Frankfurt University of Applied Science

Java Project: Fractal Tree

Tran Quang Minh - 1403567

Ngo Dinh Anh Khoa - 1403428

Java Documentation
Frankfurt University of Applied Science
Bachelor of Science in *Computer Science*

March 13, 2023

Abstract

This is an undergraduate Java project report. In computer science, a fractal tree index is a tree data structure that keeps data sorted and allows searches and sequential access simultaneously as a B-tree but with insertions and deletions that are asymptotically faster than a B-tree. The purpose of the project is to create an application that contains a GUI and a drawing tool for Fractal Tree. In this project, JavaFX and its libraries are used as the main methods to program the entire application. We find that using JavaFX and iteration methods is an effective way to create a Fractal Tree drawing tool and it also produces very positive results.

Keywords: JavaFX, iteration

Report's total word count: we expect a maximum of 10,000 words (excluding reference and appendices) and about 40 - 50 pages.

Contents

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Problem statement . . . 2
- 1.3 Aims and objectives . . . 3
- 1.4 Solution approach . . . 3
- 1.5 Organization of the report . . . 3

2 Summary of contributions and achievements **5**
- 2.1 Team Contributions . . . 5
- 2.2 Work Structure . . . 5
- 2.3 Project Idea . . . 6

3 Problem Description **7**
- 3.1 Definition of Fractal Tree . . . 7
- 3.2 Recursion Method . . . 7
- 3.3 Rotation Problem . . . 9

4 Related Work **11**
- 4.1 Algorithm . . . 11
- 4.2 Application . . . 12

5 Proposed Approaches **13**
- 5.1 Menu user interface . . . 13
- 5.2 Drawing user interface . . . 15

6 Implementation Details **21**
- 6.1 Node.java . . . 24
- 6.2 Tree3Controller.java . . . 28

7 Experimental Results and Statistical Tests **35**
- 7.1 Findings and Results . . . 35
- 7.2 Statistical Tests . . . 37
- 7.3 Evaluation . . . 39
- 7.4 Running the file . . . 39

8 Conclusions and Future Work **40**
8.1 Conclusions 40
8.2 Future work 41

List of Figures

1.1 Snowflakes made out of a fractal. 1
1.2 Sierpiński triangle. 2
1.3 Star Fractal. 2

2.1 GitHub Website. 6
2.2 Prototype. 6

3.1 Fractal Tree Example 1. 8
3.2 Fractal Tree Example 2. 9

4.1 Fractal used in computer graphics. 12

5.1 GUI of The Application. 13
5.2 Starting GUI (Menu GUI) after the Start button is selected. 14
5.3 Symmetric Tree GUI. 15
5.4 Spiral Tree GUI. 17
5.5 Binary Tree GUI. 19

6.1 Class Diagram for Game Menu. 22
6.2 Class Diagram for Spiral Tree. 23
6.3 Class Diagram for Symmetric Tree. 23
6.4 Class Diagram for Binary Tree. 24

7.1 Symmetric Tree Test Case. 35
7.2 Spiral Tree Test Case. 36
7.3 Binary Tree Test Case. 36

List of Tables

7.1 Input Data Table 38
7.2 Tree Results Table 38

Chapter 1

Introduction

1.1 Background

The public is more likely to be familiar with fractal art than the mathematical concept when it comes to the word "fractal." It can be difficult to define the mathematical concept formally, even for mathematicians, but key features can be understood with a little mathematical knowledge. A lens or other device that zooms in on digital images to reveal finer, previously invisible, new structure is an easy analogy for explaining "self-similarity". On fractals, however, no new details are revealed; nothing changes and the same pattern repeats over and over, or for some fractals, nearly the same pattern appears repeatedly. People have pondered self-similarity informally, for example, in the infinite regress in parallel mirrors or in the homunculus, the little man inside the head... Fractals differ in that the pattern reproduced must be detailed, Wikipedia(2023).
Here are some examples of fractals:

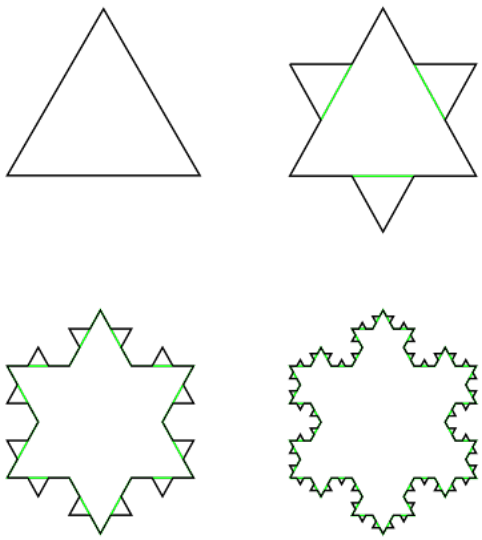

Figure 1.1: Snowflakes made out of a fractal.

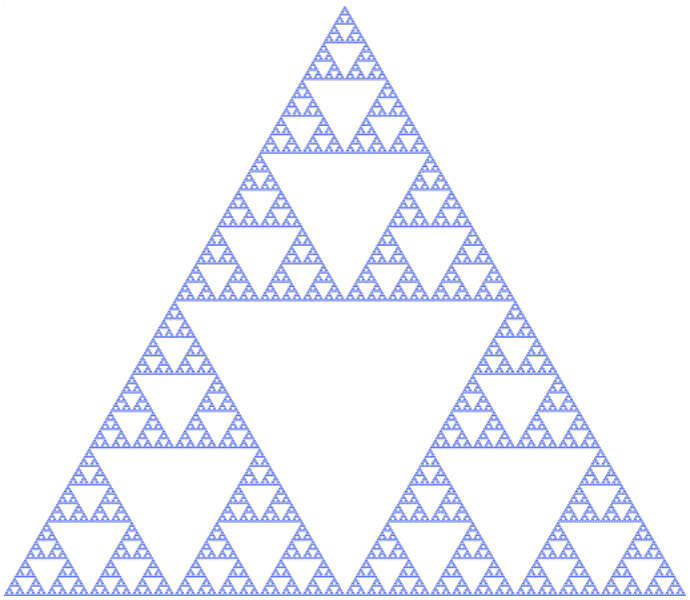

Figure 1.2: Sierpiński triangle.

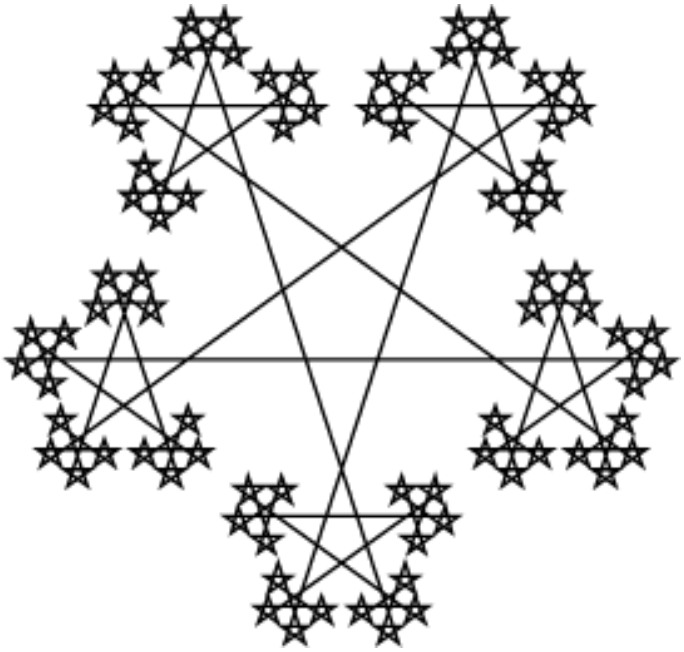

Figure 1.3: Star Fractal.

1.2 Problem statement

In this documentation, we will talk about the Fractal Tree, which is the subset of the fractal. Let's take a look at "Fractal Canopy", which is a type of fractal tree. It is one of the easiest-to-create types of fractals. Each canopy is created by splitting a line segment into two smaller segments at the end (symmetric binary tree) and then splitting the two smaller segments as well, and so on, infinitely (but in our program, we implement the tree

by level). Canopies are distinguished by the angle between concurrent adjacent segments and the ratio between lengths of successive segments. The basic fractal tree must follow these rules:

- The angle between any two neighboring line segments is the same throughout the fractal.
- The ratio of lengths of any two consecutive line segments is constant.
- Points all the way to the end of the smallest line segments are interconnected, which is to say the entire figure is a connected graph.

1.3 Aims and objectives

The aim of our group is to create a complete program containing GUI and Fractal Tree drawing tool. The GUI will have buttons connected to the drawing tool. Moreover, the tool will perform drawing fractal trees with many different levels and angles as well as colors.
The objective of the project is to use JavaFX and SceneBuilder in order to create application menus and Fractal Tree drawing tools.

1.4 Solution approach

To create the fractal tree, these rules must be followed:

- Start at some point and move a certain distance in a certain direction.
- At this point makes a branch. Turn some angle to the right and then repeat the previous step with the distance being decreased.
- Go back and do the same thing for the left side.

A fractal tree is known as a tree that can be created by recursively symmetrical branching. In practice, there are a lot of cases that we have to use recursive functions to solve. Therefore, via fractal tree problems, it is very useful to solve similar recursive problems. Moreover, fractal trees are also related to the space-filling curve problem.

1.5 Organization of the report

This report is organized into eight chapters. Chapter 1 is the introduction of the entire project which will give a glance at the content of the project. Chapter 2 summarizes the contributions and achievements of each member of the project. Chapter 3 is about the problem description which will give information about the definitions and examples, etc. Chapter 4 will point out the related work which contains the related algorithm and applications. Chapter 5 details the proposed approaches such as input/output format, benchmarks, and algorithm pseudocode. Chapter 6 will give information on the

implementation details namely application structure, GUI details, UML diagram, used libraries, and code snippets. Chapter 7 is the part of the experimental results and statistical Tests which contains simulations, uses benchmarks, tables, charts, and evaluation. The final chapter - chapter 8 is about the conclusions and future works.

Chapter 2

Summary of contributions and achievements

2.1 Team Contributions

For drawing trees, Ngo Dinh Anh Khoa has created the GUI for each tree, those GUI are very basic but the functionality is acceptable. Also, Khoa created and managed the Controller for each GUI of the tree, and the Node.java file to work with the tree branch (leaf).
For creating the starting GUI and its effect, Tran Quang Minh managed TreeApp.java and GameViewManager.java files. Minh's job is to link all GUIs together and create the transition between those GUIs.

2.2 Work Structure

Both of us have a good understanding of others' ideas, and ways to solve problems. That is the reason why our communication is favorable. For the repository, both of us agree to use GitHub for the project, and it turns out to be very effective and convenient for us to make adjustments to our project. Besides Github, we often communicate together and exchange ideas directly, or via Facebook and email. Using an application made for GitHub called GithubDesktop, we could easily push and pull code from each other.

Here is our GitHub page: `https://github.com/MinhTran1506/Game.git`

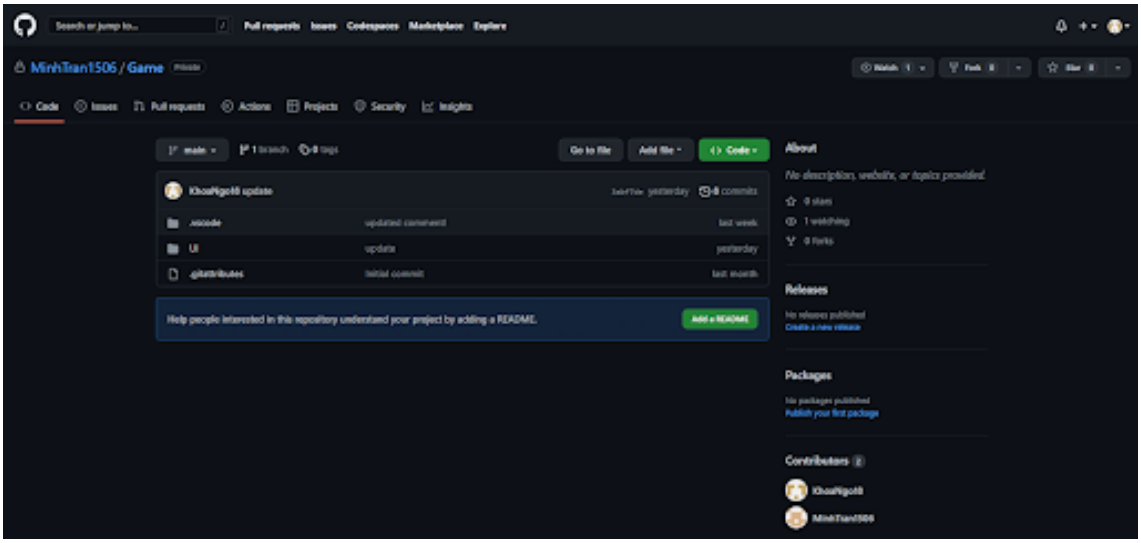

Figure 2.1: GitHub Website.

2.3 Project Idea

For the decision of the project, we consulted ideas from several sources such as Youtube (BroCode, ...), Reddits, Stackoverflow, and Github,... And finally, we agree with each other that Ngo Dinh Anh Khoa will do the implementation of trees, and Tran Quang Minh will do the UI of the application and link the GUIs together. //The first prototype contained a lot of things we wanted to do. After a few weeks, we have to reduce the amount of stuff in the project since there is not enough time to implement everything. Here is our first prototype

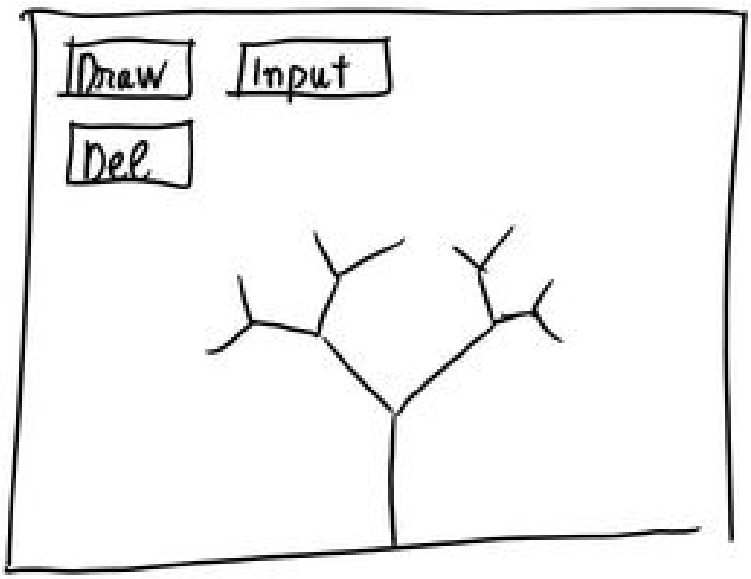

Figure 2.2: Prototype.

Chapter 3

Problem Description

3.1 Definition of Fractal Tree

If we want to understand the "Fractal Tree". Firstly, we need to understand what Fractal is. Definition of **Fractal**: In mathematics, a fractal is a geometric shape containing detailed structure at arbitrarily small scales, usually having a fractal dimension strictly exceeding the topological dimension. Many fractals appear similar at various scales, as illustrated in successive magnifications of the Mandelbrot set. This exhibition of similar patterns at increasingly smaller scales is called self-similarity, also known as expanding symmetry or unfolding symmetry; if this replication is the same at every scale, as in the Menger sponge, the shape is called affine self-similar. Fractal geometry lies within the mathematical branch of measure theory, Wikipedia (2022).

3.2 Recursion Method

A "Fractal Tree" is a subcategory of Fractal. A basic Fractal Tree is based on self-similarity. This means the next branch of the tree follows the same pattern as the parent branch (root). The angle of the branch always stays the same, and the length and the thickness of the branch are decreased by a factor.
The first approach for drawing the tree is to use recursion. Here is the algorithm for drawing the tree using recursion:

```
Function drawingTreeRes(int level){
If (level != 0){
  draw(root);
  create(left, root);
  create(right,root);
  drawingTreeRes(root.left,level--);
  drawingTreeRes(root.right,level--);
    }
}
```

Listing 3.1: Recursion Method Pseudo-code.

There is another way, that is using iteration. The idea is still the same just like recursion, here is the algorithm for iteration:

```
Function drawingTreeIteration(){
    for (int i = branchList.size() - 1; i >=0; i-){
      create(leftBranch, branchList.get(i));
      create(rightBranch, branchList.get(i));
      branchList.add(leftBranch);
      branchList.add(rightBranch);
    }
}
```

Listing 3.2: Recursion Method Pseudo-code 2.

In our project, we choose to use iteration, and for the branch, we create a new data type called Node. The detail for each file will be discussed later in Chapter 6.
Some examples of the fractal tree:

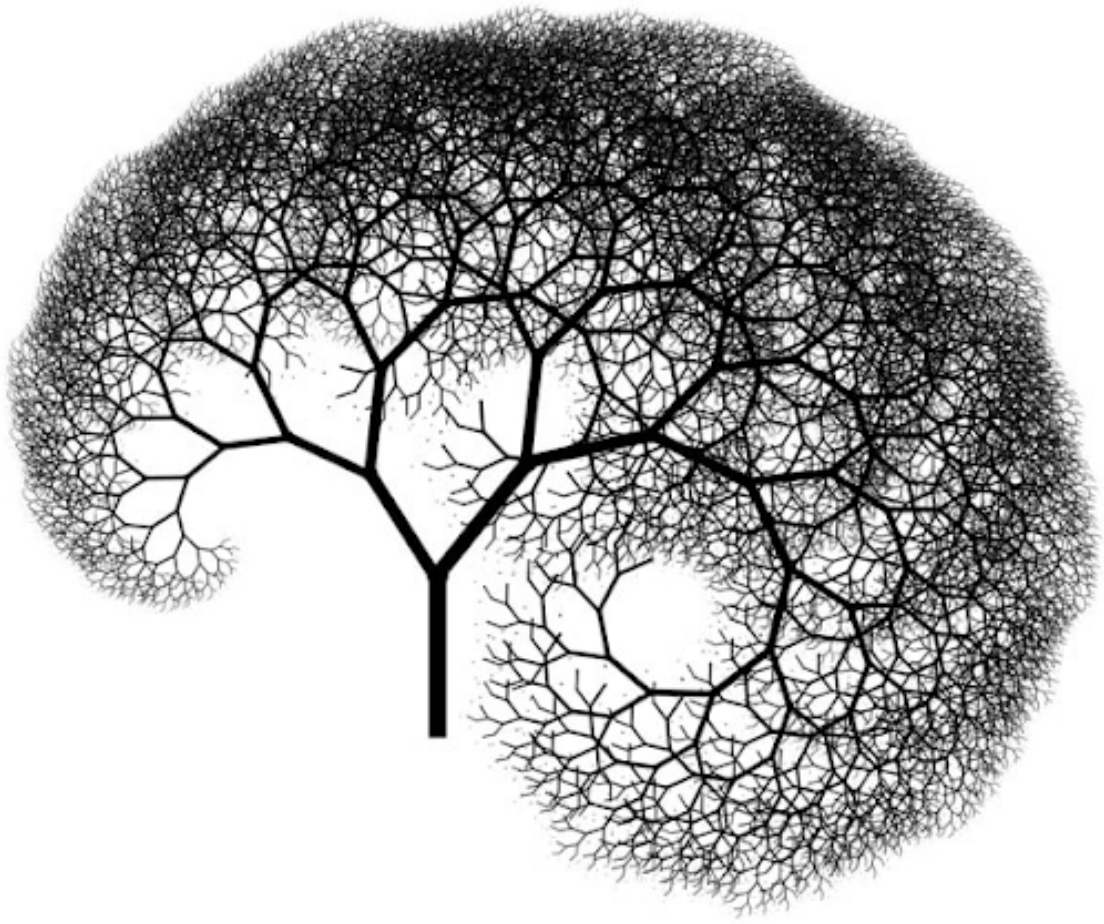

Figure 3.1: Fractal Tree Example 1.

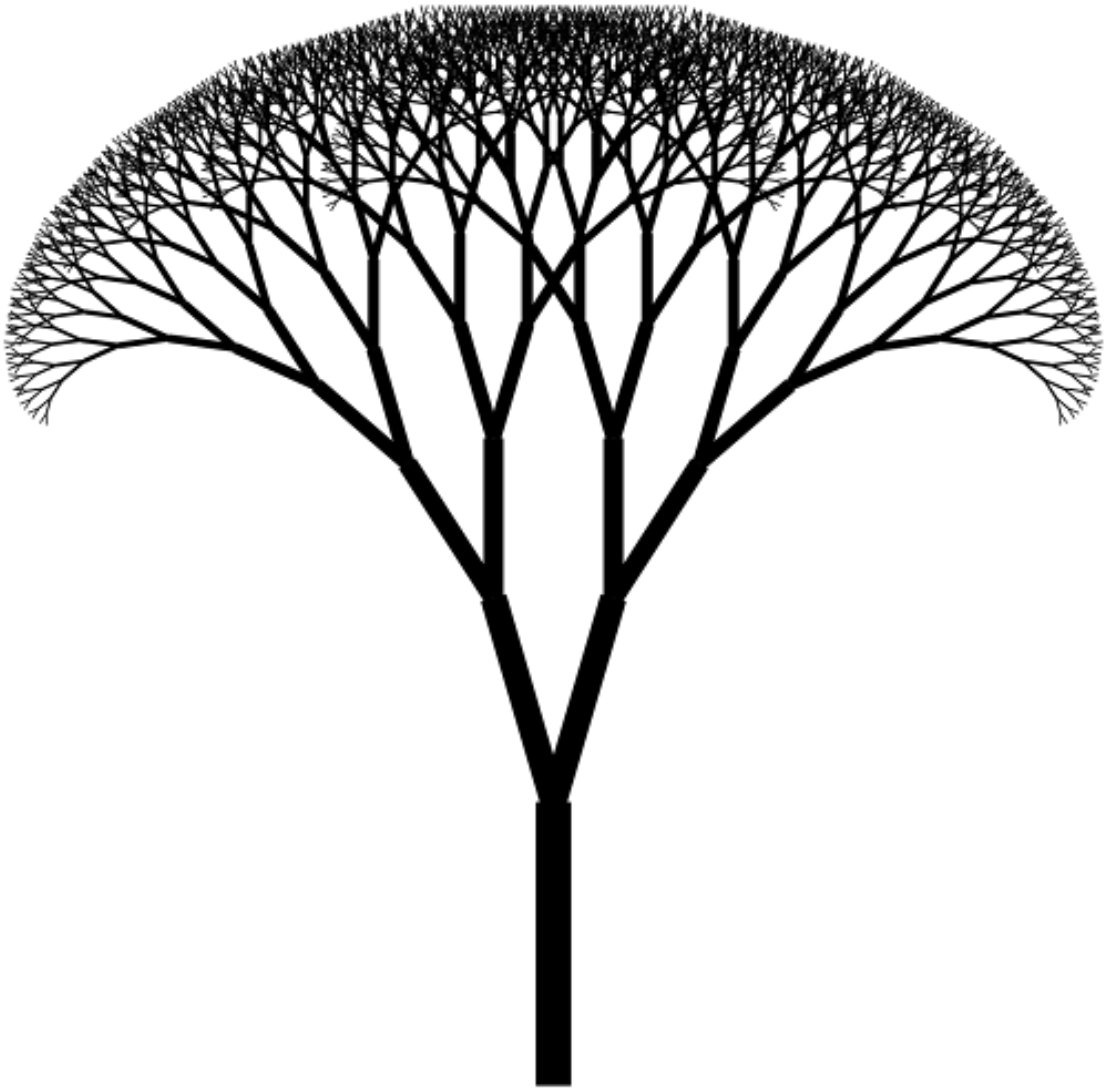

Figure 3.2: Fractal Tree Example 2.

3.3 Rotation Problem

To rotate the branches of the tree, we also need to know about the rotation in coordination. Here is the formal definition of rotation: "In mathematics, a rotation of axes in two dimensions is a mapping from a $xy-Cartesian$ coordinate system to an $x'y'-Cartesian$ coordinate system in which the origin is kept fixed and the x' and y' axes are obtained by rotating the x and y axes counterclockwise through an angle θ. A point P has coordinates (x,y) with respect to the original system and coordinates (x',y') with respect to the new system. In the new coordinate system, point P will appear to have been rotated in the opposite direction, that is, clockwise through the angle. A rotation of axes in more than two dimensions is defined similarly. A rotation of axes is a linear map and a rigid transformation.
The equations defining the transformation in two dimensions, which rotate the xy axes

counterclockwise through an angle "theta" into the $x'y'$ axes, are derived as follows.
In the xy system, let the point P have polar coordinates (r, α). Then, in the $x'y'$ system, P will have polar coordinates $(r, \alpha - \theta)$.
To rotate in the trigonometric formula:

$$x = r\cos\alpha \ (1)$$
$$y = r\sin\alpha \ (2)$$

And using the standard trigonometric formulae for differences, we have:

$$x' = r\cos(\alpha - \theta) = r\cos\alpha\cos\theta + r\sin\alpha\sin\theta \ (3)$$
$$y' = r\sin(\alpha - \theta) = r\sin\alpha\cos\theta - r\cos\alpha\sin\theta \ (4)$$

Substituting equations (1) and (2) into equations (3) and (4), we obtain:

$$x' = x\cos\theta + y\sin\theta \ (5)$$
$$y' = -x\sin\theta + y\cos\theta \ (6)$$

Equations (5) and (6) can be represented in matrix form as: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$
Given a point, which has a base (x, y). Here are some special rotations:

- 90^0 clockwise rotation: (x, y) becomes $(y, -x)$
- 90^0 counterclockwise rotation: (x, y) becomes $(-y, x)$
- 180^0 clockwise and counterclockwise rotation: (x, y) becomes $(-x, -y)$
- 270^0 clockwise rotation: (x, y) becomes $(-y, x)$
- 270^0 counterclockwise rotation: (x, y) becomes $(y, -x)$

But since the JavaFX library already includes the rotate class, we do not need to use the above formula to find the corresponding coordinate. Here is the website for all of the things you need to know about the Rotate in JavaFX library:
`https://docs.oracle.com/javase/8/javafx/api/javafx/scene/transform/Rotate.html`

Chapter 4

Related Work

4.1 Algorithm

According to *Generating a fractal tree, Jingyi Gao (2020)*, there is a way to draw a fractal tree using mathematics. In the report, he explains the definition of "Similarity dimension", "Fractal dimension and scaling relationships", "Da Vinci's rule and its analog for two-dimensional tree", "Murray's law", "Scaling exponent in the conservation law and exponent in the scaling relationship", and provides the reader about the principles and algorithm to draw a fractal tree. Here we mostly focus on the algorithm of "JINGYI GAO".

The algorithm of *Jingyi Gao* in *Generating a fractal tree*:

Step 1: Create a recursive function with parameter x coordinate, y coordinate, and branch length l.

Step 2: Check whether the length of the branch is less than 1/9. If it's true, break.

Step 3: Create a path object that draws a "v" with each branch of length 1/2 of the input length.

Step 4: Call the function on the corresponding x coordinate and y coordinate of the left endpoint of the "v" just drawn.

Step 5: Call the function on the corresponding x coordinate and y coordinate of the right endpoint of the "v" just drawn.

Jingyi Gao algorithm used recursion to draw the tree, but we used the iterative method. Both methods work in the same way, but we find that iteration is easier to manage in our project. At first, we used recursion and did not store the branches, so we can not do the animation part of the tree. Next, we changed to use iteration to store the branches so that the animation part work perfectly. However, looking back to the recursion, it still can store the branches.

At first, we used recursion and did not store the branches, so we can not do the animation part of the tree. Next, we changed to use iteration to store the branches so that the animation part work perfectly. However, looking back to the recursion, it still can store the branches.

4.2 Application

It is a tree data structure that makes data sorted and allows searches and sequential access at the same time as B-Tree, but with insertions and deletions that are asymmetrically faster than B-Tree. In fractal trees, a node can have more than two children, like a B-tree. Fractal tree indexes also feature intermediate buffers at each node, unlike a B-tree, which allows insertions, deletions, and other changes to be stored. The goal of the buffers is to schedule disk writes so that each write performs a large amount of useful work, thereby avoiding the worst-case performance of B-trees, in which each disk write may change a small amount of data on the disk, Wikipedia (2014).
According to Tutorials Point(n.d.), fractals can be used for:

- Astronomy: For analyzing galaxies, rings of Saturn, etc.
- Biology/Chemistry: For depicting bacteria cultures, Chemical reactions, human anatomy, molecules, plants, etc.
- Others: For depicting clouds, coastline, and borderlines, data compression, diffusion, economy, fractal art, fractal music, landscapes, special effect, etc.

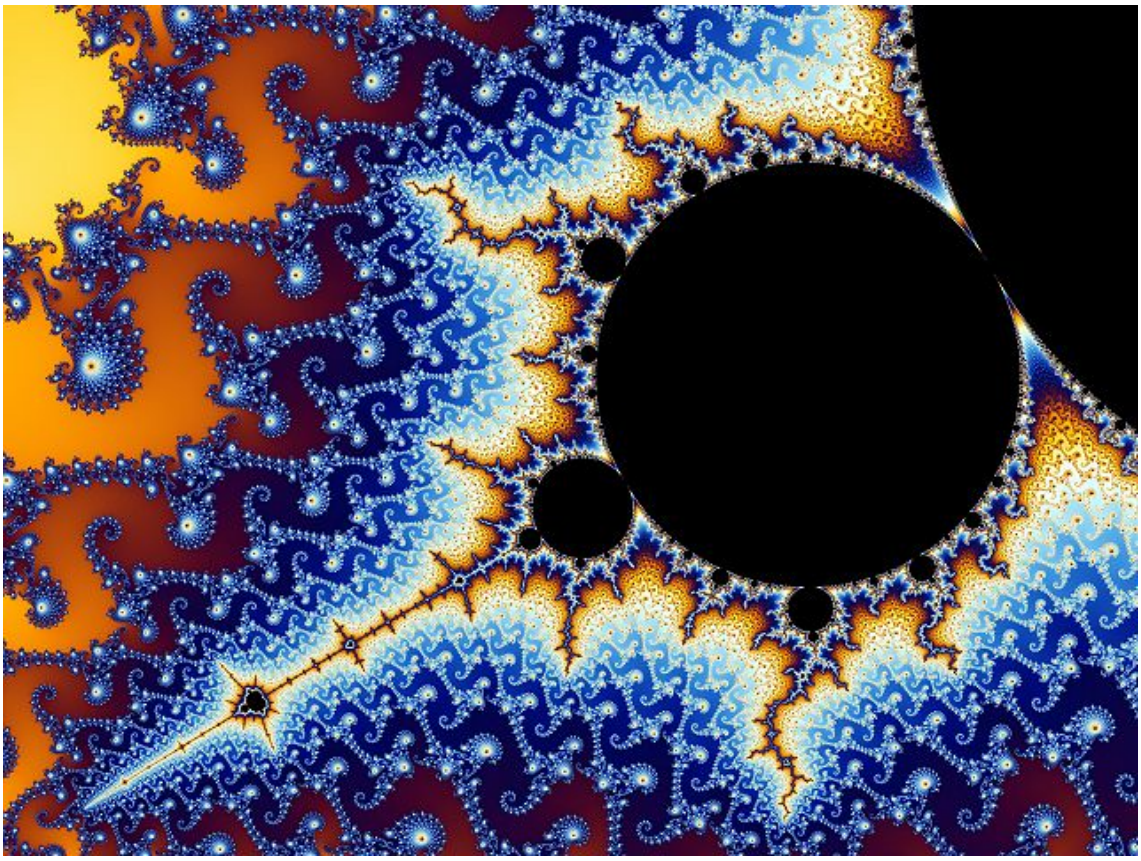

Figure 4.1: Fractal used in computer graphics.

Chapter 5

Proposed Approaches

5.1 Menu user interface

Our first approach is just to create a GUI for a single tree only. After creating the first tree, which is the symmetric tree, there is still a lot of time for us to work on this project, so we agree that we will create other GUIs for other trees and a Starting GUI (Menu GUI) to link all the GUIs of trees together. Here is the Starting GUI (Menu GUI):

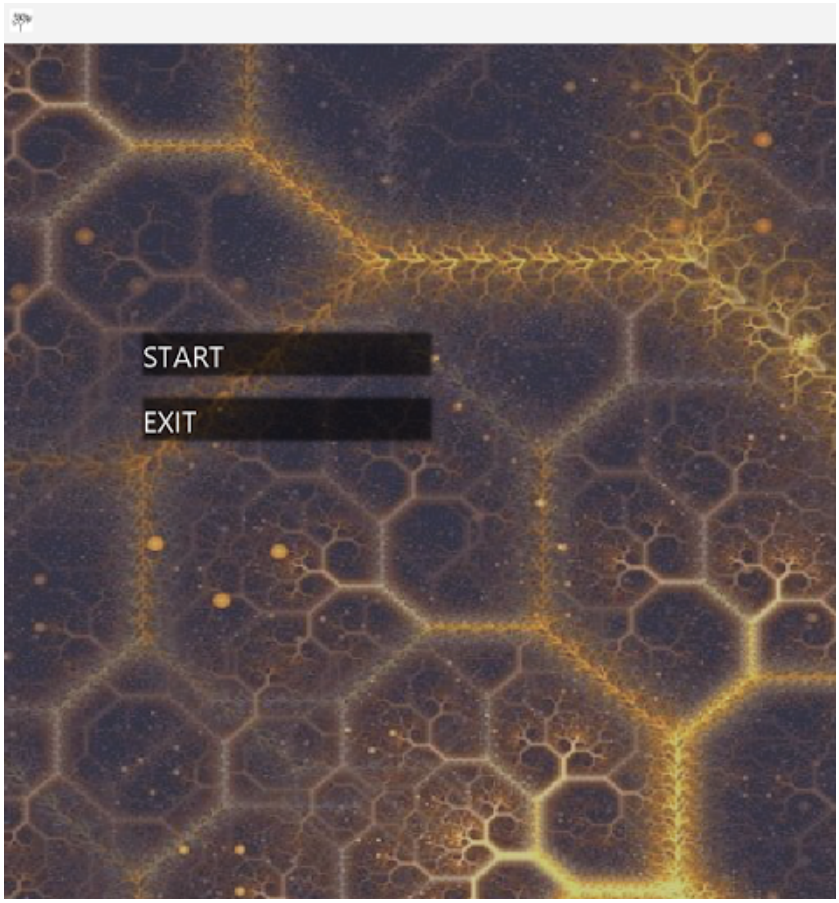

Figure 5.1: GUI of The Application.

In the Starting GUI (Menu GUI), we have only 2 buttons: Start and Exit. If the Start button is selected, we will be able to choose one of the three trees to start the application. If the Exit button is selected, the application will close (end).

Figure 5.2: Starting GUI (Menu GUI) after the Start button is selected.

After selecting the Start button, the user will be able to choose to draw a “Symmetric Tree”, “Spiral Tree” or “Binary Tree”. The name of the button said it all. With the “Symmetric Tree” button, the user will go to the GUI which draws the symmetric fractal tree. With the “Spiral Tree” button, the user will go to the GUI which draws the spiral fractal tree. With the “Binary Tree” button, the user will go to the GUI which draws the

binary fractal tree. The binary tree represents the symmetric tree and spiral tree at some special angle value.

5.2 Drawing user interface

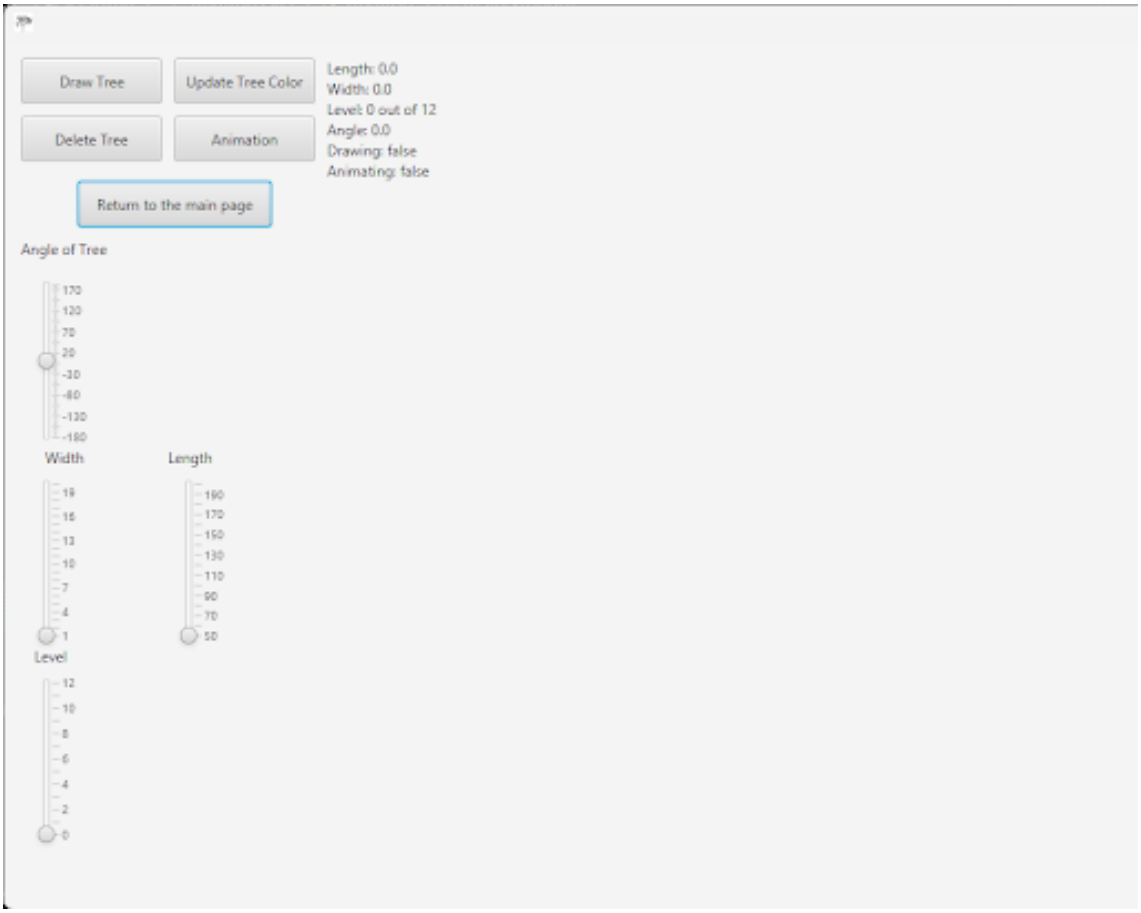

Figure 5.3: Symmetric Tree GUI.

In the Symmetric Tree GUI, there are lots of buttons and sliders:

- The **Draw Tree** button is used to draw the tree after the user has adjusted all the properties of the tree. After the textbfDraw Tree button is pressed, if the user wants to change the properties, the tree will adjust accordingly to the new properties.
- The **Update Tree Color** button is used to update the color of the branches when the tree is drawn for the first time, all branches (leaves) have the color back, and when the level is increased, the new branches (leaves) will also have the color black. The Update Tree Color button updates the color of the tree randomly, based on the HSB color system.
- The **Delete Tree** button is used to delete the tree. Besides that, if the animation is running, it will also be stopped.

- The **Animation** button is used to create animation for the tree. It will change the angle via a constant. With each time the angle change, the tree is bent. When the animation starts, all of the properties of the tree can still be changed.
- The **Return to the main page** button is used to return back to the Menu GUI for the user to choose other trees or end the program.
- The **Angle of Tree** slider is used to set the angle of the branches (leaves) compare to their parent.
- The **Width** slider is used to set the width of the root of the tree. The width of the branches (leaves) will be decreased after each level.
- The **Length** slider is used to set the length of the root of the tree. The length of the branches (leaves) will be decreased after each level
- The **Level** slider is used to set the level of the tree. The root is understood as level 0.

The pseudo-code for drawing the symmetric tree:

```
function drawSymmetric(){
  draw (root, length, width, angle = 0, level = 0)
  for (int i=0; i<level; i++){
    for each branch in the level i {
      Temp_length = root.length / 1.3;
      Temp_Width = root.width / 1.3;
      draw(branch(i).left, Temp_length, Temp_Width,
            root.angle - angleChange, level = i+1);
      draw(branch(i).right, Temp_length, Temp_Width,
            root.angle + angleChange, level = i+1);
        }
    }
}
```

Listing 5.1: Pseudo-code for drawing the symmetric tree

Figure 5.4: Spiral Tree GUI.

In the Spiral Tree GUI, there are lots of buttons and sliders:

- The **Draw Tree** button is used to draw the tree after the user has adjusted all the properties of the tree. After the **Draw Tree** button is pressed, if the user wants to change the properties, the tree will adjust accordingly to the new properties.
- The **Update Tree Color** button is used to update the color of the branches when the tree is drawn for the first time, all branches (leaves) have the color back, and when the level is increased, the new branches (leaves) will also have the color black. The **Update Tree Color** button updates the color of the tree randomly, based on the HSB color system.
- The **Delete Tree** button is used to delete the tree. Besides that, if the animation is running, it will also be stopped.

- The **Animation** button is used to create animation for the tree. It will change the angle via a constant. With each time the angle change, the tree is bent. When the animation starts, all of the properties of the tree can still be changed.
- The **Return to the main page** button is used to return back to the Menu GUI for the user to choose other trees or end the program.
- The **Left** and **Right** radio buttons are used to change the tree angle. If the **Left** radio button is chosen, the tree will bend to the left, similarly to the **Right** radio button.
- The **Angle of Tree** slider is used to set the angle of the branches (leaves) compare to their parent.
- The **Width** slider is used to set the width of the root of the tree. The width of the branches (leaves) will be decreased after each level.
- The **Length** slider is used to set the length of the root of the tree. The length of the branches (leaves) will be decreased after each level
- The **Level** slider is used to set the level of the tree. The root is understood as level 0.

The pseudo-code for drawing the spiral tree:

```
function drawSpiral(){
  draw (root, length, width, angle = 0, level = 0)
  for (int i=0; i<level; i++){
    for each branch in the level i {
      Temp_length = root.length / 1.3;
      Temp_Width = root.width / 1.3;
      draw(branch(i).left, Temp_length, Temp_Width,
          root.angle + (LeftOrRight) * angleChange, level = i+1);
        }
    }
}
```

Listing 5.2: Pseudo-code for drawing the spiral tree.label

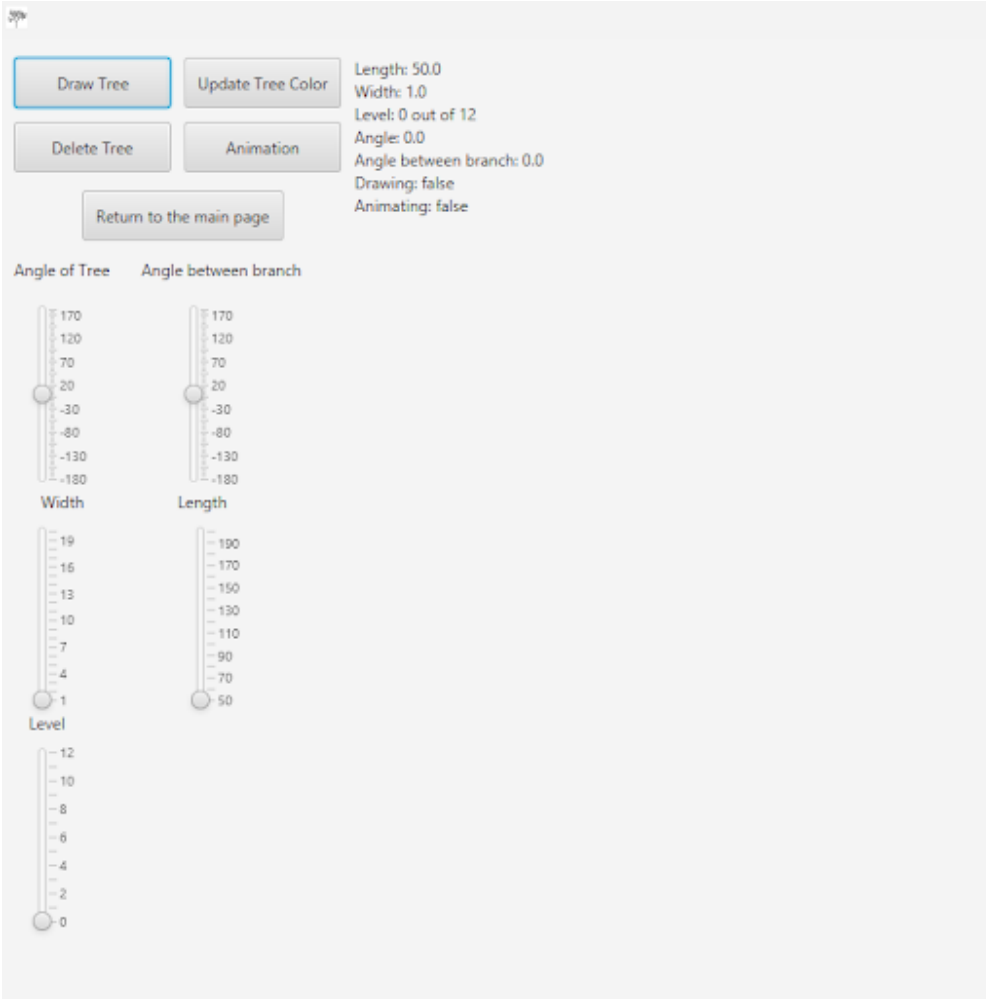

Figure 5.5: Binary Tree GUI.

In the Binary Tree GUI, there are lots of buttons and sliders:

- The **Draw Tree** button is used to draw the tree after the user has adjusted all the properties of the tree. After the **Draw Tree** button is pressed, if the user wants to change the properties, the tree will adjust accordingly to the new properties.
- The **Update Tree Color** button is used to update the color of the branches when the tree is drawn for the first time, all branches (leaves) have the color back, and when the level is increased, the new branches (leaves) will also have the color black. The **Update Tree Color** button updates the color of the tree randomly, based on the HSB color system.
- The **Delete Tree** button is used to delete the tree. Besides that, if the animation is running, it will also be stopped.
- The **Animation** button is used to create animation for the tree. It will change the angle via a constant. With each time the angle change, the tree is bent. When the animation starts, all of the properties of the tree can still be changed.

- The **Return to the main page** button is used to return back to the Menu GUI for the user to choose other trees or end the program.
- The **Left** and **Right** radio buttons are used to change the tree angle. If the **Left** radio button is chosen, the tree will bend to the left, similarly to the **Right** radio button.
- The **Angle of Tree** slider is used to set the angle of the branches (leaves) compare to their parent.
- The **Width** slider is used to set the width of the root of the tree. The width of the branches (leaves) will be decreased after each level.
- The **Length** slider is used to set the length of the root of the tree. The length of the branches (leaves) will be decreased after each level
- The **Level** slider is used to set the level of the tree. The root is understood as level 0.

The pseudo-code for drawing the binary tree:

```
function drawBinary (){
  draw (root, length, width, angle = 0, level = 0)
  for (int i=0; i<level; i++){
    for each branch in the level i {
      Temp_length = root.length / 1.3;
      Temp_Width = root.width / 1.3;
      draw(branch(i).left, Temp_length, Temp_Width,
            root.angle - angleChange, level = i+1);
      draw(branch(i).right, Temp_length, Temp_Width,
          branch(i).left.angle + angleBetweenBranch, level = i+1);
        }
    }
}
```

Listing 5.3: Pseudo-code for drawing the binary tree.label

Chapter 6

Implementation Details

For the GUI, we separated it into two parts. The GUI of the tree drawing tool is made via the Scene Builder application - which allows you to easily layout JavaFX UI controls, charts, shapes, and containers so that you can quickly prototype user interfaces. Animations and effects can be applied seamlessly for more sophisticated UIs.
In the GUI of the program menu part, we code it directly without using Scene Builder. We use Java and some of the JavaFX libraries to create the menu and the effects of the buttons.
For easier coding work, we have used several libraries. But the most used library in our group is JavaFX - a Java library that consists of classes and interfaces that are written in native Java code. The APIs are designed to be a friendly alternative to Java Virtual Machine (Java VM) languages, such as JRuby and Scala. We use it to build the drawing tool algorithm and the program menu.
Here are some of the UML Diagrams for our project.

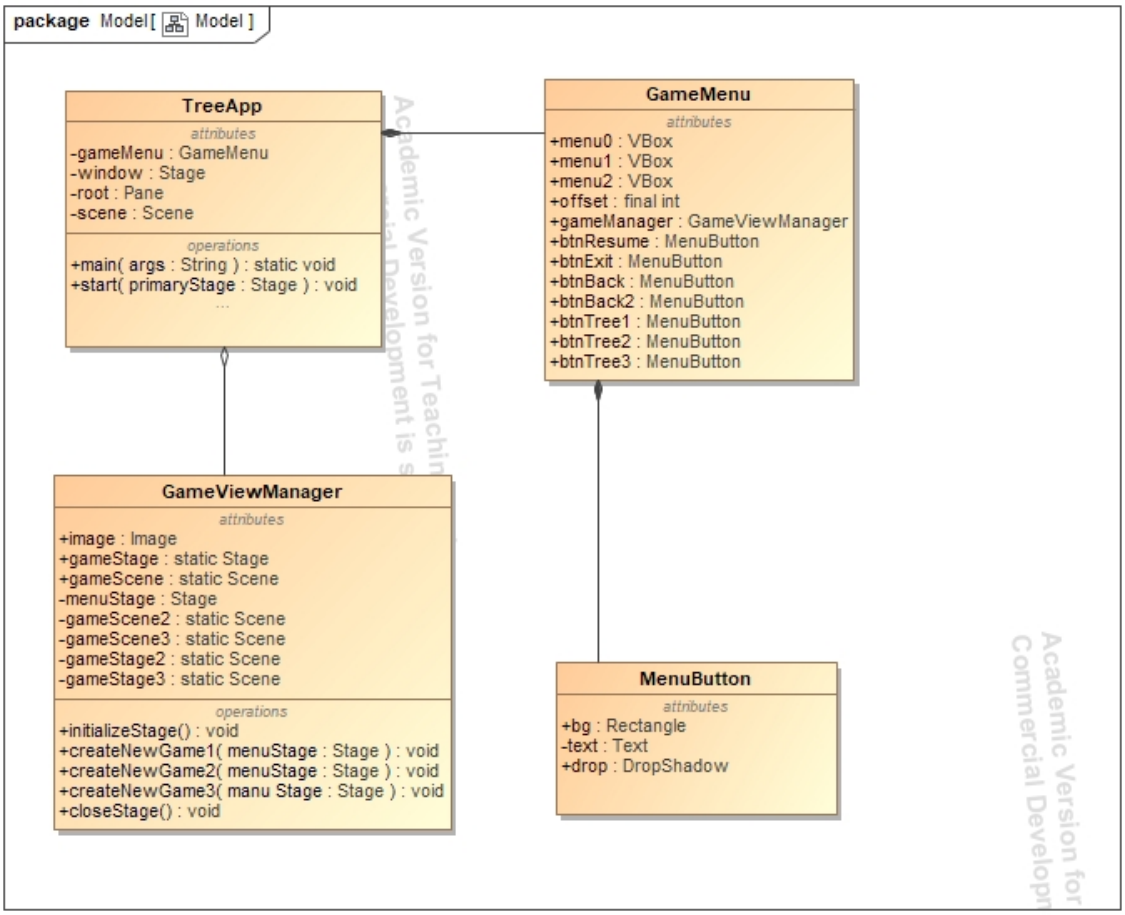

Figure 6.1: Class Diagram for Game Menu.

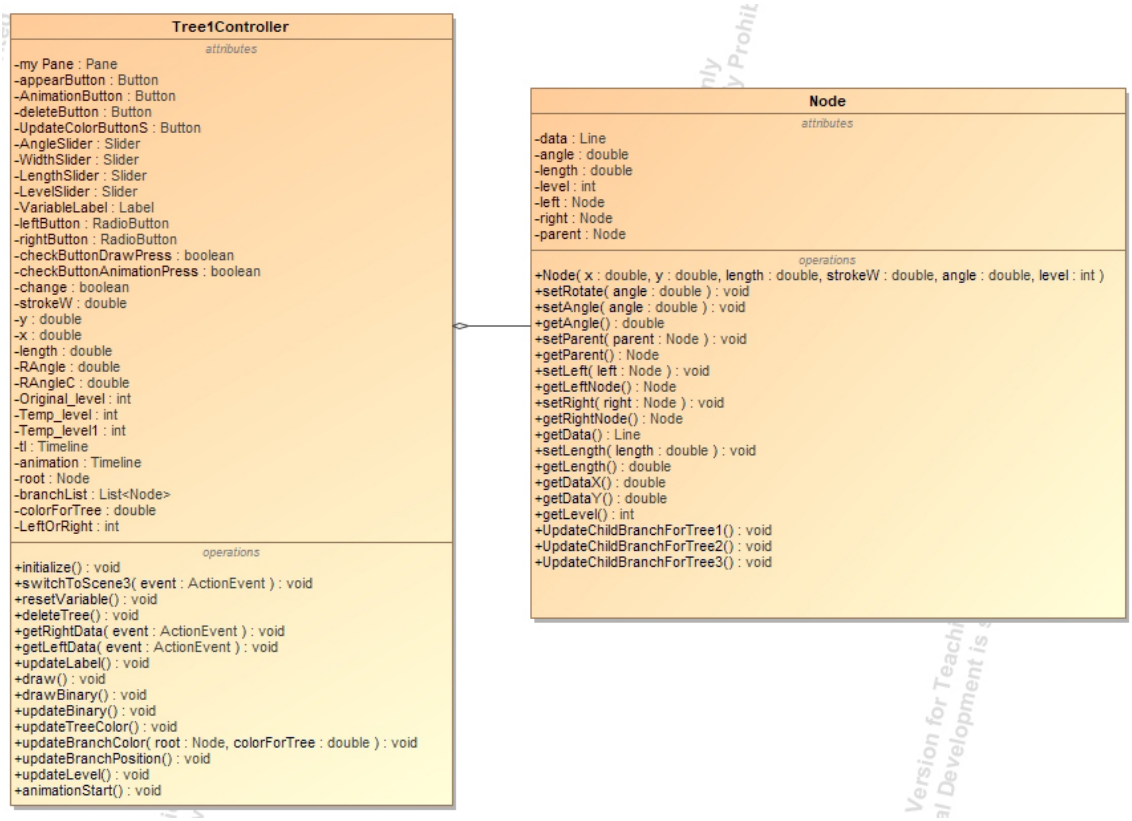

Figure 6.2: Class Diagram for Spiral Tree.

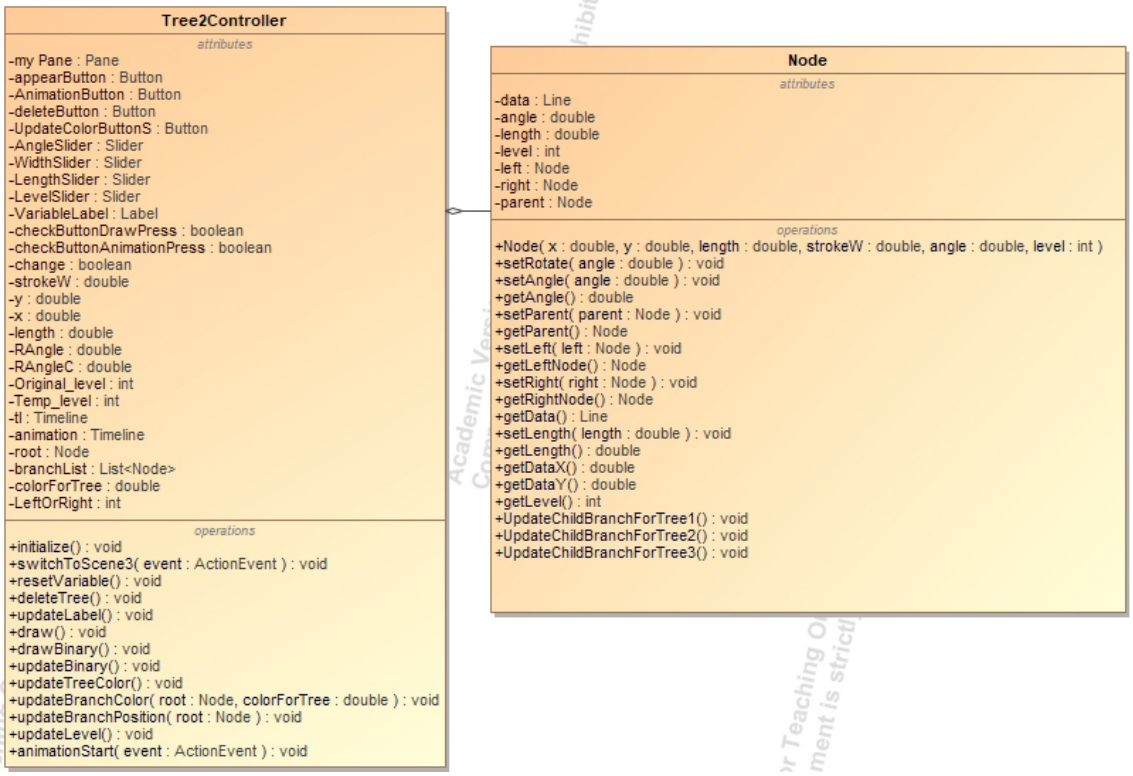

Figure 6.3: Class Diagram for Symmetric Tree.

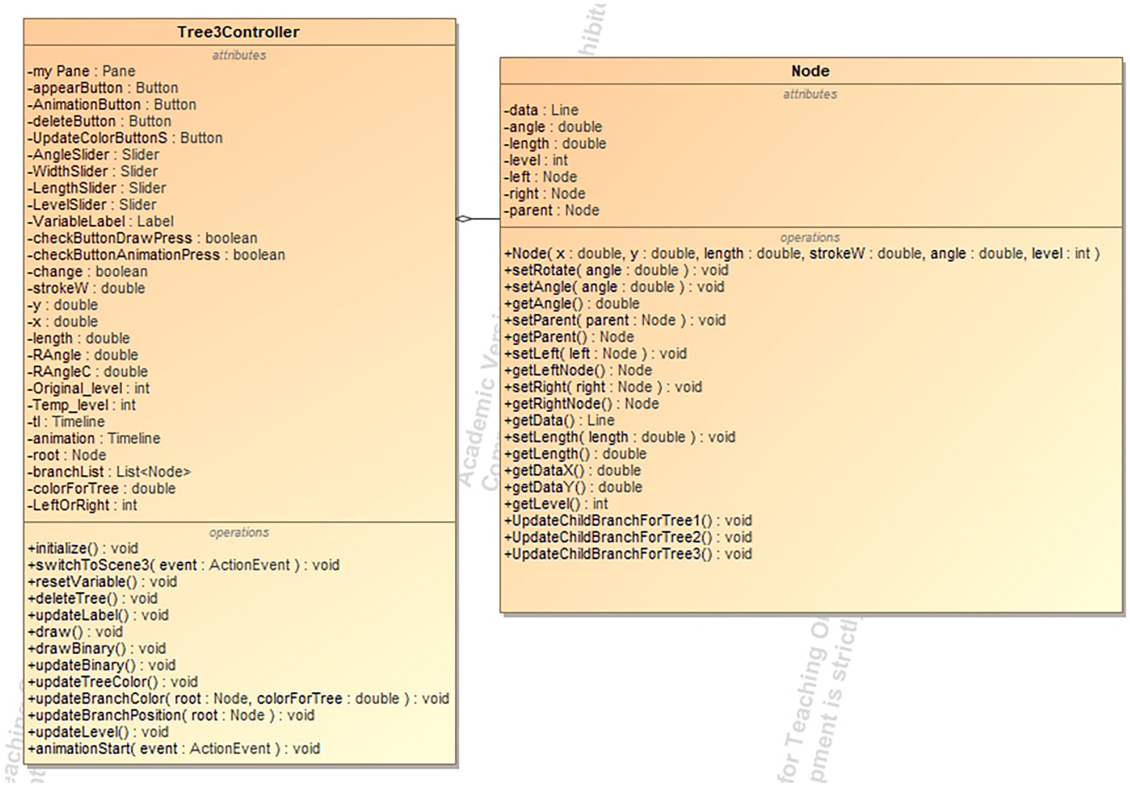

Figure 6.4: Class Diagram for Binary Tree.

6.1 Node.java

The Node class is being used by every controller flies. So firstly, we need to take a look inside the Node.java file:

```
1     private Line data;
2     private double angle;
3     private double length;
4     private int level;
5     private Node left;
6     private Node right;
7     private Node parent;
```

Listing 6.1: Needed Variables.

Here we create every needed variable for the Node.java file:

- data: this is the Line in javaFX, it represents the branch (leaf)
- angle: this is the angle of the branch (leaf)
- length: the length of the branch (leaf)
- level: the level of the branch (leaf)
- left: the left branch
- right: the right branch

- parent: the parent branch

```
1     Node(double x, double y, double length, double strokeW,double
      angle, int level){
2         this.level = level;
3         this.angle = angle;
4         this.length = length;
5         data = new Line();
6         data.setStartX(x);
7         data.setEndX(x);
8         data.setStartY(y);
9         data.setEndY(y-length);
10        data.setStrokeWidth(strokeW);
11        setRotate(angle);
12        left = right = parent = null;
13    }
```

Listing 6.2: Constructor of Node.java.

In the constructor, we create a single straight line base on the x,y and length variables, and then change the angle using the setRotate() method, change the StrokeWidth using the setStrokeWidth method that was create for the Line library.

```
1     private void setRotate(double angle){
2         Rotate rotate = new Rotate();
3         rotate.setPivotX(data.getStartX());
4         rotate.setPivotY(data.getStartY());
5         rotate.setAngle(angle);
6         data.getTransforms().add(rotate);
7     }
```

Listing 6.3: setRotate() method.

This method is used to rotate the branch (leaf) using the built-in method of the javaFX library.The reason why we create this method separately is that we will use this method in the *UpdateChildBranchForTree1()*, *UpdateChildBranchForTree2()*, and *UpdateChildBranchForTree3()*.

```
1     public void setAngle(double angle){
2         this.angle = angle;
3     }
4     public void setParent(Node parent){
5         this.parent = parent;
6     }
7     public void setLeft(Node left){
8         this.left = left;
9     }
10    public void setRight(Node right){
11        this.right = right;
12    }
13    public void setLength(double length){
14        this.length = length;
15        data.setEndY(data.getStartY()-length);
16    }
17    public Line getData(){
18        return data;
19    }
```

```
20      public Node getLeftNode(){
21          return left;
22      }
23      public Node getRightNode(){
24          return right;
25      }
26      public Node getParentNode(){
27          return parent;
28      }
29      public double getAngle(){
30          return angle;
31      }
32      public double getLength(){
33          return length;
34      }
35      public double getDataX(){
36          return data.localToParent(data.getEndX(),data.getEndY()).
    getX();
37      }
38      public double getDataY(){
39          return data.localToParent(data.getEndX(),data.getEndY()).
    getY();
40      }
41      public int getLevel(){
42          return level;
43      }
```

Listing 6.4: Setters and Getters.

The above code snippet represents all of the *setter* and *getter* needed for the controller files.

```
1       public void UpdateChildBranchForTree1(double RAngleC, int
    LeftOrRight){
2           //Update child
3           left.data.setStartX(getDataX());
4           left.data.setEndX(getDataX());
5           left.data.setStartY(getDataY());
6           left.data.setEndY(getDataY()-left.length);
7           left.data.getTransforms().clear();
8           left.setAngle(this.angle + LeftOrRight*RAngleC);
9           left.setRotate(this.angle + LeftOrRight * RAngleC);
10          left.data.setStrokeWidth(this.data.getStrokeWidth()/1.3);
11          left.setLength(this.getLength()/1.3);
12      }
```

Listing 6.5: UpdateChildBranchForTree1() method.

The *UpdateChildBranchForTree1()* method is created for the *Tree1Controller.java* file. Via this method, the child branch of the current node will be updated. The tree can change the width, length, and angle with this method.

```
1       public void UpdateChildBranchForTree2(double RAngleC){
2           //Update left child
3           left.data.setStartX(getDataX());
4           left.data.setEndX(getDataX());
5           left.data.setStartY(getDataY());
```

```
        left.data.setEndY(getDataY()-left.length);
        left.data.setEndY(getDataY()-this.getLength()/1.3);
        left.data.getTransforms().clear();
        left.setAngle(this.angle - RAngleC);
        left.setRotate(this.angle - RAngleC);
        left.data.setStrokeWidth(this.data.getStrokeWidth()/1.3);
        left.setLength(this.getLength()/1.3);

        //Update right child
        right.data.setStartX(getDataX());
        right.data.setEndX(getDataX());
        right.data.setStartY(getDataY());
        right.data.setEndY(getDataY()-left.length);
        right.data.setEndY(getDataY()-this.getLength()/1.3);
        right.data.getTransforms().clear();
        right.setAngle(this.angle + RAngleC);
        right.setRotate(this.angle + RAngleC);
        right.data.setStrokeWidth(this.data.getStrokeWidth()/1.3);
        right.setLength(this.getLength()/1.3);
    }
```

Listing 6.6: UpdateChildBranchForTree2() method.

The *UpdateChildBranchForTree2()* method is created for the *Tree2Controller.java* file. Via this method, the children branches of the current node will be updated. The tree can change the width, length, and angle with this method.

```
    public void UpdateChildBranchForTree3(double RAngleC, double
    AngleBetweenBranch){
        //Update left child
        left.data.setStartX(getDataX());
        left.data.setEndX(getDataX());
        left.data.setStartY(getDataY());
        left.data.setEndY(getDataY()-left.length);
        left.data.setEndY(getDataY()-this.getLength()/1.3);
        left.data.getTransforms().clear();
        left.setAngle(this.angle - RAngleC);
        left.setRotate(this.angle - RAngleC);
        left.data.setStrokeWidth(this.data.getStrokeWidth()/1.3);
        left.setLength(this.getLength()/1.3);

        //Update right child
        right.data.setStartX(getDataX());
        right.data.setEndX(getDataX());
        right.data.setStartY(getDataY());
        right.data.setEndY(getDataY()-right.length);
        right.data.getTransforms().clear();
        right.setAngle(left.getAngle() + AngleBetweenBranch);
        right.setRotate(left.getAngle() + AngleBetweenBranch);
        right.data.setStrokeWidth(this.data.getStrokeWidth()/1.3);
        right.setLength(this.getLength()/1.3);
    }
```

Listing 6.7: UpdateChildBranchForTree3

The *UpdateChildBranchForTree3()* method is created for the *Tree3Controller.java* file. Via this method, the children branches of the current node will be updated. The tree can

change the width, length, and angle with this method.

6.2 Tree3Controller.java

Since the *Symmetric Fractal Tree*(Tree2Controller.java) and *Spiral Fractal Tree*(Tree1Controller.java) can be represented by *Binary Fractal Tree*(Tree3Controller.java), we only show the code for the Binary Fractal Tree:

```
1     private boolean checkButtonDrawPress = false;
2     private boolean checkButtonAnimationPress = false;
3     private boolean change = false;
4     private double strokeW = 1;
5     private double y = 720;
6     private double x = 1280/2+100;
7     private double length = 50;
8     private double RAngleC = 0;
9     private double AngleBetweenBranch = 0;
10    private double RAngle = 0;
11    private int Original_level = 0;
12    private int Temp_level = 1;
13    private int Temp_level1 = 0;
14    private Timeline tl;
15    private Timeline animation;
16    Node root;
17    private List<Node> branchList = new ArrayList<Node>();
18    private double colorForTree=0;
```

Listing 6.8: Variables for Tree3Controller.java.

Here we create every needed variable for the Tree3Controller.java file:

- checkButtonDrawPress: to check if the draw button is pressed or not
- checkButtonAnimationPress: to check if the animation button is pressed or not
- change: to check if there is any change in the variables that are used to draw the tree
- strokeW: the tree's trunk thickness
- length: the tree's trunk length
- RAngleC: the angle of the branches compare to the trunk
- AngleBetweenBranch: the angle between the left and right branches
- Original_level: store the level of the tree
- Temp_level: the level of the branches (leaf)
- Temp_level1: the current level of the tree
- x: the starting X position of the tree base on coordinate
- y: the starting Y position of the tree base on coordinate

- root: the tree trunk
- branchList: store every tree branch
- tl: the timeline to deal with animation when changing the angle
- animation: the timeline to deal with animation
- colorForTree: get the color base on HSB color system

```
1     public void switchToScene3(ActionEvent event) throws
    IOException {
2           deleteTree();
3           AngleSlider.setValue(0);
4           AngleBetweenBranchSlider.setValue(0);
5           WidthSlider.setValue(1);
6           LengthSlider.setValue(50);
7           LevelSlider.setValue(0);
8           resetVariable();
9           updateLabel();
10          Stage stage;
11          stage = TreeApp.window;
12          Scene scene = TreeApp.scene;
13          stage.setScene(scene);
14          stage.show();
15          GameViewManager.gameStage3.close();
16      }
```

Listing 6.9: switchToScene3() method.

The *switchToScene3()* method is used to return to the *Menu GUI*. When we return to the *Menu GUI*, the GUI of the Binary Fractal Tree will be closed.

```
1     public void updateLabel(){
2           VariableLabel.setText("Length: "+length+"\nWidth: "+
    strokeW+"\nLevel: "+
3               Original_level+" out of 12"+"\nAngle: "+RAngleC+"\
    nAngle between branch: "+AngleBetweenBranch +
4               "\nDrawing: "+checkButtonDrawPress +"\nAnimating: "+
    checkButtonAnimationPress);
5       }
```

Listing 6.10: updateLabel() method.

The *updateLabel()* method is used to update the label when the variables for drawing the tree are changed. This method will not update the *Angle* and *Angle between branch* when the *Animation* button is pressed.

```
1     public void draw(){
2           if (checkButtonDrawPress == false){
3               Temp_level1 = Original_level;
4               root = new Node(x,y,length,strokeW,0,0);
5               branchList.add(root);
6               myPane.getChildren().add(root.getData());
7               checkButtonDrawPress = true;
8               if (Original_level > 0){
```

```
9                 Timeline drawTreeTimeLine = new Timeline(new
    KeyFrame(Duration.millis(300), e ->
10                {
11                    drawBinary();
12                }
13                ));
14                drawTreeTimeLine.setCycleCount(Original_level);
15                drawTreeTimeLine.play();
16            }
17            tl = new Timeline(new KeyFrame(Duration.millis(10), e
    ->
18            {
19                updateBinary();
20            }
21            ));
22            tl.setCycleCount(Timeline.INDEFINITE);
23            tl.play();
24            updateLabel();
25        }
26     }
```

Listing 6.11: draw() method.

The *draw()* method will start the drawing process when the *Draw* button is pressed. If the *Draw* button is already pressed, nothing will happen. Else, it will show the drawing animation of the tree level by level via *drawTreeTimeLine*. The *tl* timeline is used for updating the tree when the variables for drawing the tree are changed.

```
1     public void drawBinary(){
2         //using for loop
3             for (int i = branchList.size()-1; i >=0;i--){
4                 if (branchList.get(i).getLevel() == (Temp_level-1)
    ){
5                     double strokeWTemp = branchList.get(i).getData
    ().getStrokeWidth();
6                     double lengthTemp = branchList.get(i).
    getLength();
7                     strokeWTemp /= 1.3;
8                     lengthTemp /= 1.3;
9                     Node left = new Node(branchList.get(i).
    getDataX(), branchList.get(i).getDataY(),
10                                            lengthTemp, strokeWTemp,
    branchList.get(i).getAngle()-RAngleC,Temp_level);
11                    Node right = new Node(branchList.get(i).
    getDataX(), branchList.get(i).getDataY(),
12                                            lengthTemp, strokeWTemp,
    left.getAngle() + AngleBetweenBranch,Temp_level);
13
14                    left.setParent(branchList.get(i));
15                    right.setParent(branchList.get(i));
16
17                    branchList.get(i).setLeft(left);
18                    branchList.get(i).setRight(right);
19
20                    branchList.add(left);
21                    branchList.add(right);
```

```
22
23                      myPane.getChildren().add(0,left.getData());
24                      myPane.getChildren().add(0,right.getData());
25
26                  }
27              }
28              Temp_level++;
29      }
```

Listing 6.12: drawBinary()

The *drawBinary()* method is used to draw the branches for a new level each time this method is called in the *draw()* method. Each time the level increase, the stroke width and length of the branches are decreased, and the angle of the branches is based on the parent branch.

```
1      public void updateBinary(){
2          if (change == true){
3              updateBranchPosition(root);
4              updateLevel();
5              change = false;
6          }
7      }
```

Listing 6.13: updateBinary() method.

The *updateBinary()* method is used to change the branches of the tree when there are changes in the variables for drawing the tree.

```
1      public void updateBranchPosition(Node root){
2          if (this.root == root){
3              root.getData().setStrokeWidth(strokeW);
4              root.setLength(length);
5          }
6          if (root.getLeftNode() != null && root.getRightNode() !=
       null){
7              root.UpdateChildBranchForTree3(RAngleC,
       AngleBetweenBranch);
8              updateBranchPosition(root.getLeftNode());
9              updateBranchPosition(root.getRightNode());
10         }
11     }
```

Listing 6.14: updateBranchPosition() method.

The *updateBranchPosition()* method is used to change the branches' position, width, and length by using recursive. The root width and length are changed first, and then the child branches are updated accordingly to the root.

```
1      public void updateLevel(){
2          if (Temp_level1 > Original_level){
3              Timeline updateLevel = new Timeline(new KeyFrame(
       Duration.millis(1), e ->
4                  {
5                      for (int i=branchList.size()-1; i>0;i--){
6                          if (branchList.get(i).getLevel() ==
       Temp_level1){
```

```
7                               myPane.getChildren().remove(branchList
    .get(i).getData());
8                               branchList.remove(i);
9                           }
10                      }
11                      Temp_level1--;
12                  }
13                  ));
14              if (Original_level == 0){
15                  updateLevel.setCycleCount(Temp_level1);
16              } else {
17                  updateLevel.setCycleCount(Temp_level1-
    Original_level);
18              }
19              updateLevel.play();
20          } else if (Temp_level1 < Original_level){
21              Temp_level = Temp_level1+1;
22              Timeline updateLevel = new Timeline(new KeyFrame(
    Duration.millis(1), e ->
23                  {
24                      drawBinary();
25                  }
26                  ));
27              updateLevel.setCycleCount(Original_level-Temp_level1);
28              Temp_level1 = Original_level;
29              updateLevel.play();
30          }
31      }
```

Listing 6.15: updateLevel() method.

The *updateLevel()* method is to change the tree base on the level. If the level increase, more branches are added to the pane, else if the level decrease, the branches which have a level larger than the level of the tree will be removed from the pane.

```
1      public void animationStart(ActionEvent event){
2          if (checkButtonAnimationPress == false &&
    checkButtonDrawPress == true){
3              tl.stop();
4              animation = new Timeline(new KeyFrame(Duration.millis
    (50), e ->
5                  {
6                      AngleBetweenBranch+=3;
7                      RAngleC+=2;
8                      updateBranchPosition(root);
9                      updateLevel();
10                 }
11             ));
12             animation.setCycleCount(Timeline.INDEFINITE);
13             animation.play();
14             checkButtonAnimationPress=true;
15         } else if (checkButtonAnimationPress == true) {
16             checkButtonAnimationPress=false;
17             animation.stop();
18             tl.play();
19             RAngleC = RAngleC % 360;
20             if (RAngleC > 180){
```

```
                RAngleC = RAngleC-360;
            }
            AngleBetweenBranch = AngleBetweenBranch % 360;
            if (AngleBetweenBranch > 180){
                AngleBetweenBranch = AngleBetweenBranch-360;
            }
            AngleSlider.setValue(RAngleC);
            AngleBetweenBranchSlider.setValue(AngleBetweenBranch);
        }
        updateLabel();
    }
```

Listing 6.16: animationStart() method.

The *animationStart()* method is used to run the animation of the tree by changing the *angle* and *angle between branch* variables. If the animation is already running and the *Animation* button is pressed, the animation will stop. The *angle* and *angle between branch* will be updated after the animation stop.

```
    public void deleteTree(){
        if (checkButtonDrawPress == true){
            for (int i=0;i<branchList.size();i++){
                myPane.getChildren().remove(branchList.get(i).
getData());
            }
            // Original_level = 0;
            Temp_level = 1;
            checkButtonDrawPress = false;
            tl.stop();
            branchList.clear();
        }
        if (checkButtonAnimationPress == true){
            animation.stop();
            checkButtonAnimationPress = false;
        }
        updateLabel();
    }
```

Listing 6.17: deleteTree() method.

The *deleteTree()* method is used to delete the tree when the *Delete* button is pressed. It will stop the animation if the animation is running when the *Delete* button is pressed.

```
    public void resetVariable(){
        length = 50;
        strokeW = 1;
        Original_level = 0;
        RAngleC = 0;
        AngleBetweenBranch =0;
    }
```

Listing 6.18: resetVariable() method.

The *resetVariable()* method is used to reset the variables to their first value when the application start. This method is called when we want to go back to the *Menu GUI*.

```
    public void updateTreeColor(){
        colorForTree = (new Random()).nextInt(360);
```

```
3           updateBranchColor(root,colorForTree);
4       }
```

Listing 6.19: updateTreeColor() method.

The *updateTreeColor()* method is used to update the color of the tree randomly based on the HSB color system when the *Update Color* button is pressed. The color of the tree is only updated to the corresponding level. If the level of the tree is changed, the new branches will have the color black.

```
1 public void updateBranchColor(Node root, double colorForTree){
2           if (root.getLeftNode() != null){
3               updateBranchColor(root.getLeftNode(),colorForTree+20);
4               updateBranchColor(root.getRightNode(),colorForTree-20)
    ;
5           }
6           root.getData().setStroke(Color.hsb(colorForTree, 0.8, 0.9)
    );
7       }
```

Listing 6.20: updateBranchColor() method.

The *updateBranchColor()* method is used to update the color for the branches base on recursive.

Chapter 7

Experimental Results and Statistical Tests

7.1 Findings and Results

For the purpose of testing and simulating the application, there are multiple cases have been checked and recorded. Cases generated for Symmetric Tree, Spiral Tree, and Binary Tree will be shown as followed:

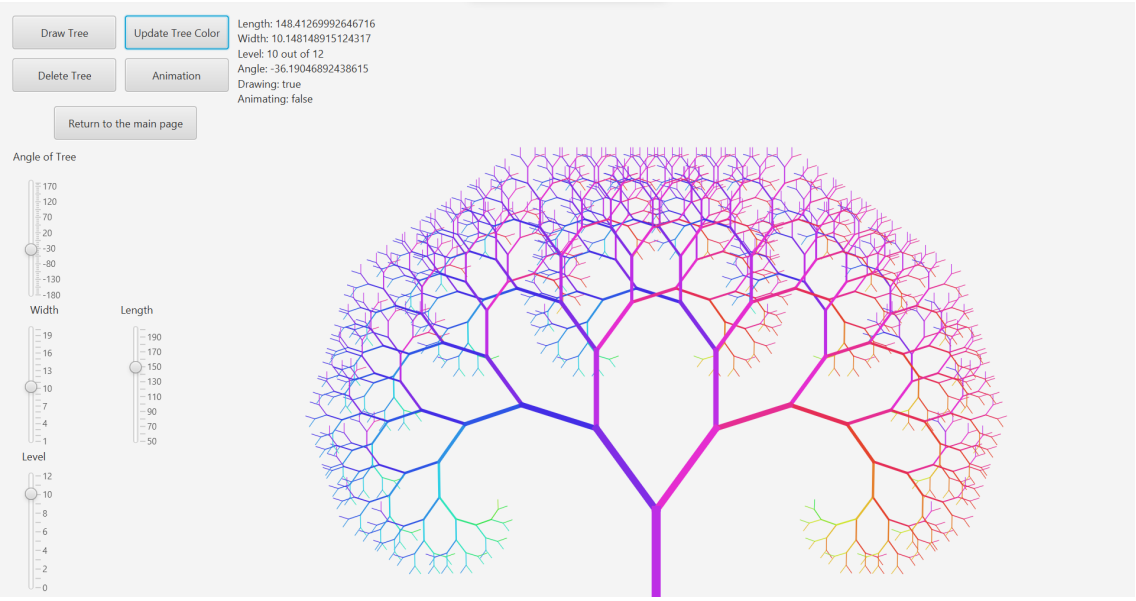

Figure 7.1: Symmetric Tree Test Case.

Figure 7.2: Spiral Tree Test Case.

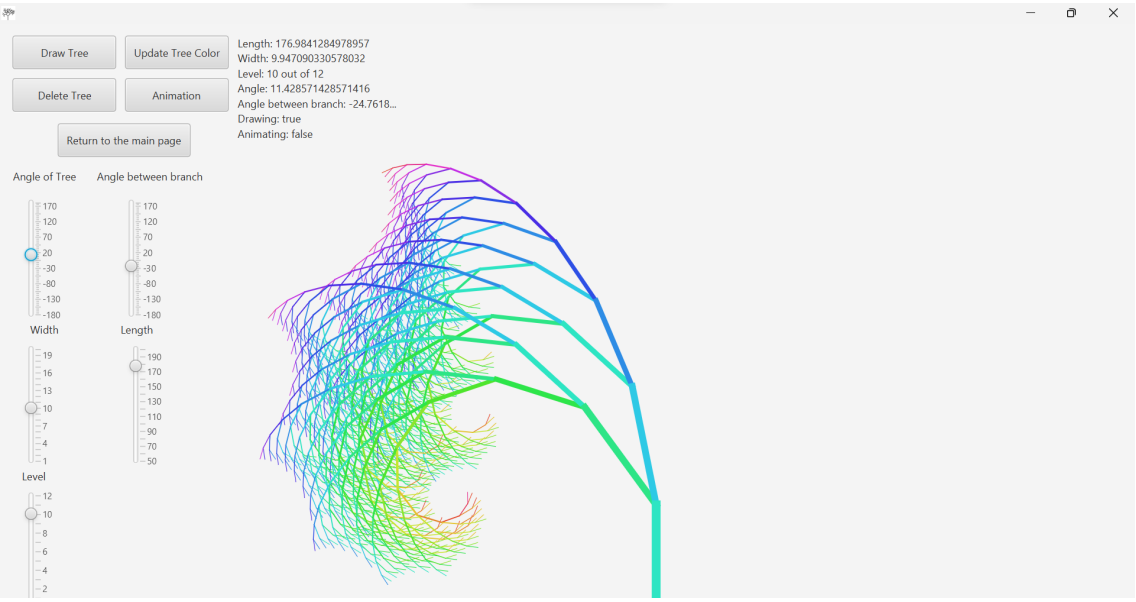

Figure 7.3: Binary Tree Test Case.

Each tree is tested with different lengths, widths, angles, and levels. The application was able to draw the trees smoothly and presented positive results.
Findings:

- Fractal trees are a type of self-similar pattern that can be generated through iteration algorithms and recursive algorithms. In this project, we decided to use iteration algorithms for better results as well as performance.
- Java provides a powerful platform for generating fractal images due to its built-in libraries for graphics and user-friendly syntax. Especially, JavaFX provides a wide

range of built-in functions and sources. Therefore, we were able to program this application effectively.

- To generate a fractal tree, the starting point is to create a basic line segment (trunk) and then repeatedly split it into two smaller segments (branches) and angle them at a certain degree.
- This process is repeated for each branch until a certain depth of iterations is reached.
- A animation button is added to the application which can make the trees move at different angles. This can create a great visual effect whenever the user uses the application.

Results:

- The Java code for generating a fractal tree was successfully implemented and tested. Moreover, the application is able to run smoothly and return accurate results.
- The code allows for user-specified parameters such as the angle of branching, the length of branches, and the number of iterations. We also added a function that enables users to update the color of the tree. As a result, generated trees seemed to be more colorative and fascinating.
- The generated fractal tree images display the self-similar pattern characteristic of fractals. In addition, the animation makes the program more interesting and impressive.
- The code can be easily modified to create different variations of fractal trees by adjusting the parameters. The application added several types of trees namely Symmetric Tree, Spiral Tree, and Binary Tree. This created a diversity of options for the application and therefore, make the program more interesting.

7.2 Statistical Tests

The table 7.1 is an input data table that contains the input parameters of the test cases and their description. The user will adjust the parameters by the slider to create the most suitable tree.
The table 7.2 is a tree results table that contains the parameters of the test cases and the results.

Table 7.1: Input Data Table

Input Parameter	Description
Branching Angle	Angle at which branches split off from the main trunk
Branch Length	Length of each branch
Level of Recursion	Number of times the branching pattern is repeated
Line Width	Width of the branches in the tree
Color	Color of the branches in the tree

Table 7.2: Tree Results Table

Parameter	Symmetric	Sprial	Binary
Branching Angle	-36	48	11
Branch Length	148	280	126
Level of Branch	10	11	10
Line Width	10	10	10
Color	Blue and Pink	Blue and Green	Blue and Green
Result	(Figure 7.1)	(Figure 7.2)	(Figure 7.3)

Note: The Algorithm Results in the Table demonstrate how adjusting the input parameters affects the generated fractal tree. The parameters in the table are in an approximate value.

The statistical test for the Java fractal tree project appears to be well-designed and executed. The results show a good level of accuracy in the fractal tree generation and the use of statistical analysis provides a solid evaluation of the code's performance. However, there may be room for improvement in the interpretation and presentation of the results to make the evaluation more comprehensive and objective.

7.3 Evaluation

The Java project for generating a fractal tree demonstrates a strong understanding of object-oriented programming principles and the ability to create visually appealing outputs. The code is well-organized, using appropriate class and method structures to keep the logic clean and maintainable. The use of iteration for branching in the tree is a clever solution and results in a fractal pattern that is both mathematically interesting and aesthetically pleasing.
Additionally, the use of JavaFX's graphics library to render the tree is efficient and results in a smooth display of the output. The ability to adjust the parameters such as branching angle and branch length allows for a high degree of customization, adding to the overall functionality of the project.
However, there may be room for improvement in terms of the user interface. Currently, the user must manually adjust the parameters by the slider, which may not be user-friendly. Implementing a GUI or user input system could greatly improve the accessibility of the project. Overall, the Java project for generating a fractal tree is a solid experience that gives us more knowledge about the language and the ability to create visually appealing outputs.

7.4 Running the file

To run the file, first, you need to check your java version via Terminal (Powershell) using:

```
java -version
```

If your java version is 17 and above, you will be able to run the file via Terminal (Powershell):

```
java -jar --enable-preview --module-path "javafx-sdk-19/lib" --add-
    modules javafx.controls,javafx.fxml FractalTree.jar
```

Chapter 8

Conclusions and Future Work

8.1 Conclusions

After all, our teamwork went extremely well. Everything worked smoothly and there was no barrier between our communication. First of all, we discussed with each other how we would divide the work into several parts and who would be in charge of which part in a reasonable way. Then, we set up the working calendar and the deadlines for each part in order to finish the work on time. Next, we started to do the work that we had planned such as coding, finding the image, and learning about which algorithm we would use....

Afterward, we discussed one more time to see if the part of each other was good or not, and decided what to fix and what to keep. Finally, we combined everything together to make a complete program, ran it, and checked everything up to see if the program had been done correctly and executed smoothly. Thanks to our good teamwork and communication, we were able to finish the project with barely any difficulties.

We have learned a lot throughout this project, and we found a lot of valuable knowledge as well as important skills. Firstly, we learned how to make a work plan for a project. This is one of the most important skills because if we did not communicate well with each other and plan the project well, then everything would be messed up and the result would not be as we expected. Secondly, we learned how to use JavaFX - one of the most effective software platforms for creating and delivering desktop applications, as well as rich web applications that can run across a wide variety of devices. Thirdly, we learned about the mathematical algorithm to make a Fractal Tree. This is a very interesting algorithm that we need to use recursive programming to implement. Finally, we were able to self-equip ourselves with the skill to work with SceneBuilder - which is an application for JavaFX that brings with it the possibility of testing our designs against different form factors.

8.2 Future work

Although we decided to end the project where it is right now. But in the future, we have plenty of ideas to upgrade our application. We are planning to make the application more interesting by adding some sound effects as well as 3D effects to it. Moreover, we also want to try another alternative algorithm in order to make the running time short as well as make sure that there are no bugs in our code program. We also want to make more alternative trees such as new tree forms, and new colors,... to make the program even more interesting. In conclusion, there are several potential avenues for future work in a Java coding project about generating a fractal tree, including:

1. Improved customization options: The program could be expanded to offer more customization options, such as the ability to adjust the branching pattern, the shape of the branches, and the colors used in the tree.

2. Enhanced image quality: The program could be optimized to provide higher-quality images, with improved resolution and clarity, making it possible to generate larger and more complex trees.

3. Increased performance: The program could be optimized to improve performance and speed, making it possible to generate fractal trees more quickly and efficiently.

4. Improved user experience: The program could be refined to provide a better user experience, with a more intuitive and user-friendly interface, and improved visual appeal.

5. Expansion to other platforms: The program could be ported to other platforms, such as mobile devices or web-based applications, to make it more widely accessible.

By pursuing these areas of future work, the Java coding project about generating a fractal tree can be further developed and improved to provide the best possible user experience.

References

Wikipedia. (n.d.). Fractal. [Online]. Available: https://en.wikipedia.org/wiki/Fractal [Accessed: 4 Feb 2023].

Wikipedia. (n.d.). Fractal Canopy. [Online]. Available: https://en.wikipedia.org/wiki/Fractal_canopy [Accessed: 4 Feb 2023]

Tutorials Point. (n.d.). Computer Graphics - Fractals. [Online]. Available: https://www.tutorialspoint.com/computer_graphics/computer_graphics_fractals.htm#:~:text=What%20are%20Fractals%3F,results%20from%20the%20previous%20iteration [Accessed: 4 Feb 2023].

Gao, J. (2020). [Report]. [Online]. Available: https://lsa.umich.edu/content/dam/cscs-assets/cscs-documents/jingyig_REU_report.pdf [Accessed: 4 Feb 2023].

Wikipedia. (n.d.). Fractal Tree Index. [Online]. Available: https://en.wikipedia.org/wiki/Fractal_tree_index#:~:text=In%20computer%20science%2C%20a%20fractal,faster%20than%20a%20B-tree [Accessed: 4 Feb 2023].