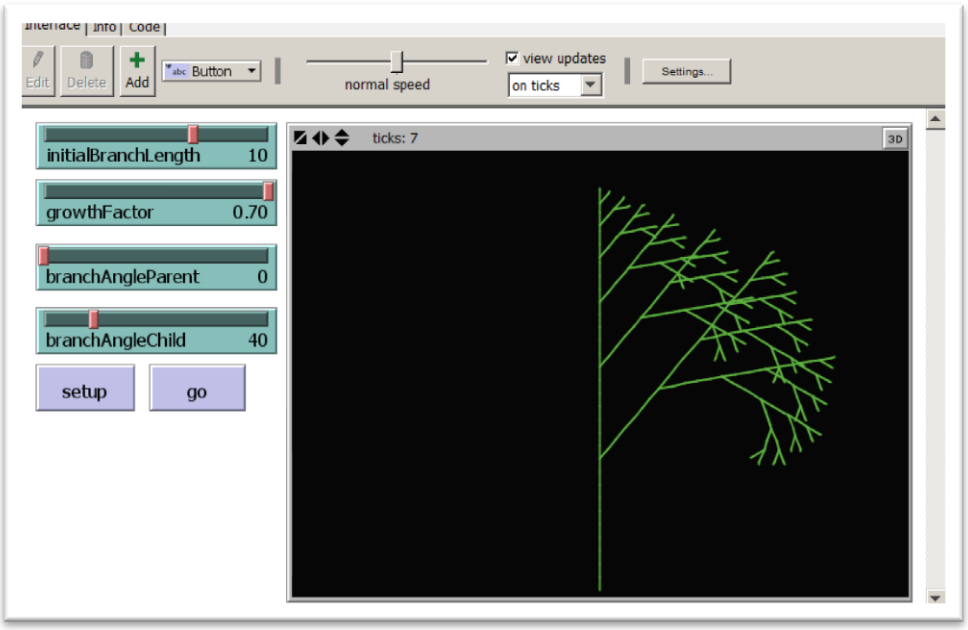



Fig 5

At this point, you should stop reading and get coding. With what is given above, you should be able to create a model that works up through the capture above.

## Running With More Generations:

After you get the model above running correctly, change the recursion stop condition so that the model runs for more generations. That is, the version above only branches when the branch length is at least one patch. Change this to half a patch, a quarter patch, a tenth patch, ...

When should you stop?

Well, with each branch, the distance gets smaller, so eventually, the branches will be smaller than a pixel and different branches will blur together – not looking very good. Also, with each branch, the number of turtles doubles, so each extra branch will take twice the runtime as the previous branch.

## Implementing the Branch Angle of the Parent:

In the version above, the parent always went straight. In the final version, the parent should only move straight when the slider value of "branchAngleParent" equals 0. When this slider is set to a different value, the parent should branch left by the specified angle.

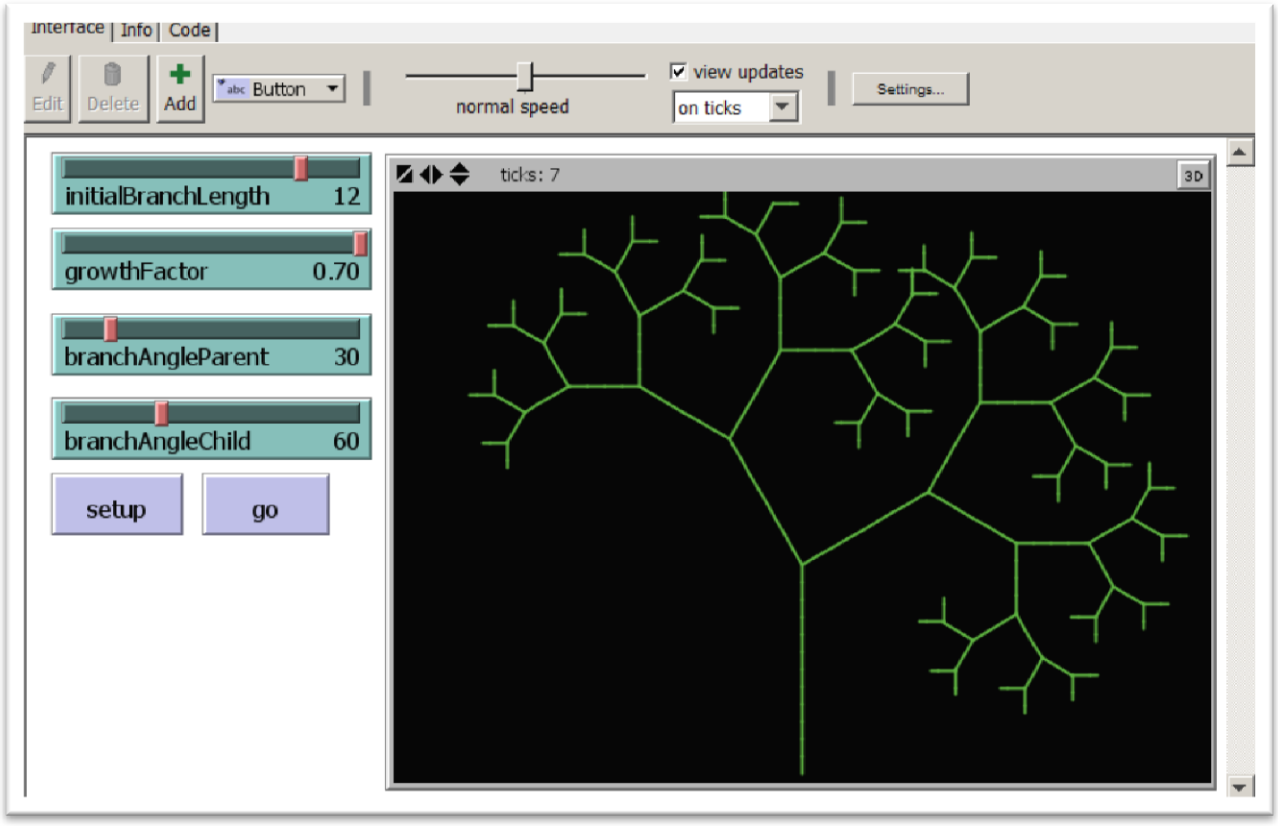



Fig 6

## Implementing Changing Branch Thickness:

After all of the above parts are working, implement the branch thickness that is thickest for the "trunk" and gets smaller as the tree branches. There are a number of different ways this can be done. If you like, you may choose to create an additional slider to allow the user to make adjustments to the way line thickness is used. For example, selecting the thickness of the starting line and/or the rate at which the line gets thinner.

In my implementation, I created a global variable, "**`BrarchThickness`**". In "setup", I set this to a value of 7 and also in setup, where I create the great grandfather turtle, I set its pen-size to "**`BrarchThickness`**". Then, each time I call "grow" as

long as thickness is greater than 1, I subtract 1 from it. Finally, within "ask turtles" of "grow", I set every turtle to the current value of thickness.

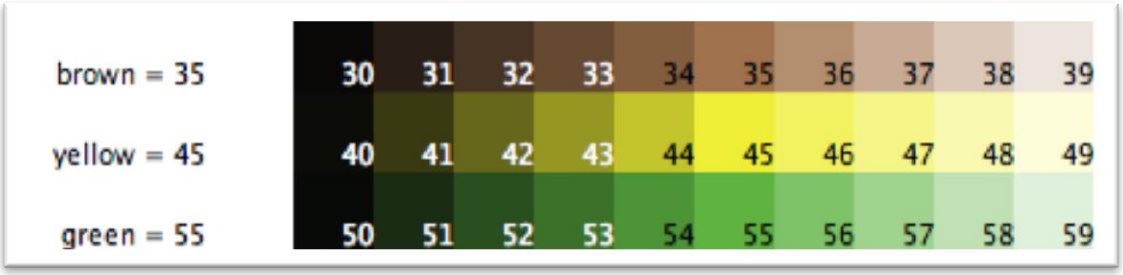
## Implementing Changing Branch Color:

The requirements are:

1) All lines of the same length (drawn in the same generation) have the same color.

2) The final 14 or so generation image has at least three different line colors.

As with implementing the changing branch thickness, there are many different ways you may choose to implement these requirements. What I did in my version is to use the value of the global "**`BrarchThickness`**" variable I created for the branch thickness together with some knowledge of how NetLogo represents colors. In particular, Netlogo's default color table is given in the on-line documentation. In particular, I wanted to use browns and greens, so the part of the table I used was:




A segment of Netlogo's Default Color Table

In my implementation, when the branch thickness is greater than 3, I wanted a shade of brown – the thicker the branch, the darker the brown. To get this, I set the color to `(39 - BranchThickness)`. This worked for me since:

1) For me, the largest value I use for **`BranchThickness`** is 7 and 39 – 7 equals 32 – a nicely dark shade of brown.

2) I only used this equation when **`BranchThickness`** was greater than 3, so the lightest color this ever picks is 39 – 4 or 35, which is lighter than the dark brown, but not too light.

When **`BranchThickness`** is not greater then 3, I use a similar equation to set the color to one of the shades of green.

THE UNIVERSITY of NEW MEXICO

## Grading Rubric [20 points total]:

**[A: 1 point]:** Submit Netlogo source code named: `W11.`*firstname*`.`*lastname*`.nlogo`.

**[B: 1 point]:** The first few lines of your code tab are comments including your name, the date, your school, and the assignment name.

**[C: 1 point]:** The code in the code tab of your program is appropriately documented with "inline comments".

**[D: 4 points]:** Clicking "setup" and "go" with the setting shown in Fig 5, produces, on tick #7, the pattern shown in Fig 5. NOTE: you get full points for this section with or without branch color and/or thickness changes. You also get full points for this section if your drawing stops at tick #7 or if it continues on to stop at a later tick count.

**[E: 4 points]:** Clicking "setup" and "go" with the setting shown in Fig 6, produces, on tick #7, the pattern shown in Fig 6. NOTE: you get full points for this section with or without branch color and/or thickness changes. You also get full points for this section if your drawing stops at tick #7 or if it continues on to stop at a later tick count.

**[F: 3 points]:** Clicking "setup" and "go" with the setting shown in Fig 1, produces, on tick #15, the pattern shown in Fig 1. NOTE: you get full points for this section with or without branch color and/or thickness changes. You also get full points for this section if your drawing stops at tick #15, #16, or #17 – your choice.

**[G: 3 points]:** Clicking "setup" and "go" with the setting shown in Fig 1, produces a treelike pattern with at least 3 different branch thicknesses for different generations – which may or may not match the choice of thicknesses shown in figure 1.

**[H: 3 points]:** Clicking "setup" and "go" with the setting shown in Fig 1, produces a treelike pattern with at least 3 different branch colors for different generations – which may or may not match the choice of colors shown in figure 1.