CS Report Shah Mahir Hitesh M21301

Overview And Features

AlgoHolic is an application intended to be used as a teaching tool for teachers to use to teach students about certain algorithms, or for students to learn certain algorithms by themselves. The application contains Sorting, Pathfinding and Hashing algorithms for students to learn and explore.

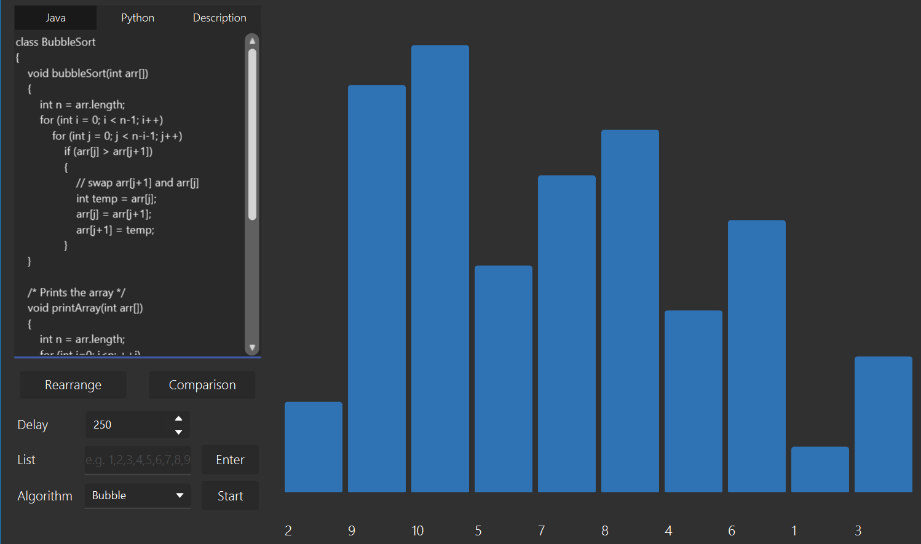
Splash Screen

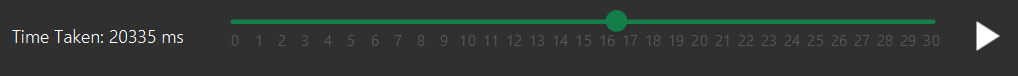
The splash screen shows a pathfinding search animation from one point of the grid to another. Below that, text can be seen as the loading continues.

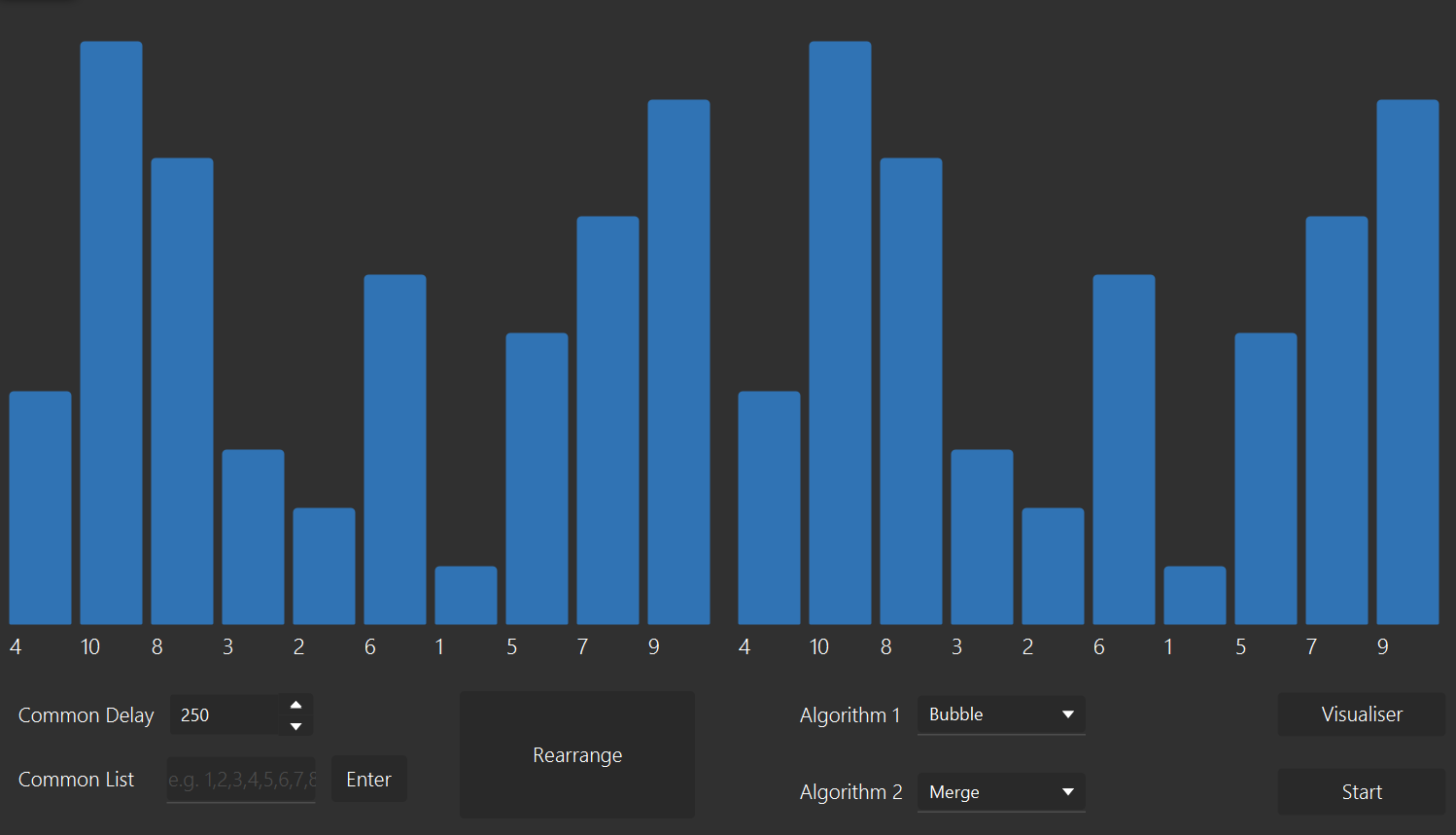
Graphical user interface, application

Description automatically generated

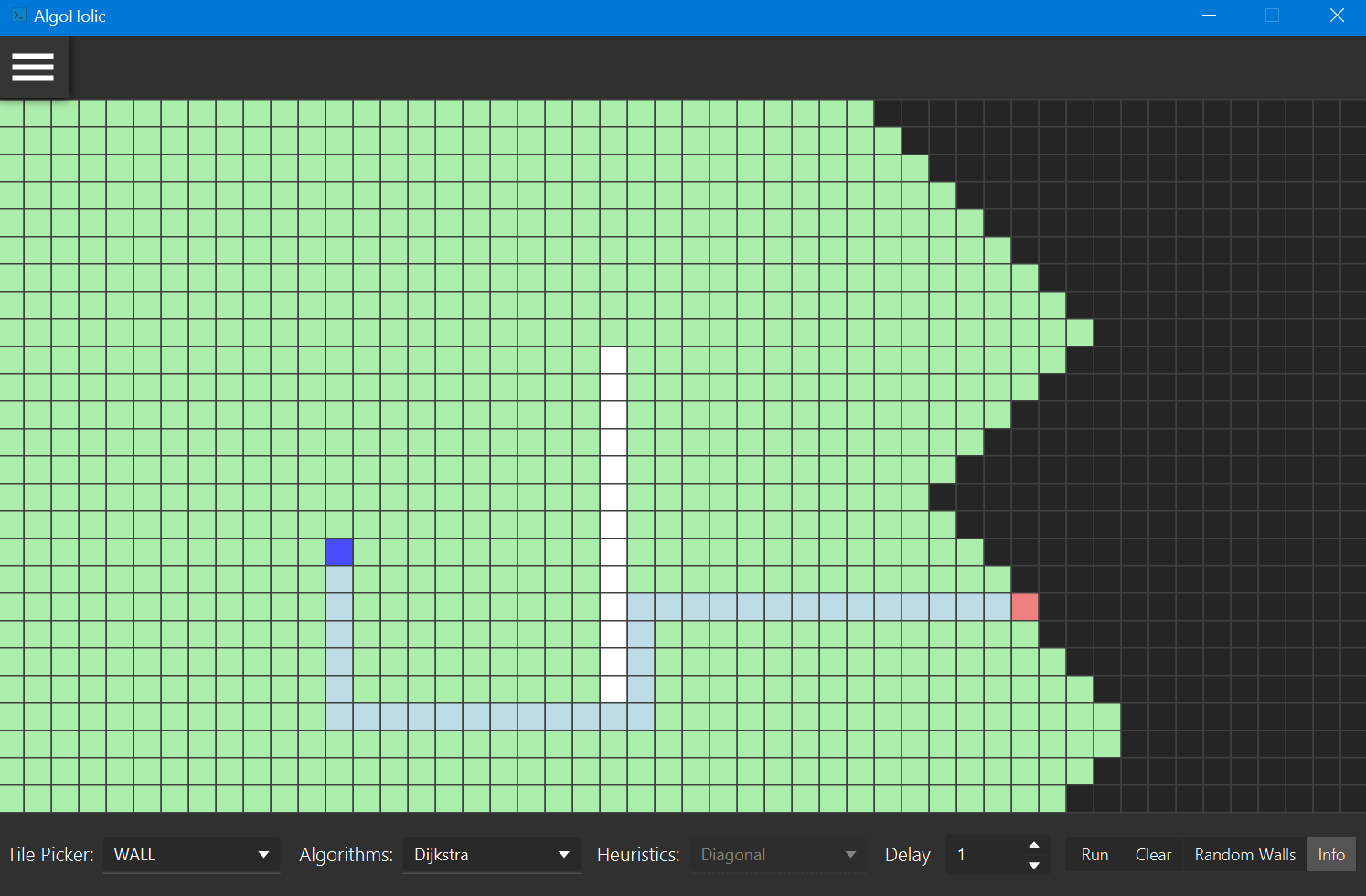
Sorting Menu



The first part of the application is the Sorting Menu. The right part of the menu is the main part, where sorting can be visualised. The left part are the controls that edits the sorting seen in the visualizer. Users can use the rearrange button to rearrange the sorting list seen in the right. Users can use the delay spinner to control the delay of the animation seen. Users can also manually input their own sorting list if they want to. Users must make a choice between 4 algorithms to visualise: Bubble Sort, Merge Sort, Quick Sort and Insertion Sort. The info for each sort can be seen in the TextArea in the top left of the application. Users can choose to see the Java code, Python code and description for each sort. When the sorting is completed, the time taken with the delay can be seen. A slider will also appear, which allows users to backtrack and see the previous states of the list. The play button causes the slider to move forward and the animations to play. There is also a comparison button in the left side that changes the scene to the Comparison Menu.

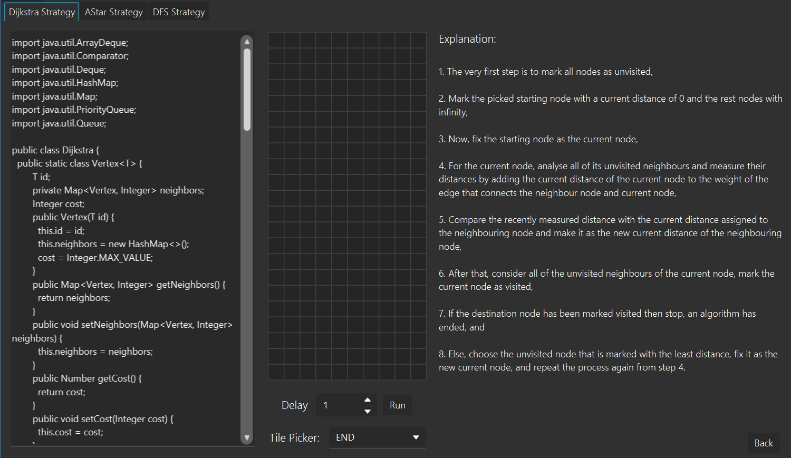
Comparison Menu

In the comparison menu, you can compare the speeds of two different algorithms. When an algorithm is done sorting, the time taken with delay is shown. The user can change the delay for both lists or change the values in the lists. There is also a visualiser button to go back to the Sorting Menu.

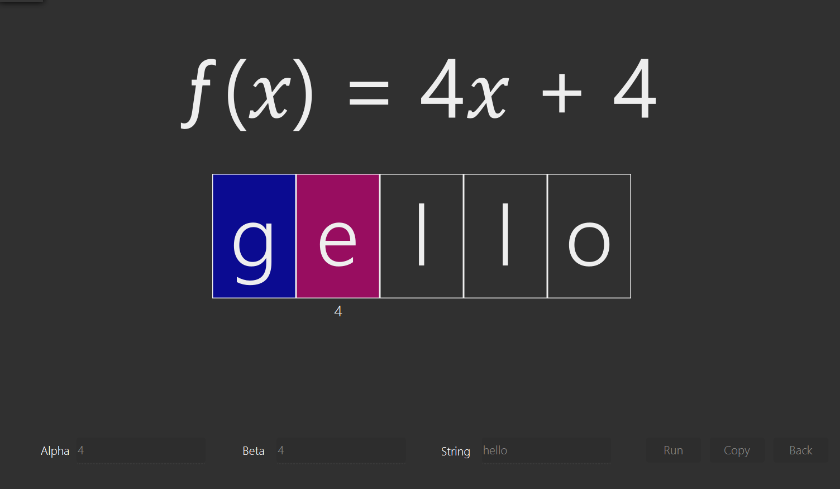
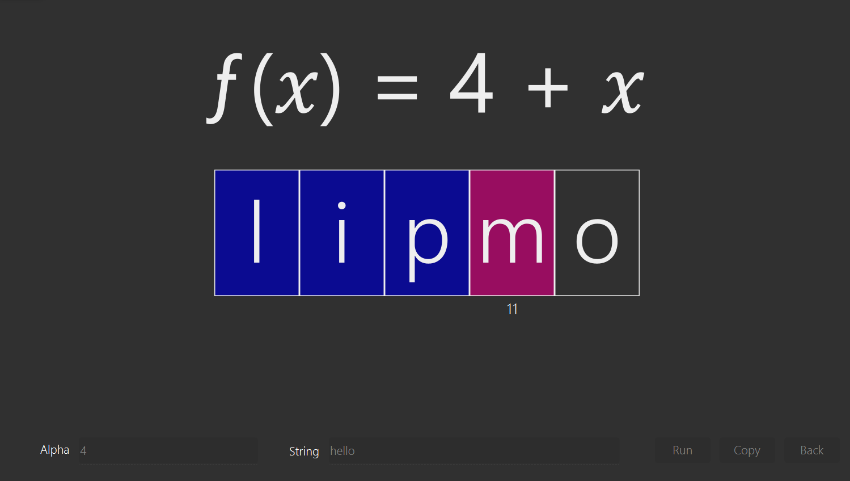
Pathfinding Menu

Using the pathfinding menu, users can place walls, starting and ending points on the grid view. This would allow the user to see how the different pathfinding algorithms would differ in terms if the way they search and how much time they take. Users can choose among Dijkstra Strategy, AStar Strategy and DFS Strategy. For AStar Strategy, different heuristics can be chosen, such as Diagonal, Manhattan and Euclidean. Users can create their custom grids, where they can pick the type for each tile using the Tile Picker ComboBox. Users can also change the delay to visualize the pathfinding algorithm in different speeds. There is a Random Walls button that allows random walls to be generated in the grid. After sorting is done, users can click on the grid to empty it, except for the walls. The Info button is used to change scene to the Info Menu.

Info Menu

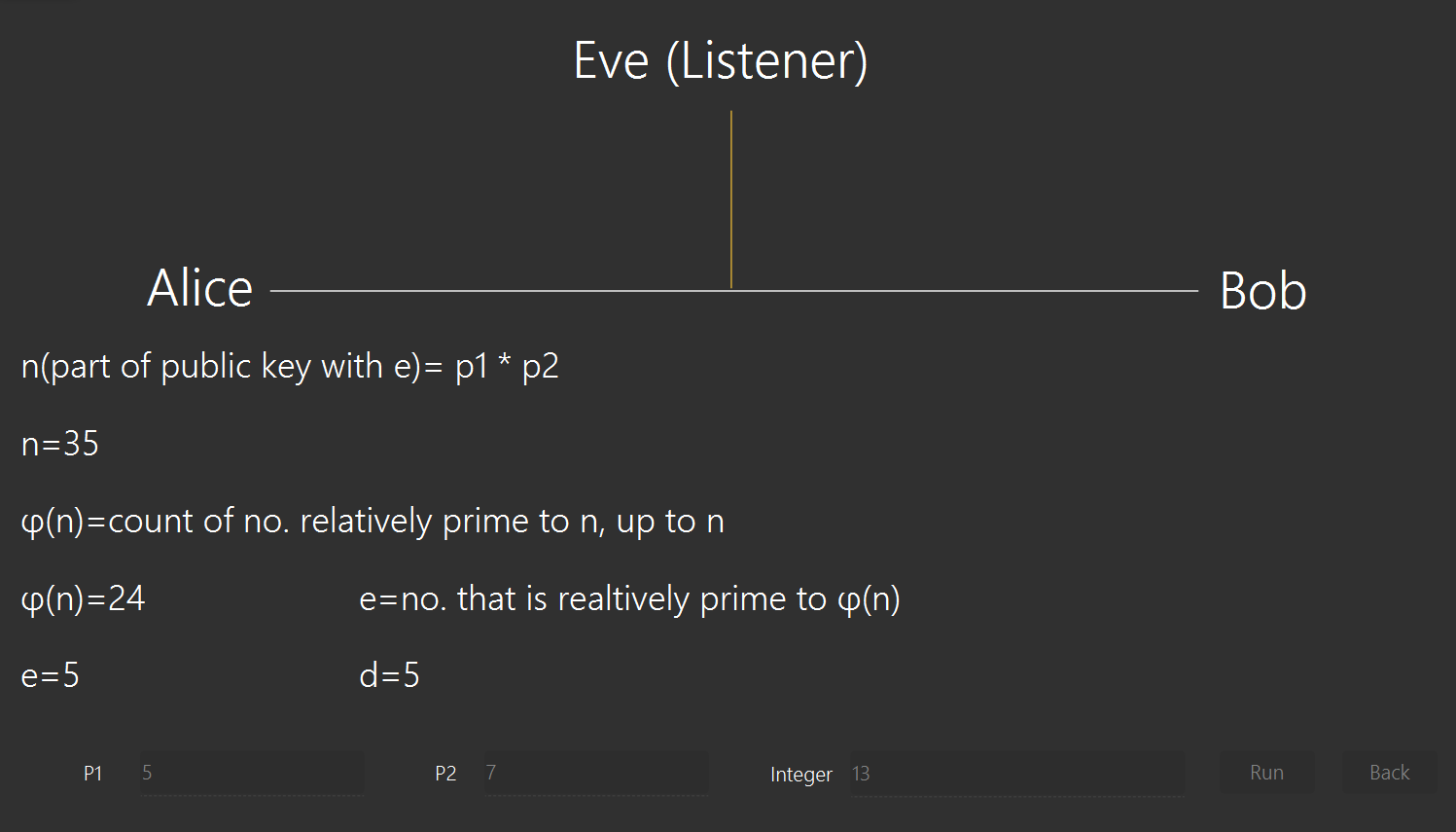
In the Info Menu, students can learn about the three pathfinding algorithms. On the left side, there is a TextArea, which shows the code for the algorithm in Java. In the center, there is a small editable grid where students can play around for the algorithm. Beside the grid, there is an explanation for how the algorithm works. The grid and explanation combined can teach students who are visual learners or reading learners effectively. The back button allows the user to go back to the Pathfinding Menu.

Affine Menu

In the Affine Menu, the user can input the alpha and beta which will be animated to move in the function seen. In the animation, the x value of the letter is shown below the letter, which when used in the function, will result in the ciphered letter The user can also input the string needed to be ciphered. There is also a copy button which allows you to copy the ciphered text shown at the end of the animation.

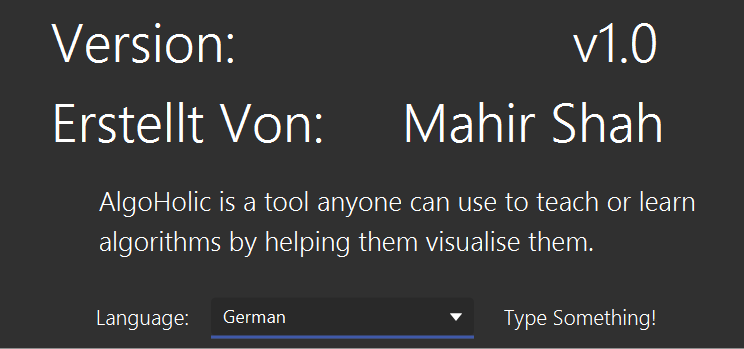
Caesar Menu

In the Caesar Menu, the user can input the alpha and the string needed to be ciphered. In the animation, the letters are cycled through until it reaches the end. There is also a copy button which allows you to copy the ciphered text shown at the end of the animation.

RSA Menu

In RSA Menu, the user will be showed how RSA works in a scenario such that any listener can not get the encrypted the message, except for the receiver. In RSA Menu, the user can input prime number 1 and 2 and the integer message. First, n and its phi function are calculated. Afterwards, e and d are calculated, with all the equations being shown. The public keys are then sent to the sender, who calculates the message and sends the encrypted message to the receiver, who decrypts it. This shows how RSA generally works in real life and allows students to learn effectively due to the animations.

Internalisation

In the About AlgoHolic section of the application, the Version and Created By is translated into three languages: English, German and French. The user can change the language using a combo box. This is so that people from other languages can use the app. Translations are stored in .properties files. This feature is still a work in progress for the rest of the pages.

Code Explanation

Sorting Animation

For the Sorting Animation, AnimationController.java was coded. In it are two classes, Bar, which is a slightly modified Rectangle that has a colour property, and AnimationController which initialises the Bars and does the animation. It contains a method called barMaker() which creates the Bars. It also calls calculateHeight(int value, double y), which calculates the height of the Bars using value/max \* y (height of max bar). The apply(Object object) method controls the animation seen. It first goes through the sorting list gets the value of the bar and the colour of the value of the bar. A timeline animation is then created to change the colour of the bar and if the bar has a different height, it does the bar height animation. The code for the Sorting Algorithms is all separately in their own classes.

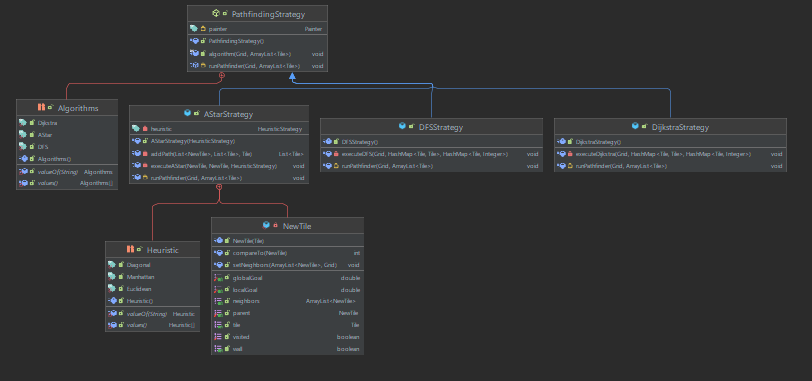
Pathfinding Animation

For the Pathfinding Animation, Painter.java is used. In it there is one class, Painter. The class has a drawTile() method that colours the Tile depending on its Type. The clearPath() method clears the path by changing the Tile.Types to Empty, which has no colour. Finally, there is drawPath() that gets the Path List and changes the Tiles in the list to Path. I separately coded this class so if I wanted to edit how the colour changing of the Tile worked, I did not need to explore much in my Tile, or Grid Class, and could easily edit here. The code for the Pathfinding Algorithms is all separately in their own classes.

RSA Animation

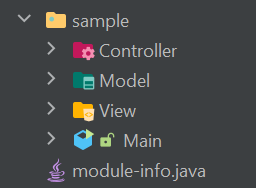
For the RSA Animation, I used a thread in RSAController.java. I have separately coded RSA in RSA.java and extensively used it in the thread. The thread is located in the method runBtnOnClick() which is linked to the runBtn in SceneBuilder. Firstly, in the thread, I start making multiple labels visible. These labels show the public and private keys of the receiver. Afterwards, I calculate multiple values, and update the labels with these values using Platform.runLater(). Afterwards, I made multiple FadeTransition for the various labels that I previously made visible. I made a ParallelTransition that contains the FadeTransitions. I then played the ParallelTransition. Afterwards, only the public key labels can be seen. I made copies of these labels using Texts. Then, I created some TranslateTransitions in a ParallelTransition to make the labels move to sender and also the listener. A created another label and set it with the encrypted message, that was encrypted using the public keys. I made copies of the label using a Text. Then, I created some TranslateTransitions in a ParallelTransition to make the labels move to receiver and the listener.

OOP Design



Text

Description automatically generated with medium confidenceThis is the UML of the Pathfinding Strategy. PathfindingStrategy is an abstract Parent class. This is because all the pathfinding strategies have similar methods but not similar code in the method, hence the parent class is abstract. Concepts like Inheritance and Polymorphism was important due to the similarity shown in the UML. Many methods, classes, and variables were used, hence showing the usage of Abstraction. Many methods such as addPath in AStar are also private, showing Encapsulation. The full UML can be seen at the right.

MVC Design

There is a CSS file, Images and multiple FXMLs in View. Each controller in View has its respective Controller in the Controller package. All the Algorithms are in Model, along with Painter.java and the persistent storage files. Model also contains a StrategyFactory that helps me for the ComboBoxes found in the Pathfinding Visualizer.

Testing

Individual Testing

Firstly, I coded each area of the application separately and started testing them separately. First, I would see of each of the sections have minimal to no bugs. I believe I effectively cleared all the bugs I could find. Then, I would test to see if the visualisers were fun for me. Afterwards, I would edit the design so it became more fun. After testing them separately, I did multiple complete testing with everything being completed. This allowed me to find some bugs that could occur when an animation was played and the user tried to change scenes.

Random User Testing

I gave my computer to multiple of my friends and took their feedback on the design of the application, the how fun the application was, and if they learnt anything from the application. Most, if not all those random users stated that they learnt something from the application, and I listened to those who said otherwise, and slightly changed the UI how they wanted. I also asked multiple questions to the users after they used the application to test them, and most could muster a sufficient answer. As certain codes could have more comments, I made them into persistent storage so I can edit them to make it much better.

Reflection

Obstacles Faced

I faced multiple obstacles at the start of the project. This was because I did not know how to effectively design the Model and Animation to make it look good. I also had trouble in the Hashing UI for Affine and Caesar Cipher, as I had a different UI planned, but most of the random user testers wanted to change it to the current UI, but others still disagree and prefer the old UI. Another challenge faced was learning how the algorithms worked and how to code it after learning it.

Learning Points

I learnt that I must think more carefully for my Model than I initially thought as I had to refer to another source for how to design the Grid, especially the update and gridInit method in Grid, however, the other methods were coded by myself. I also learnt about the enum type, which I found out by referring to the source and afterwards GeekforGeeks. I also felt I learnt a lot about animations from the RSA, and Sorting Animation parts and also about threads from my Hashing Animation.

Improvements to Application

An improvement I could make was making the internalization feature affect more text. I could also add a button to change the UI of my ciphers. I could also add a Light mode feature for those who prefer having that design.

Improvements to Task

The task was fun. However, I felt that we could have more time to experiment more for our project. I also felt that the task could be more enjoyable if there would be milestone checks so we could make a better final product.