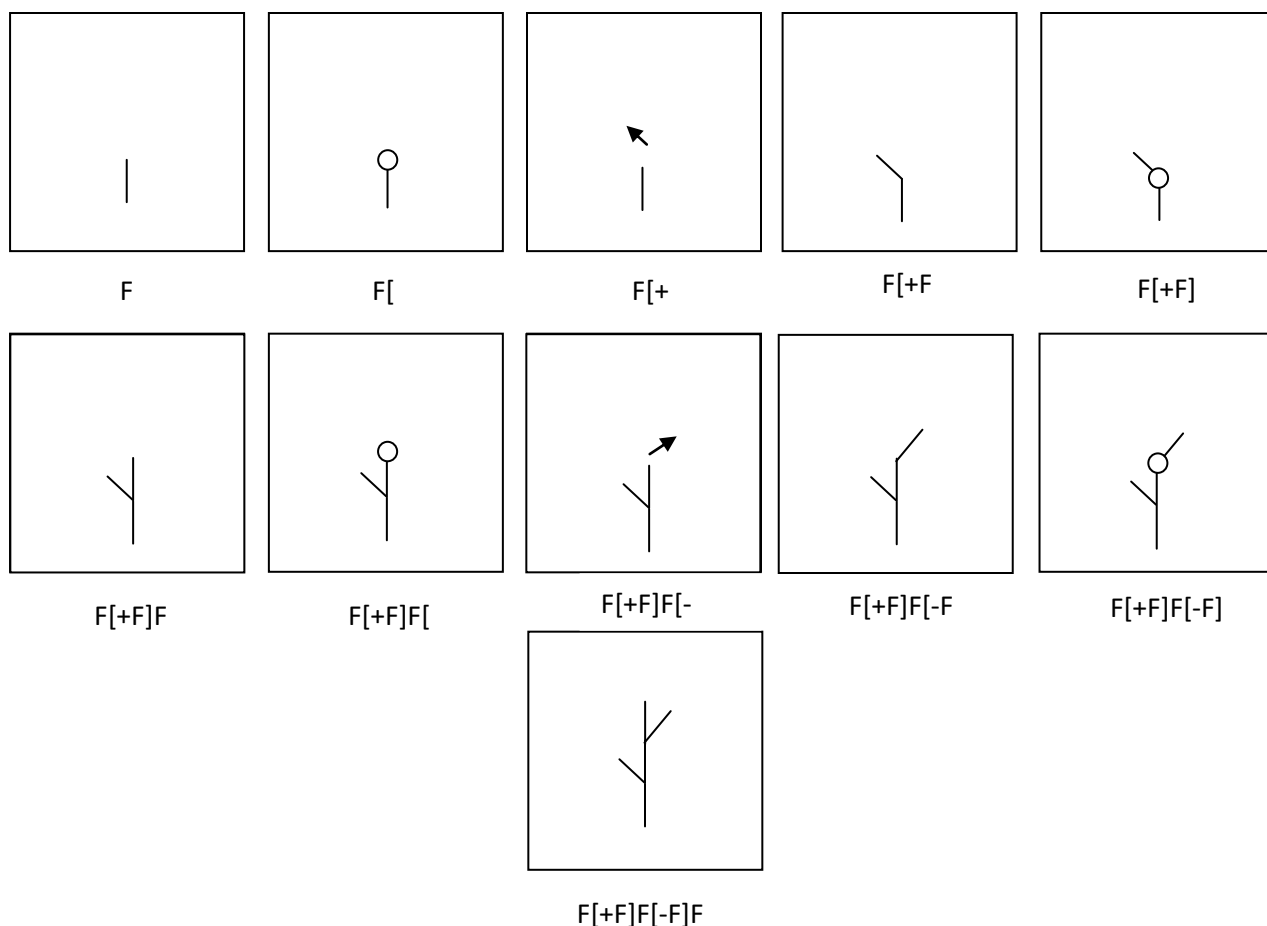


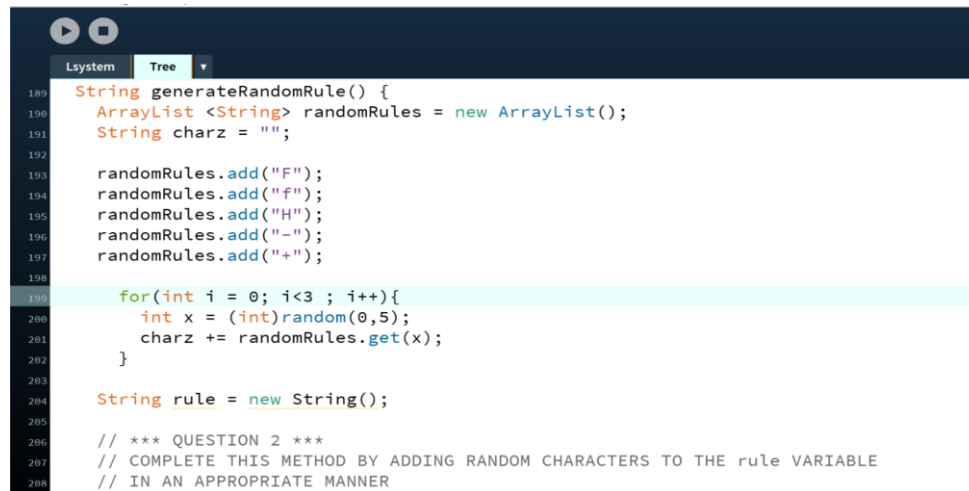
CO1112 Creative Computing I: Image, Sound and Motion (Course Work 2)

1a) Within the string “F[+F]F[-F]F” located the tree sketch, each char has a specific characteristic that actually indicates where the turtle method will move in order to draw the output (tree). The rule ‘F’, draws a line from the centre of the most bottom part of the screen (and slightly upwards), whilst ‘[’ saves the current position and ‘]’ goes back to the previously saved position. The ‘+’ rule changes the direction/angle of the line to be drawn (with the ‘F’ rule) to $-(25.7/180.0)*\pi$, meaning that the next line will be drawn to the left. On the other hand, when a ‘-’ rule is presented, then the point will rotate to $(25.7/180.0)*\pi$ and draw the next line to the right, using the ‘F’ rule.



The diagrams above illustrate the steps the turtle method will take when drawing the sketch. First, a line is drawn and the position of the last point of the line is saved. After, the direction of the next line is directed to the left (-) and after the line is drawn, the sketch returns to the saved point. A horizontal line is drawn right after, where a line is drawn to the right direction, where we can start seeing the creation of the branch-like sketch. Finally, another straight horizontal line is drawn, finishing the rule sequence F[+F]F[-F]F.

2a) For the first part of question 2, I was required to create a random rule each time the code ran. The rule required was to be of length 3. By creating an Array-List, and placing each individual rule into the said Array-List, I was able to easily generate 30 random string rules of length 3. Below is a snippet of the code, followed by the respective output of 30 random rule strings of length 3.

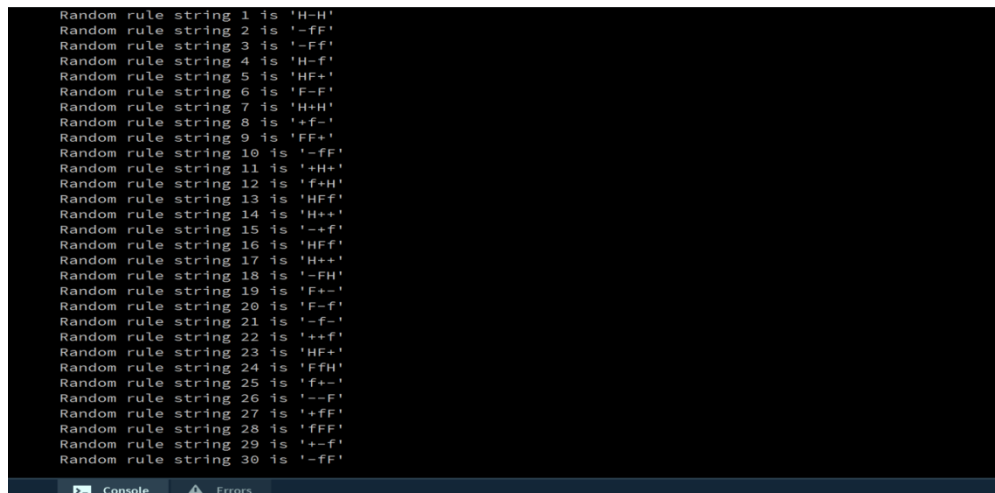


```

189 String generateRandomRule() {
190     ArrayList<String> randomRules = new ArrayList();
191     String charz = "";
192
193     randomRules.add("F");
194     randomRules.add("f");
195     randomRules.add("H");
196     randomRules.add("-");
197     randomRules.add("+");
198
199     for(int i = 0; i<3 ; i++){
200         int x = (int)random(0,5);
201         charz += randomRules.get(x);
202     }
203
204     String rule = new String();
205
206     // *** QUESTION 2 ***
207     // COMPLETE THIS METHOD BY ADDING RANDOM CHARACTERS TO THE rule VARIABLE
208     // IN AN APPROPRIATE MANNER

```

- Fig 1. Code for 30 Random Rules of Size 3



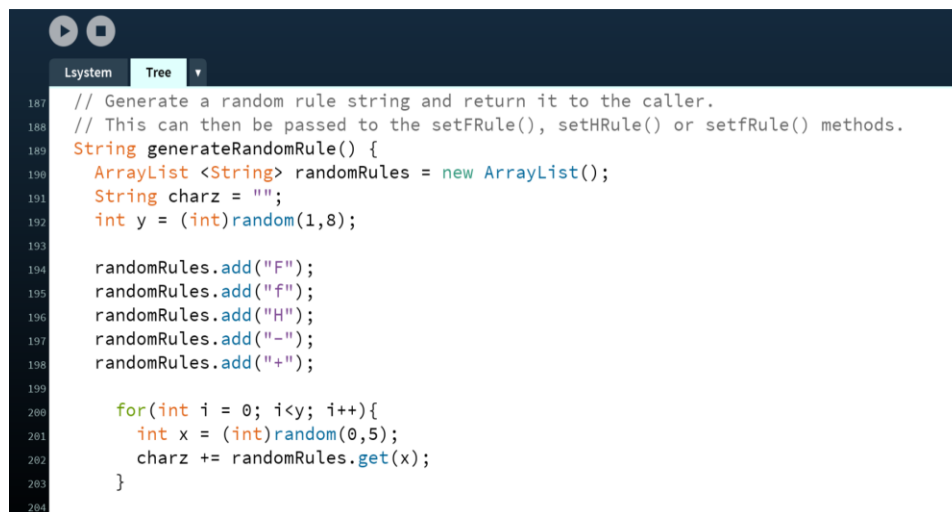
```

Random rule string 1 is 'H-H'
Random rule string 2 is '-FF'
Random rule string 3 is '-ff'
Random rule string 4 is 'H-f'
Random rule string 5 is 'HF+'
Random rule string 6 is 'F-F'
Random rule string 7 is 'H+H'
Random rule string 8 is '+f-'
Random rule string 9 is 'FF+'
Random rule string 10 is '-fF'
Random rule string 11 is '+H+'
Random rule string 12 is 'f+H'
Random rule string 13 is 'HFF'
Random rule string 14 is 'H++'
Random rule string 15 is '-+f'
Random rule string 16 is 'HFF'
Random rule string 17 is 'H++'
Random rule string 18 is '-FH'
Random rule string 19 is 'F+-'
Random rule string 20 is 'F-f'
Random rule string 21 is '-f-'
Random rule string 22 is '+++'
Random rule string 23 is 'HF+'
Random rule string 24 is 'FfH'
Random rule string 25 is 'f+-'
Random rule string 26 is '---F'
Random rule string 27 is '+fF'
Random rule string 28 is 'FFF'
Random rule string 29 is '+-f'
Random rule string 30 is '-fF'

```

- Fig 2. Output for 30 Random Rules of Size 3

2b) The second part of question 2 was completed without any problems as it was being created. In order to implement another element of randomness within the random string generated by adding random lengths (from 1-8), I first declared an integer 'y' at the beginning of the generateRandomRule() method. Following this, a for loop was created, and the guard was set to 'y', meaning that the length of the word will be as long as the randomly generated rule string to be later shown as output. When a new string is to be generated, 'y' is reset and another random length is given for the following rule string. The following images depict the code used for this part of question 2, followed by the output of said code.

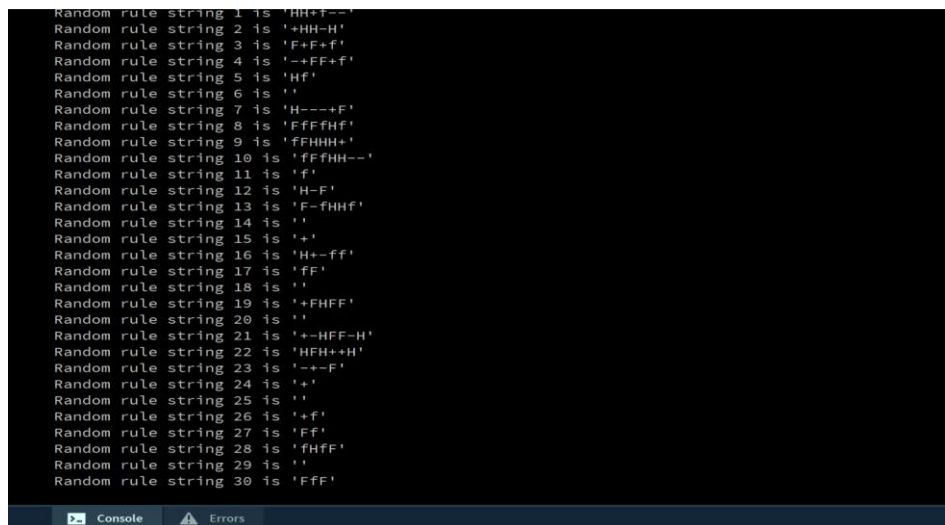


```

187 // Generate a random rule string and return it to the caller.
188 // This can then be passed to the setFRule(), setHRule() or setfRule() methods.
189 String generateRandomRule() {
190     ArrayList<String> randomRules = new ArrayList();
191     String charz = "";
192     int y = (int)random(1,8);
193
194     randomRules.add("F");
195     randomRules.add("f");
196     randomRules.add("H");
197     randomRules.add("-");
198     randomRules.add("+");
199
200     for(int i = 0; i<y; i++){
201         int x = (int)random(0,5);
202         charz += randomRules.get(x);
203     }
204 }

```

- Fig 3. Code for 30 Randomly Generated Strings of Lengths 1-8



```

Random rule string 1 is 'HH+-'
Random rule string 2 is '+HH-H'
Random rule string 3 is 'F+F+f'
Random rule string 4 is '-+FF+f'
Random rule string 5 is 'HF'
Random rule string 6 is ''
Random rule string 7 is 'H---+F'
Random rule string 8 is 'FFFfHf'
Random rule string 9 is 'fFHHH+'
Random rule string 10 is 'FFfHH--'
Random rule string 11 is 'f'
Random rule string 12 is 'H-F'
Random rule string 13 is 'F-fHHF'
Random rule string 14 is ''
Random rule string 15 is '+'
Random rule string 16 is 'H+-ff'
Random rule string 17 is 'ff'
Random rule string 18 is ''
Random rule string 19 is '+FHHF'
Random rule string 20 is ''
Random rule string 21 is '+-HFF-H'
Random rule string 22 is 'HFH++H'
Random rule string 23 is '-+-F'
Random rule string 24 is '+'
Random rule string 25 is ''
Random rule string 26 is '+f'
Random rule string 27 is 'Ff'
Random rule string 28 is 'fHfF'
Random rule string 29 is ''
Random rule string 30 is 'FfF'

```

- Fig 4. Output of 30 Randomly Generated Strings of Lengths 1-8

2c) Finding flaws within this method of acquiring string rules took some further experimentation, leading to following questions to be done before this one. A major issue found within this method was that most of the time, sketches were being left blank (empty) due to the fact that two of the most crucial rules 'F' or 'H' (both of which draw a straight line), were not being generated within the random rule string most of the time. Therefore, by adding the conditional statement below, if the final generated string did not contain one of the two rules, an 'F' rule would be placed in front of the rule string. The image below shows the code for this implementation, and the image of the output of the last part of question 2 shows this code functioning properly.


```

205
206         if(z == randomRules.get(5) && charz.length() <= y-2){
207             charz += z + randomRules.get(x2) + " ";
208         }else if(charz.length() < y){
209             charz += randomRules.get(x2);
210         }
211     }
212
213
214     if(!charz.contains("F") || !charz.contains("H") && charz.charAt(0) != '['){
215
216         charz = "F" + charz;
217     }
218
219     return charz;
220 }
221 }

```

- Fig 5. This Image Shows The Method Used for Implementing The 'F' Rule When Not Found

2d) Adding the rules '[' and ']' randomly within each random rule string, whilst maintaining the lengths of the rules to be from 1-8, was a bit of a challenge. After trying to figure out ways to add both brackets randomly and failing at many different attempts, I decided to play it safe and go for a different alternative. Instead of having both brackets being randomly generated separately, I decided to implement a conditional statement, that if an expected opening bracket ([]) was generated randomly, then another random character from the other rules is generated using a second declared integer 'x2', followed by another closed bracket (]). The length of the current string was also taken into consideration since I did not want the entire rule to extend the length of 8 characters. The images below show the respective code used for this part of question 2, as well and the output.

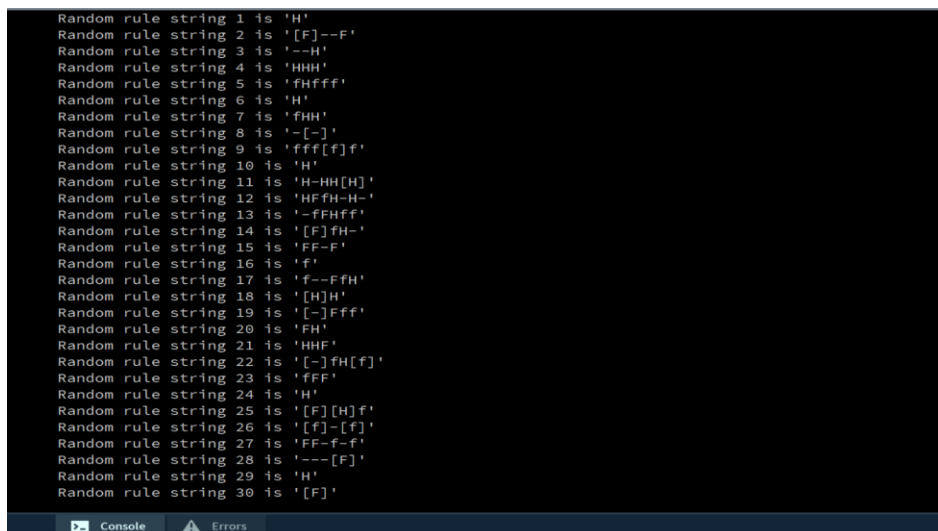


```

190 ArrayList<String> randomRules = new ArrayList();
191 String charz = "";
192 String z = "";
193 int occ1 = 0;
194 int occ2 = 0;
195 int y = (int)random(1,8);
196
197 randomRules.add("F");
198 randomRules.add("f");
199 randomRules.add("H");
200 randomRules.add("-");
201 randomRules.add("+");
202 randomRules.add("[");
203 randomRules.add("]");
204
205 for(int i = 0; i<y; i++){
206     int x = (int)random(0,6);
207     int x2 = (int)random(0,4);
208     z = randomRules.get(x);
209
210     if(z == randomRules.get(5) && charz.length()<=y-2){
211         charz += z + randomRules.get(x2) + "[";
212     }else if(charz.length()<y){
213         charz += randomRules.get(x2);
214     }
215 }
216 return charz;

```

- Fig 6. Code Used for Generating 30 Random Rule Strings With The Additional Characters



```

Random rule string 1 is 'H'
Random rule string 2 is '[F]--F'
Random rule string 3 is '--H'
Random rule string 4 is 'HHH'
Random rule string 5 is 'fHfff'
Random rule string 6 is 'H'
Random rule string 7 is 'fHH'
Random rule string 8 is '[-]'
Random rule string 9 is 'fff[f]f'
Random rule string 10 is 'H'
Random rule string 11 is 'H-HH[H]'
Random rule string 12 is 'HFfH-H-'
Random rule string 13 is '-fFHff'
Random rule string 14 is '[F]fH-'
Random rule string 15 is 'FF-F'
Random rule string 16 is 'f'
Random rule string 17 is 'f--FfH'
Random rule string 18 is '[H]H'
Random rule string 19 is '[-]Fff'
Random rule string 20 is 'FH'
Random rule string 21 is 'HHF'
Random rule string 22 is '[-]fH[f]'
Random rule string 23 is 'fff'
Random rule string 24 is 'H'
Random rule string 25 is '[F][H]f'
Random rule string 26 is '[f]-[f]'
Random rule string 27 is 'FF-f-f'
Random rule string 28 is '---[F]'
Random rule string 29 is 'H'
Random rule string 30 is '[F]'

```

- Fig 7. Output for The Generated Random Rule Strings Containing The Additional Characters

3) Printing out the random rule strings lead to different patterns resembling nature-like objects, such as branches and leaves. Some of the outputs however, resulted into very minimal outputs, such as having a single dash in the middle of the sketch, which is not really so interesting, but other than that, some outputs I was personally happy with (in terms of their aesthetic appearance). In order to save the outputs, I declared a Boolean value “checkSave” in the beginning of the code, when the code runs through question3setup(), checkSave will become true, activating the conditional statement used in the draw() method, which saves one sketch at a time.

The images below show the code for the final version of question3setup(), along with modifications to other parts of the code to be able to save outputs, as well as the outputs generated by the code itself.

```

29 // Any drawing should be done here!
30 void draw() {
31
32     String setName = "Lsystem-";
33     tree.draw();
34
35     if(checkSave == true){
36         saveFrame(setName + ".jpg");
37     }
38
39 }

```

-Fig 8. Code Within The draw() Method, Used to Draw The Draw and Save The Sketch Produced.

```

59 // Specific setup for Question 3
60 void question3setup() {
61     checkSave = true;
62     String r = tree.generateRandomRule();
63     println("Using F rule '"+r+"'");
64
65     tree.setFRule(r);
66     tree.setX(300); // start the drawing in the centre of the screen
67     tree.setY(300);
68     tree.calculateState();
69
70 }
29 // Any drawing should be done here!
30 void draw() {
31
32     String setName = "Lsystem-";
33     tree.draw();
34
35     if(checkSave == true){
36         saveFrame(setName + ".jpg");
37     }
38
39 }

```

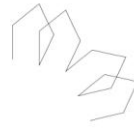
-Fig 8. Code Within The question3setup() Method, Used to Generate a Random Rule String, Setting "checkSave" to True



- Lsystem-0.jpg



- Lsystem-1.jpg



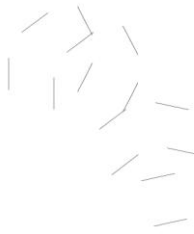
- Lsystem-2.jpg



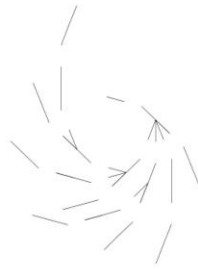
- Lsystem-3.jpg



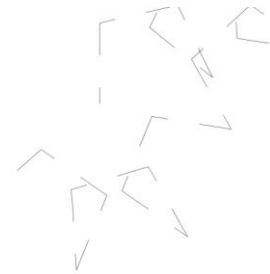
- Lsystem-4.jpg



- Lsystem-5.jpg



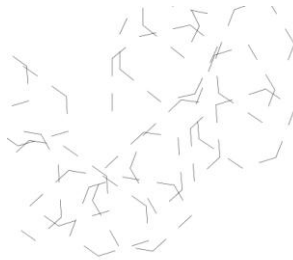
- Lsystem-6.jpg



- Lsystem-7.jpg



- Lsystem-8.jpg



- Lsystem-9.jpg



- Lsystem-10.jpg



- Lsystem-11.jpg

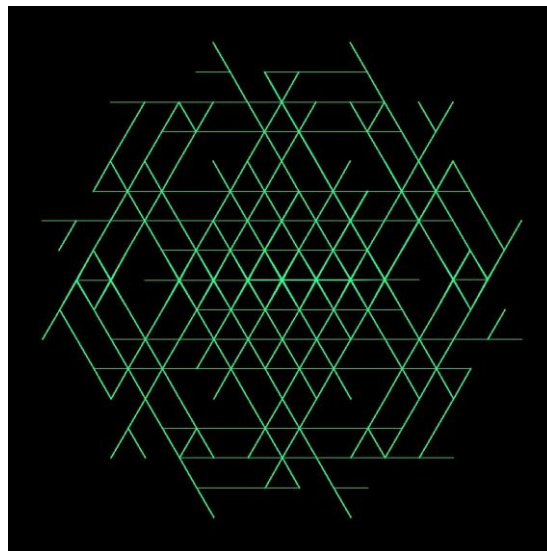
4) Genetic algorithms work in the similar biological process of human genetic inheritance, where some traits of the offspring are acquired from the parents, or distant ancestors. Such biological phenomena are being used in the world of computing to enhance performance/efficiency and to create tools which are revolutionizing the technological industry, (Carr, 2014).

Using such techniques in the case of our L-system sketch, one can implement a method which starts off with an initial parent rule string (which has a set of given rules that are either generated randomly or hard coded). The method `String mutate()` will generate an array of children, sharing similar properties to the first tree, and said children will be sketched next to their parent tree. Each child will share the initial branch starting point, as to avoid complete distortion, possibly providing children which look nothing like a tree, ruining future generations. The offspring of the parent, will share similar rules

but could have random enhancements in their respective scale factors (change in size of branches per sub-branch), changes in the number of sub-branches, and different rotations for every branch. Implementing certain genetic rules such as, making the number of branches of each side equal, ensures that the trees generated have a better chance of being a stronger generated tree. In addition, one could also implement an algorithm which, when a new genetic trait is generated, that trait has a percentage chance of actually being given to the child. If the said trait happens to be generated, it can only be changed by a certain percentage of the parent's original trait.

The user could be given the option to click on one of the respective children. This would change the initial parent tree with the selected child, generating the said child's new offspring, and sharing similar traits in terms of branch lengths, similar number of sub-branches, and so on. The ranges of such variable traits should be regulated, due to the possibility that the generated offspring will share almost no characteristics of its previously generated ancestor trees.

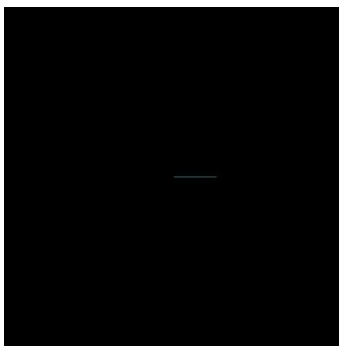
5) Throughout my studies, I have experimented with various mathematical phenomena in order to display a visual representation of these different concepts, ranging from trigonometry, to fractals. When experimenting with the different L-systems, specifically to draw what is known as a Koch Snowflake, which is a very interesting topic found within fractal geometry (Shakiban & Bergstedt, 2000), I found it quite interesting how simple rules represented by a string, could result in such intricately beautiful patterns. Therefore, I decided to work with such concepts to develop my sketch for this part of the coursework.



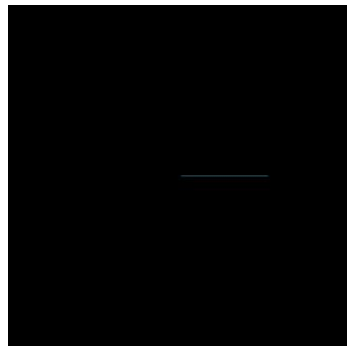
-Fig 9. Saved Image of Final Sketch.

The code for this sketch starts by declaring the initial state of the L-system "F", which draws a short line. Each character, like the "F", has a specific property, for instance, "H" will do the same as rule "F", but gives diversity, allowing for a two-handed rule set to be implemented. The "-" and "+" characters give a change in angle for the next line to be drawn, whereas the "[" and "]" brackets represent the push-matrix and a pop-matrix (as mentioned above, saving current position and returning to the last

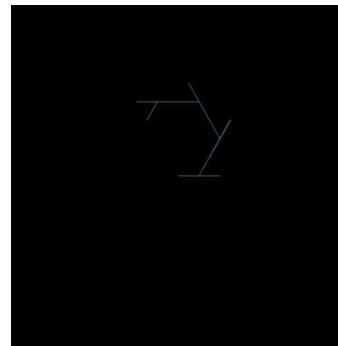
saved position). In addition to these rules, one can find the dedicated F-rule and H-rule ("H-[F]-H" and "F-[H]+F"). Whenever the string contains an F it will be substituted by H-[F]-H, in addition, an H will be substituted by F-[H]+F. For instance, if the current state of the rule is "H-[F]-H" (which would be the case with the first iterated rule), the rule to follow this would be "F-[H]+F- [H-[F]-H]- F-[H]+F", and this would continue to iterate until the sketch is complete. The images below show the sequence at which each rule is incremented, leading to the final sketch. The sketch makes use of two turtle method calls, in order to create the geometric design simultaneously. The turtle method works in a similar step by step fashion to the LOGO educational programming language, which makes simple steps or iterations depending on the command given.



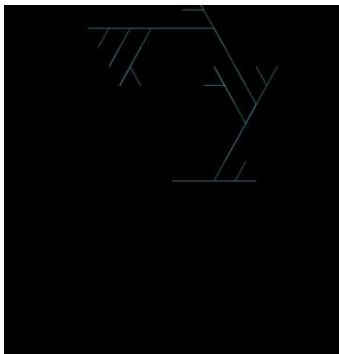
-Sequence 0 (Initial stage)



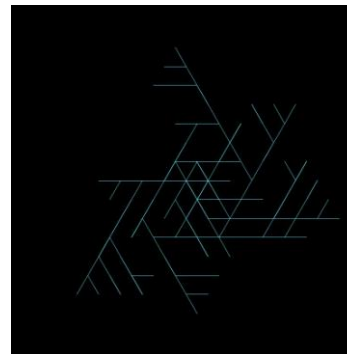
-Sequence 1



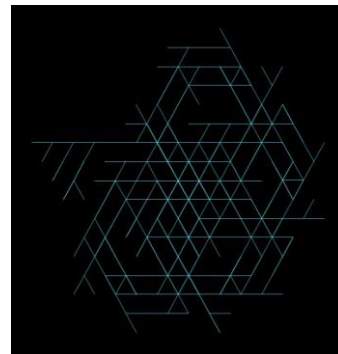
-Sequence 2



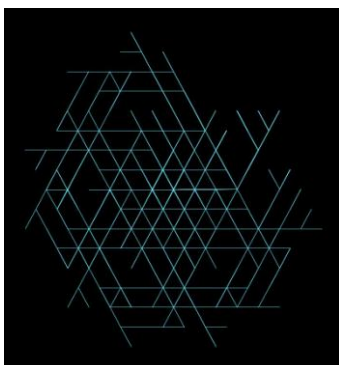
-Sequence 3



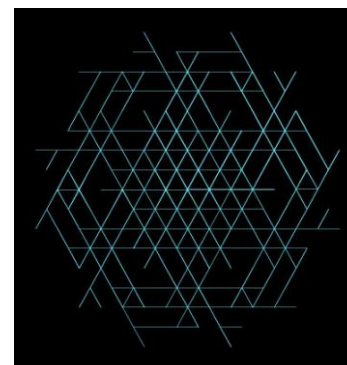
-Sequence 4



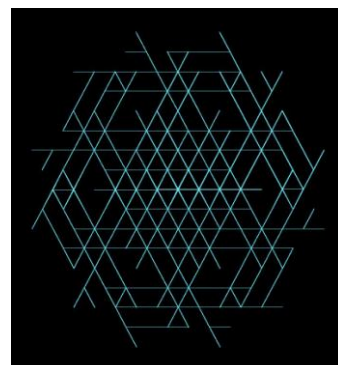
-Sequence 5



-Sequence 6



-Sequence 7



-Sequence 8

The user is given the option to either start the looping sketch (animation), by pressing the space-bar or the enter key, this creates the final sketch with a randomly generated rgb (red, green, blue) values, in order to give a different colour with every newly generated design. The background is set to $-r$, $-g$, and $-b$ so that there is contrast between the sketch and the background. Apart from this option, the user can choose to save the current state of the sketch by pressing “s” or “S”, reset the sketch (with a new colour) by pressing “r” or “R”, and exit the sketch by pressing “e” or “E”. The code for such features can be seen hereunder.

```

18
19 void draw(){
20   background(-r,-g,-b); //background is set to -ve values of rgb in order to always have
21                           //and the background
22   stroke(r,g,b); //sets the stroke colour to the random rgb values generated
23   translate (width/2,height/2); //will set the line at the centre of the screen
24   turtle(state); //the turtle method will be given the current value of the state
25
26   translate (0,0); //will set the line at the beginning of the previous line
27   turtle(state); //the turtle method will be given the current value of the state
28
29   if(key == ' ' || key == ENTER){
30     state = substitute(state); //when the spacebar or ENTER-key are pressed, the state v
31                               // substitute method eventually producing the new rule upc
32     //saveFrame("sketchSequence"+saveID+".jpg"); was used to save each step of the sket
33     //saveID++; was used to increment the name of each file, to avoid over-writing
34
35   }else if(key == 's' || key == 'S'){
36     saveFrame("savedSketch.jpg"); //saves the current frame of the sketch
37   }
38
39   delay(1000); //delays each movement of the sketch by 1000 milliseconds

```

-fig 10. Code For The draw() Method Containing User Choices s/S and ' '/ENTER

```

98
99 void keyPressed(){
100
101   if(key == 'e' || key == 'E'){ //if e or E is pressed, the sketch will exit
102     exit();
103   }else if(key == 'r' || key == 'R'){ //if r or R is pressed:
104     clear(); // the sketch will clear
105     state = "F"; //the state will return the its natural, initial position
106     r = (int)random(255); //the r value is reset to another generated position
107     g = (int)random(255); //the g value is reset to another generated position
108     b = (int)random(255); //the b value is reset to another generated position
109   }
110 }

```

-fig 11. Code For The keyPressed() Method Containing User Choices e/E and r/R

All in all, I am quite content with the output of my sketch, especially since it has not been very easy on a personal level to get accustomed to how processing works, and how to execute the tasks I had been assigned. With this being said, there could have been adjustments to how the output of the sketch resulted. Implementing a more random approach to the direction of the lines as well as their lengths would be a great possible update to this sketch, but I must be careful to avoid having too much randomness, since if the direction of the lines are too unguided, then the sketch could turn out very distorted, leading to an unpleasant output to say the least. I could have also implemented a more interactive approach to my sketch, by adding a 3-dimensional element to it, making it rotatable on the x, y, and z-axis.

Bibliography

Carr, J. (2014). An Introduction to Genetic Algorithms. 1-4.

Shakiban, C., & Bergstedt, J. E. (2000). Generalized Koch Snowflakes. *Department of Mathematics; University of Saint Thomas Saint Paul, Minnesota 55105* , 1-4.