National College of Ireland

Computing Project

Software Development

<Academic Year i.e. 2020/2021>

Christopher Tobaras

X20324573

X20324573@student.ncirl.ie

Rail-AI-Leader

Technical Report

Contents

Executive Summary

The aim of this technical report is to document and the requirements and development of a signalling tool that can be used enhance safety, optimize the flow of traffic and further research the idea of automating railway traffic. The report explores the technology used throughout the development process, which includes traversal algorithms, GUI interfaces, and scheduling of threads. The architecture of the program is also discussed, which consists of a Model-View-Controller style interface, displays a track layout which follows a transactional rules-based approach for the generation, and movement of trains.

The report describes the functional and other requirements of the application, following with the implementation of the key features which meet these requirements. The graphical interface, pathfinding algorithm and transactional model will be described in detail, utilizing screenshots and snippets of code to demonstrate the functionality. The report discusses the challenges encountered and resolved throughout the duration of development and demonstrates the usage of technologies such as GitHub, JavaFX, and IntelliJ to meet the requirements set forth for the program.
After the development section, the app is then tested and evaluated to see if the final product meets the requirements. The report afterwards will discuss features that could be considered if further development was possible. The report concludes that the application could present a viable need for automating railway traffic, however, needs further time to implement more features that would improve the performance, flexibility, and impact of the application.

1.0 Introduction

1.1. Background

The idea to research and implement an application for this project came from reality that the current age is leveraging concepts such as machine learning and advanced intelligence to create solutions that perform much more efficiently and faster than traditional means.[1]

Topics such as data analysis, automation of systems and predictive analytics benefit from machine learning algorithms as humans struggle in this task, especially with an exponential rise in the size of data. [1][4]

An idea that has been theorised with these algorithms is the possibility of implementing them into traffic management, which would see the use of machine learning algorithms control a large traffic network, providing predictive traffic flow monitoring, coordinated signalization, path access control and incident detection. [2]

In the pursuit of optimising railway operations and exploring the possibilities of using emerging technologies, this project was created to pair the two together in the form of an application, allowing automation of movements into railway signalling operations. The Rail-AI-Leader application would be a modular application, taking a similar form to modern signalling software, but allow for on-demand automation of train operations set by the signaller, in order to reduce the workload of traffic management.

The reason for why I undertook this project is that the transportation sector is a very competitive one, and developing an application that utilizes modern technologies can make this form of transport more relevant in this age of tech. Creating an application where machine learning could be used to promote easier to use, safer and efficient methods of operation would massively benefit the way railways operate.

1.2. Aims

The central aim of the project is to research and implement an application which can utilize an automated system to augment the movement of railway traffic. By providing a safer and much more efficient mode of moving trains, this would help ease the workflow of the signaller, which could liberate time for them to perform other duties.

The graphical interface of the application should be able to effectively show a layout which visualises the movement of trains on that layout. Combined with a menu, it will allow the user to start/stop the automation, and import/export information which can be analysed by administrative staff.

The application aims to safely automate railway traffic according to how railway operations are conducted in Ireland.[3] Trains should be able to move around the layout without being in danger of colliding with another train. The application must be able to follow these rules regardless of the algorithm it is using to automate train movements.

Overall, the app aims to help ease the job of a signaller by automatically moving trains to their destination while following the safety procedures outlined by Iarnrod Eireann.[3]

1.3. Technology

To develop this application, I am using Java as my language alongside IntelliJ, and GitHub.

IntelliJ is a popular choice for Integrated Development Environments as it contains support for multiple java-based applications, such as Maven and Gradle projects. It also contains built in version control system which are compatible with GitHub, allowing me to quickly save and compare different versions of my application. It also contains support for unit testing and an integrated debugger, which is an excellent tool that helped me with debugging the applications features.

Maven is used as the build automation tool for the application, due to its simplified dependency management, rich plugin ecosystem and extensive documentation. As I implemented JavaFX and JGraphX into my project, Maven proved useful in implementing these libraries, alongside its Combine, Test, deploy cycle which allowed me to test a program before building it successfully.

JavaFX is the graphing library used to draw the window of the application alongside its mapping and import feature. Using the Model-View-Controller cycle, it provides a wide range of UI components, alongside dynamic customization, which helped me introduce a less visually intrusive dark mode to the application. It also contained support for IntelliJ's SceneBuilder plugin, which allowed the simplified creation of the application's visual elements.

JGraphX is a port of the popular MxGraph 5 library for JavaScript, giving applications the ability to create and visualise graph-based diagrams and layouts. Using its API support, modifications to the layouts customisation and data can be done in the form of transactions, alongside support for extensive customization of edges and vertexes. Its support for JavaFX programs proved much useful in visualising train movements.

Finally, Junit5 is the testing framework used for the unit, system and integration testing of the application. It is praised for its modular architecture, parameterized tests, and integration with IntelliJ, which proved handy in quickly implementing unit tests to test modular functions.

1.4. Structure

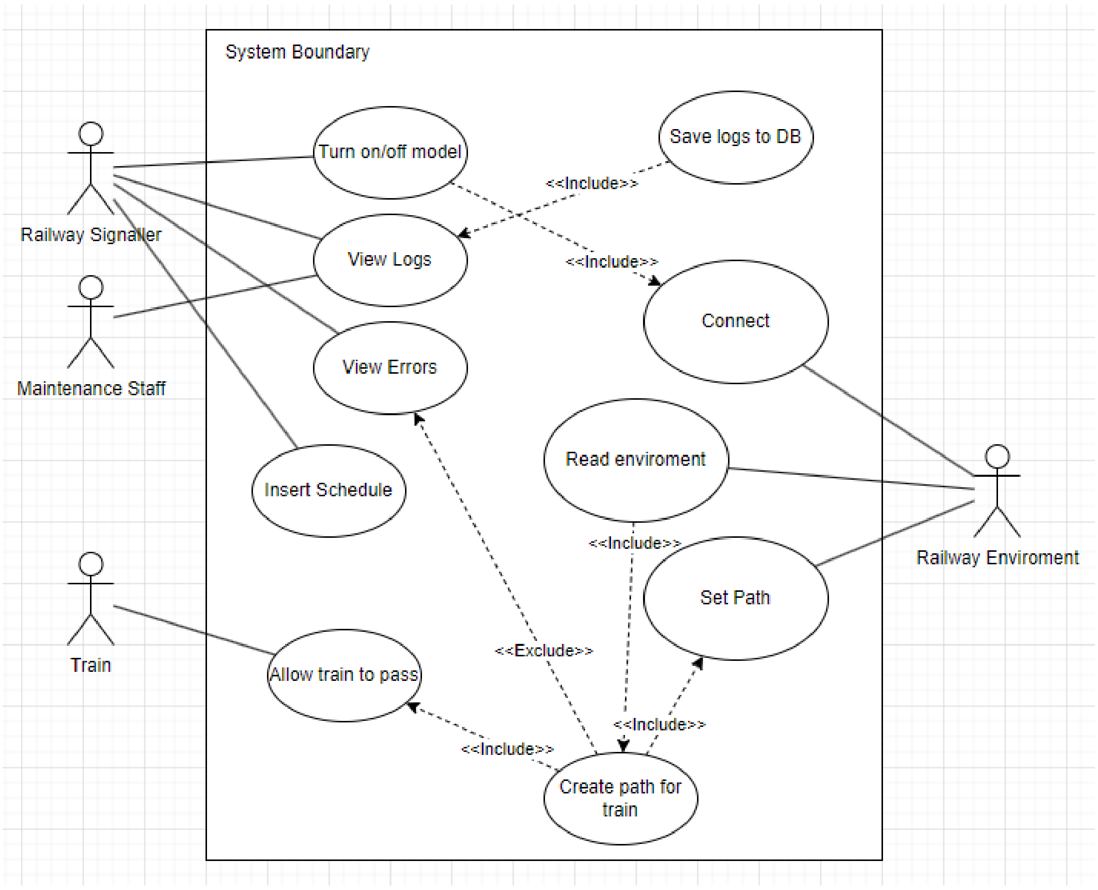
The structure of this document is divided into four main parts: the Introduction, the System, the Testing, and the Conclusion. The introduction already discussed, talks about the background, motivation and technology used for the project. The System Implementation is the biggest part of the report as it extensively covers all the requirements of the application, the design and development, the User interface(And associating wireframes) and the implementation of tests. The testing section will discuss the testing methods utilized to evaluate the 6performance and functionality of the finished system, and the final section will discuss the products impact, and potential future development.

In each section, screenshots and snippets of text will be used to give a better understanding of the various design decisions and implementations of features.

2.0 System

2.1. Requirements

2.1.1. Functional Requirements



R-AI-L Figure 1: Use Case Diagram

2.1.1.1. Requirement 1 – Direct train to destination

Name	Move a train to their destination
Use Case	UC-1
Description	This is a use case for the application setting a path for the train
Priority	High
Precondition	The system is turned on and is engaged. There are trains that are on the layout.
Trigger	A train appears on the layout
Flow	1. The application reads the schedule and selects the latest one. 2. The application starts finding a path for the train from its origin to its destination. 3. The application checks if the path conflicts with paths already set. 4. The application locks the path, tells the train to move ahead. 5. The application then releases the path once the train is at its destination
Alternate Flow	Alternate Flow 1: Current path conflicts with another locked path 1. The path given to the train conflicts with a path already set previously and is still in use. 2. The system saves the path and does not set it. 3. The path is saved to the train and tries again in a set amount of time.
Exceptional Flow	Exceptional Flow 1: Destination is not reachable. 1. The destination for the train cannot be reached from its point of origin. 2. Warn the user with an error stating the train number, and the pathing error.
Includes	
Extends	UC-2 and UC-3
Postcondition	The train has arrived at its destination.

2.1.1.2. Requirement 2 – Engaging the System

Name	Turning on the system
Use Case	UC-2
Description	This is a use case for enabling the automatic movement by the application
Priority	High
Precondition	The system is turned off and the system is disengaged
Trigger	The user has opened the program
Flow	1. The user navigates to the application and presses on the start button. 2. The system turns on and starts checking for trains.
Alternate Flow	
Exceptional Flow	
Includes	
Extends	
Postcondition	The system is enabled and is moving trains automatically.

2.1.1.3. Requirement 3 – Disengaging the System

Name	Turning off the system
Use Case	UC-3
Description	This is a use case for disabling the automatic movement of the application
Priority	High
Precondition	The system is turned on and the system is engaged
Trigger	The user has decided to stop the application
Flow	1. The user navigates to the application and presses on the stop button. 2. The system stops checking for new trains to automatically move.
Alternate Flow	
Exceptional Flow	
Includes	
Extends	US-2
Postcondition	The system is disabled and its now inactive.

2.1.1.4. Requirement 4 – Accepting Schedules

Name	Giving the application a Schedule

Use Case	UC-4
Description	This is a use case for entering a Schedule for the app to use
Priority	High
Precondition	The application is open. The user has a CSV file filled with train data for the application.
Trigger	The user has pressed the Insert Schedule button
Flow	1. The user navigates to the application and presses on the Insert Schedule button. 2. A popup is shown that allows the user to insert a CSV file of the schedules. 3. The file is then accepted with a visual prompt telling the user that the Schedule is updated
Alternate Flow	Alternate Flow 1: Schedule data error 1. Data inside the CSV file does not match the format needed. 2. Discard the data and alert the user to the correct format.
Exceptional Flow	Exceptional Flow 1: Incorrect Format 1. The user sends a file that is not in CSV format. 2. The system will bring up an error and notify the user that the formatted type is not accepted.
Includes	
Extends	
Postcondition	The system now has a Schedule and is adding trains to the layout from the new schedule.

2.1.1.5. Requirement 5 – Viewing/Exporting Logs

Name	Viewing/Exporting the system logs
Use Case	UC-6
Description	This is a use case for allowing the user to view, and export logs in a CSV format
Priority	High
Precondition	Errors have been made during the applications duration of operation.
Trigger	The user has pressed the View logs button.
Flow	1. The user navigates to the application and presses the “View Logs” button. 2. The system presents a popup which contains the logged actions which it has done since the system was started. 3. The user then presses the “Export to CSV” button. 4. The system notifies the user that the logs have been exported to a CSV file in the base directory.
Alternate Flow	
Exceptional Flow	
Includes	
Extends	
Postcondition	The system has exported the logs and continues as normal.

2.1.1.6. Requirement 6 – Viewing Performance Metrics

Name	Viewing the applications performance
Use Case	UC-7
Description	This is a use case for allowing the user to view and export the performance metrics in a CSV format
Priority	High
Precondition	The system is turned on and the system is engaged. At least one train has been moved, saving the data in the program.
Trigger	The user has pressed the “Debug” button.
Flow	1. The user navigates to the application and presses the “Debug” button. 2. The system presents a popup which contains the performance metrics of the algorithm. 3. The user then presses the “Export to CSV” button. 4. The system notifies the user that the logs have been exported to a CSV file in the base directory.
Alternate Flow	
Exceptional Flow	
Includes	
Extends	
Postcondition	The system has exported the metrics and continues as normal.

2.1.2. Data Requirements

Here are the data requirements necessary for the successful operation of the program from the user.

1. <u>Train Data</u>

The application provides the track layout, however, does not provide any trains on layout. To move trains, it must be supplied with data in order to first generate and visualise the trains on the layout. The data needed to create the trains requires information such as the Name, Origin, Destination, Appearance, and Arrival Time. This data is required to be in a CSV format, where each row contains the appropriate information in the stated order.

2. <u>Log Data</u>

The application stores debug information about the program and pathfinding algorithms actions, which contain information such as Train Created, Train Added, Path Found for Train, and Train Moving, alongside Errors such as Train Generation errors, Path not found, and Movements blocked. These logs will also have a timestamp beside them as to when they were logged.

3. <u>Performance Data</u>

The application stores information such as the difference between the current time the train arrived, compared to the Arrival Time contained in the Train data. The time taken to create a path for a train will also be stored.

2.1.3. User Requirements

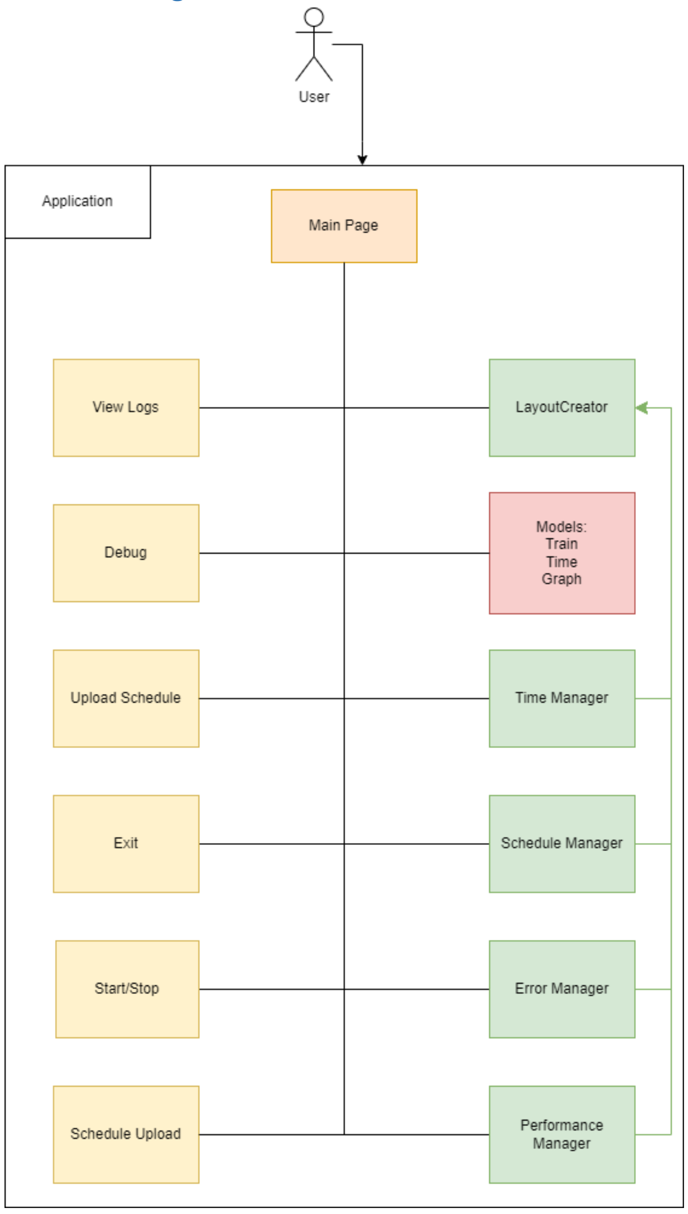
The Rail-AI-Leader application is designed to be as easy to understand as possible for experienced and novice users. The application should have functions such as window views clearly visible and scalable to match the users' preferences. The track layout should be able to be made bigger, in order to accommodate the full size of it on a normal monitor screen. The main menu should be placed strategically in an easy to reach position and display all functionality to the user.

2.1.4. Environmental Requirements

<u>Operating System</u>

The Rail-AI-Leader application is compatible with Windows 10 and above and requires a CPU with more than 8 threads to run optimally.

2.2. Design & Architecture

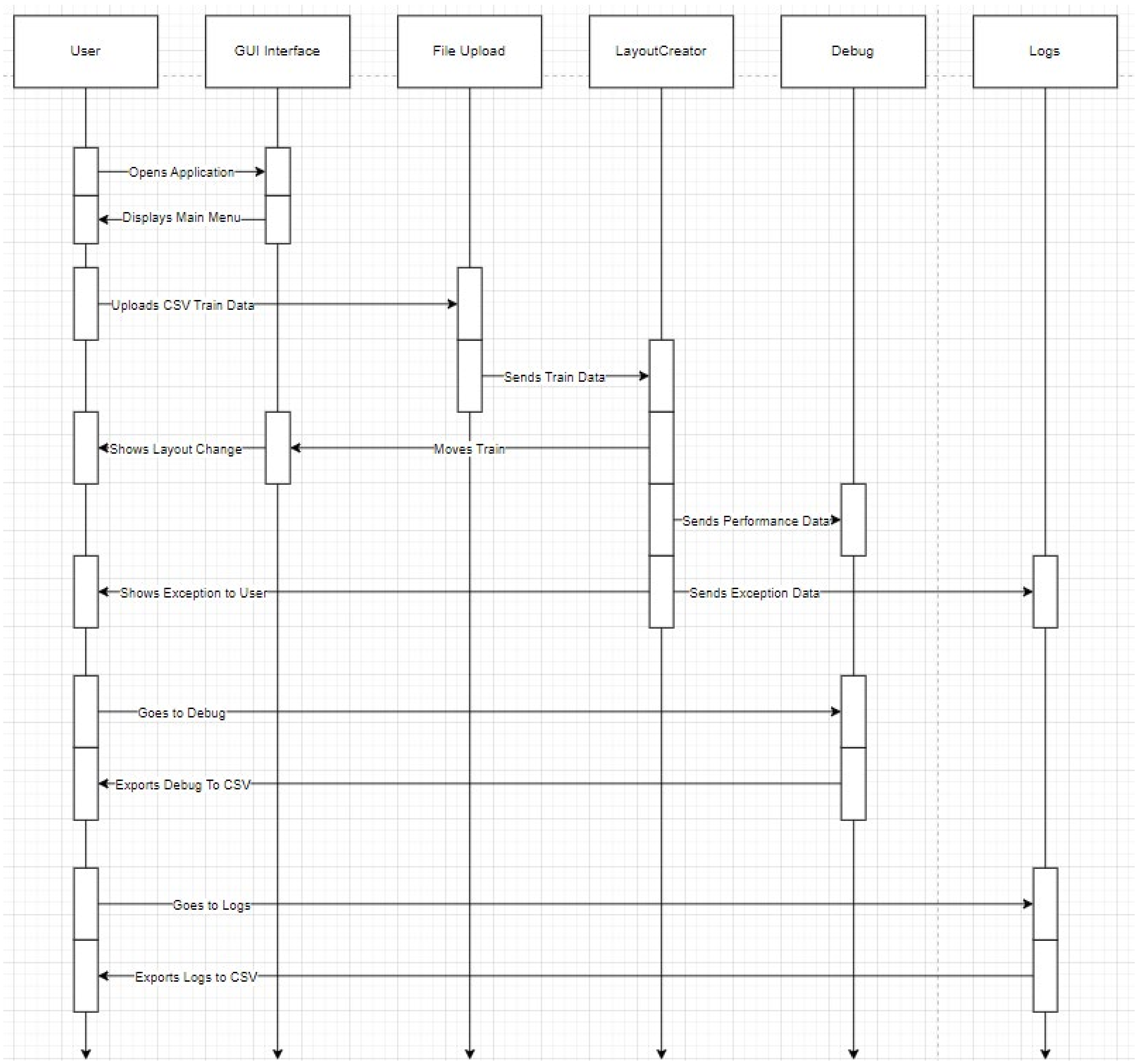


R-AI-L Figure 2: Architecture Diagram

The application is supposed to automate train movements, while also being similar in design with Train Control Systems. This means that the architecture and systems commonly used in these types of software are necessary for the basic functions of the program.

The Rail-AI-Leader application is a standalone application that relies on a number of critical systems in order to achieve its desired functionality. The first component is the Graphical User Interface, utilizing JavaFX which provides a user-friendly interface for the operator to interact with the system. This also allows the user to interact with the layout display and observe the trains position. This component follows a Model-View-Controller architecture, separating the logic used for the presentation of the graph from the business logic of the movement system and other components. The controllers are located in the java directory present as Java files, the view pages are located inside the resource's directory as FXML files, and the Model present in as the NodeFactory package.

Other components used in the app include the Layout Model, and the Train Controller. With JGraphX, the Layout Model is created and populated with information given by the Train Controller. This information is first fed to the system using the File Upload system, which is then generated then stored inside the application.

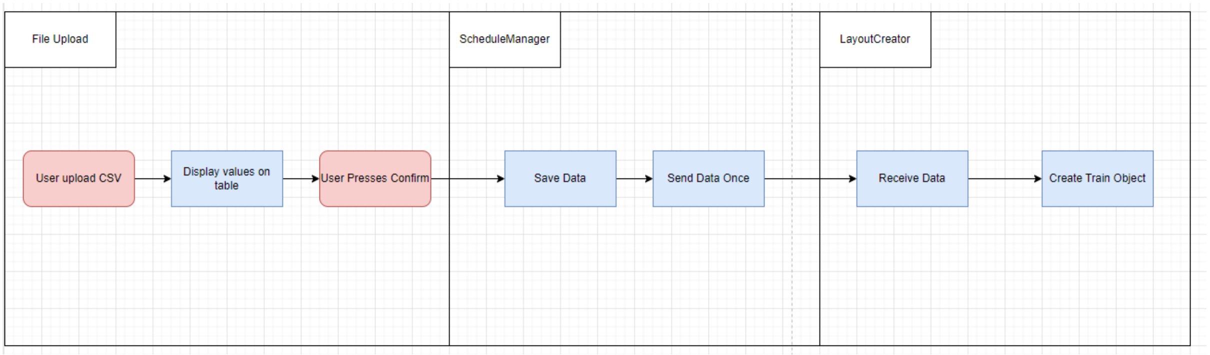


R-AI-L Figure 3: Architecture Flow Diagram

The primary algorithms of the app are the layout creation, the file upload system, the train controller and finally exception and performance handling.

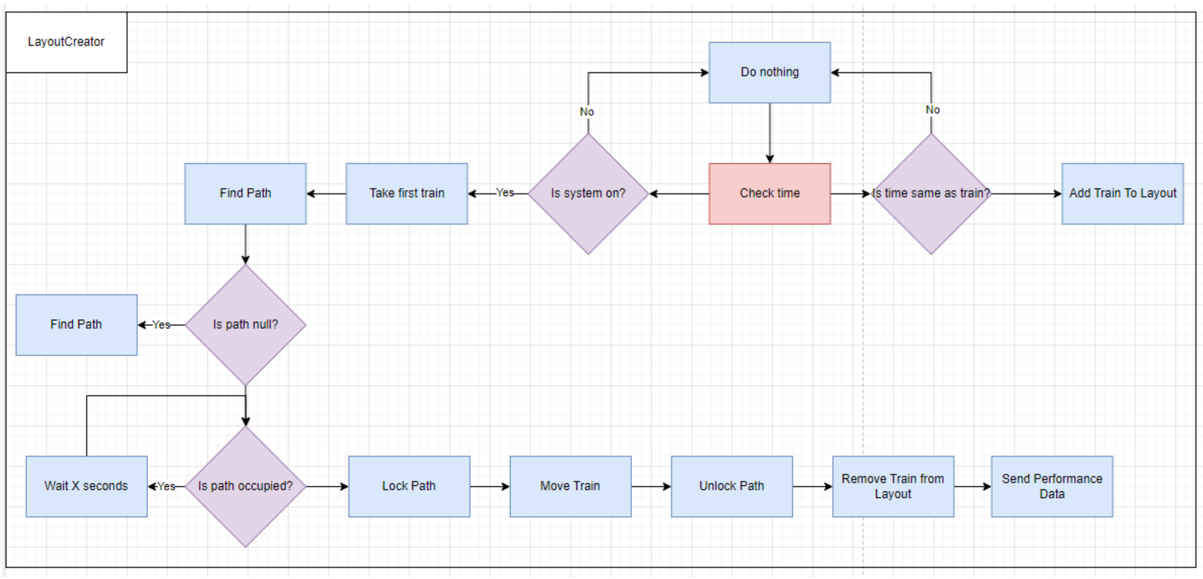
The GUI display is initialised upon running the application. It opens a main window, which displays the layout view in the centre of the screen, initially empty. A menu bar is positioned at the bottom of the screen providing the user with the necessary functions to operate the application, such as starting and stopping the program, accessing logs, uploading schedules, and entering the settings menu. The GUI display will display the layout created during the layout creation system, and any changes by the train controller will be displayed for the user to observe.

The layout creation system is responsible for creating the initial layout used for simulating trains. It uses JGraphX to create a graph model and initiate a transaction with the graph. In this transaction, vertexes are added to the graph with a set size, associated colour, and position on the graph. Edges are then added, connecting the vertexes together. Afterwards, the transaction is closed, and the graph is updated in the GUI display, showing the user the layout.



R-AI-L Figure 4: User Data Flow.

The file upload and export system are a very important feature for R-AI-L, as it allows the user to upload a file containing trains that will be automatically moved within the app. The user is able to input a CSV file, on which a table in the Schedule window will be filled with the user's data, allowing the user to verify the data is correct before confirming. Upon confirmation, the data is then sent to the train controller, for processing before being put on to the layout.



R-AI-L Figure 5: Train Flow Diagram.

The train controller system is the largest system in the application, compromising a variety of smaller subsystems included to provide the full functionality of train movements. The train controller receives and stores any trains from the file upload system and waits until the scheduled time of the train matches the current time. Once the current time is met, the train is positioned on the layout, and is now marked active. A pathfinding algorithm is then used to determine a path between the origin and destination of the train, and a locking mechanism is used to check and secure the path if possible. Once these conditions are met, the train is then moved by the train controller through the layout to its destination, and the path is then unlocked.

Overall, the R-AI-L application compromises a variety of systems with each serving a function needed to complete the function of the application.

2.3. Implementation

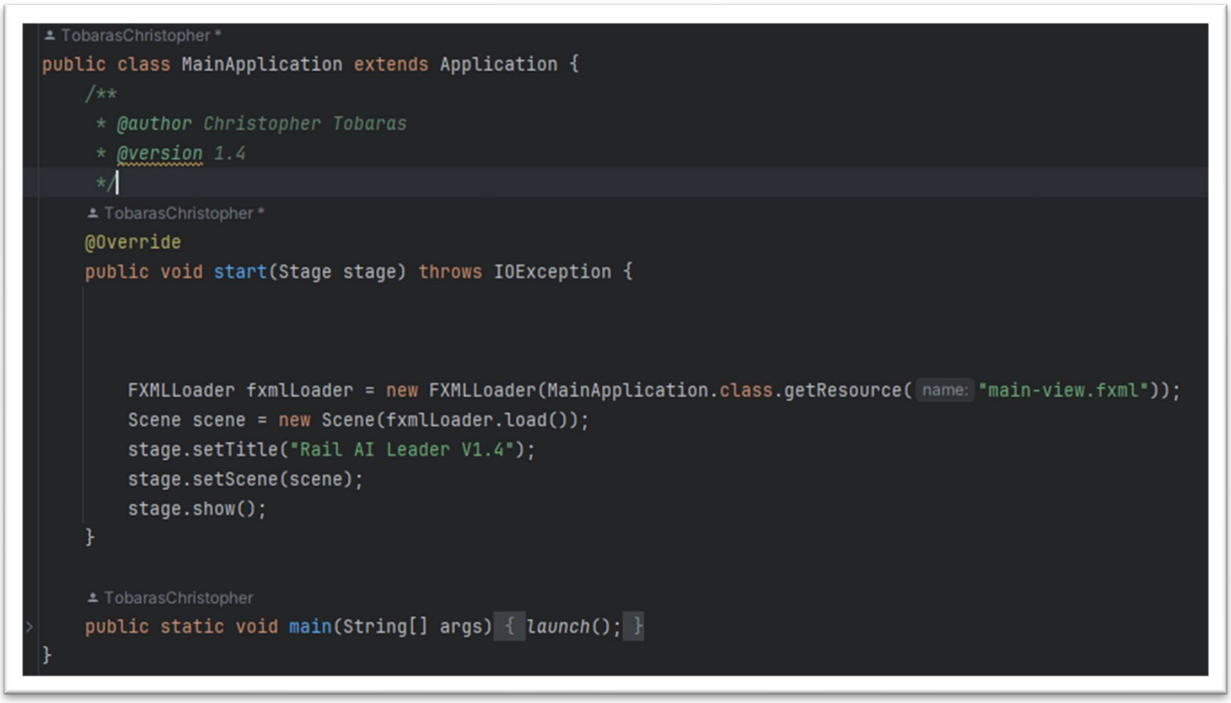
This section of the report will dive into the implementation of the components in the Rail-AI-Leader application. Supported by screenshots and snippets of code, this section will go over the main logic utilized in each component, divided into their own individual section.

The implementation will first focus on the GUI Display of the program, as it forms the necessary base for our next features. JavaFX plays an important role in the application, providing a solid architecture for us to build later components on.

Other tools used in this development include Junit5 for the testing framework, and IntelliJ's built in "Scene Builder" plugin for easier development of JavaFX elements.

2.3.1. Graphics User Interface Display

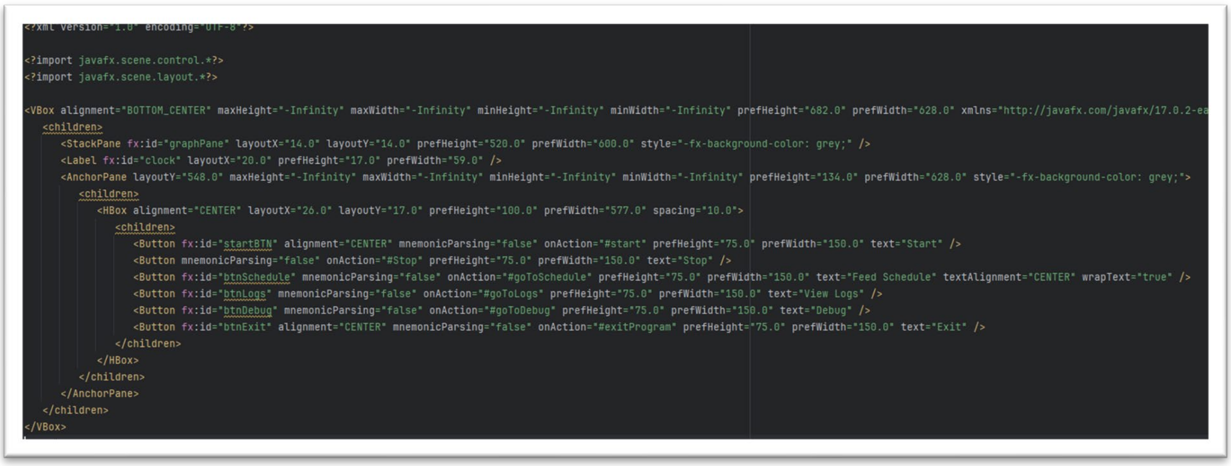
The Main Application class serves as the starting point for the JavaFX application, as it creates and displays the main view window, along with its associated controller. This is required by the JavaFX library to provide a structured entry point into the program.



```
TobarasChristopher *
public class MainApplication extends Application {
    /**
     * @author Christopher Tobaras
     * @version 1.4
     */
    TobarasChristopher *
    @Override
    public void start(Stage stage) throws IOException {



        FXMLLoader fxmlLoader = new FXMLLoader(MainApplication.class.getResource( name: "main-view.fxml"));
        Scene scene = new Scene(fxmlLoader.load());
        stage.setTitle("Rail AI Leader V1.4");
        stage.setScene(scene);
        stage.show();
    }

    TobarasChristopher
    public static void main(String[] args) { launch(); }
}
```

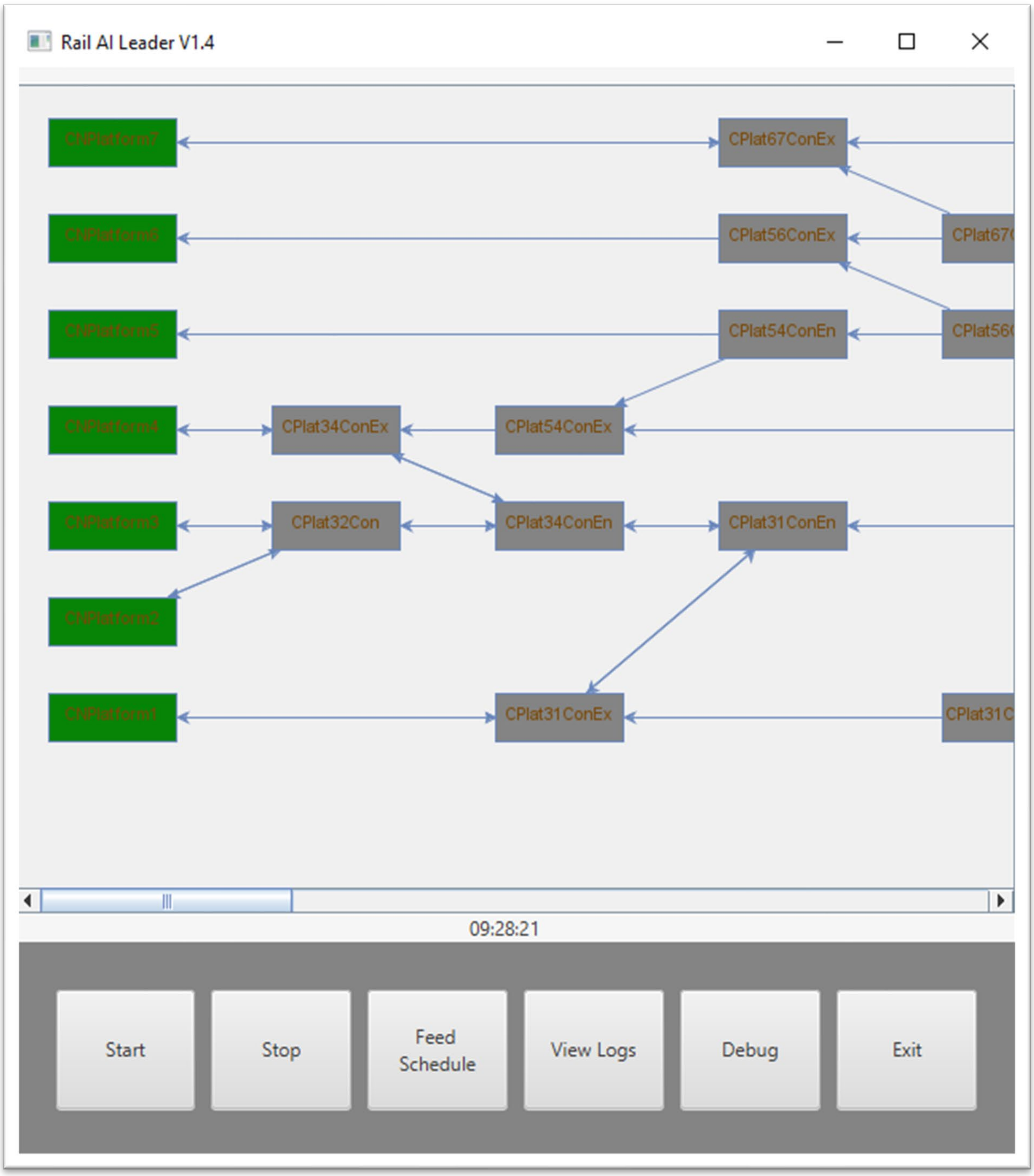
R-AI-L Implementation 1: Menu Start Function.

The code screenshot above shows how the application is first launched into the main view by using the Main Application method. It extends the application class, which is a process used to create an entry point into a JavaFX application. Afterwards, it acquires the “main-view.fxml” resource, which then sets and displays the window using the resource.

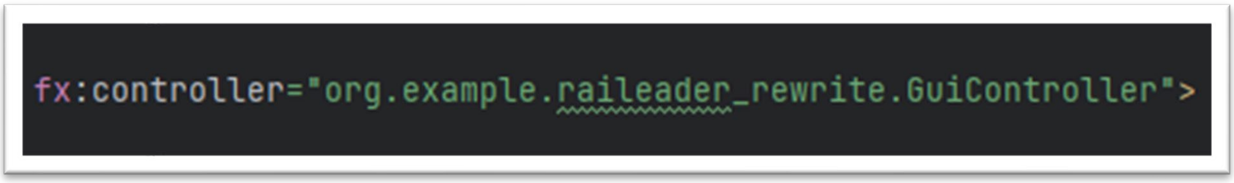


```xml
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<VBox alignment="BOTTOM_CENTER" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="682.0" prefWidth="628.0" xmlns="http://javafx.com/javafx/17.0.2-ea
   <children>
      <StackPane fx:id="graphPane" layoutX="14.0" layoutY="14.0" prefHeight="520.0" prefWidth="600.0" style="-fx-background-color: grey;" />
      <Label fx:id="clock" layoutX="20.0" prefHeight="17.0" prefWidth="59.0" />
      <AnchorPane layoutY="548.0" maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity" prefHeight="134.0" prefWidth="628.0" style="-fx-background-color: grey;">
         <children>
            <HBox alignment="CENTER" layoutX="26.0" layoutY="17.0" prefHeight="100.0" prefWidth="577.0" spacing="10.0">
               <children>
                  <Button fx:id="startBTN" alignment="CENTER" mnemonicParsing="false" onAction="#start" prefHeight="75.0" prefWidth="150.0" text="Start" />
                  <Button mnemonicParsing="false" onAction="#Stop" prefHeight="75.0" prefWidth="150.0" text="Stop" />
                  <Button fx:id="btnSchedule" mnemonicParsing="false" onAction="#goToSchedule" prefHeight="75.0" prefWidth="150.0" text="Feed Schedule" textAlignment="CENTER" wrapText="true" />
                  <Button fx:id="btnLogs" mnemonicParsing="false" onAction="#goToLogs" prefHeight="75.0" prefWidth="150.0" text="View Logs" />
                  <Button fx:id="btnDebug" mnemonicParsing="false" onAction="#goToDebug" prefHeight="75.0" prefWidth="150.0" text="Debug" />
                  <Button fx:id="btnExit" alignment="CENTER" mnemonicParsing="false" onAction="#exitProgram" prefHeight="75.0" prefWidth="150.0" text="Exit" />
               </children>
            </HBox>
         </children>
      </AnchorPane>
   </children>
</VBox>
```

R-AI-L Implementation 2: FXML File of Main Menu.



R-AI-L Implementation 3: Main Menu GUI.

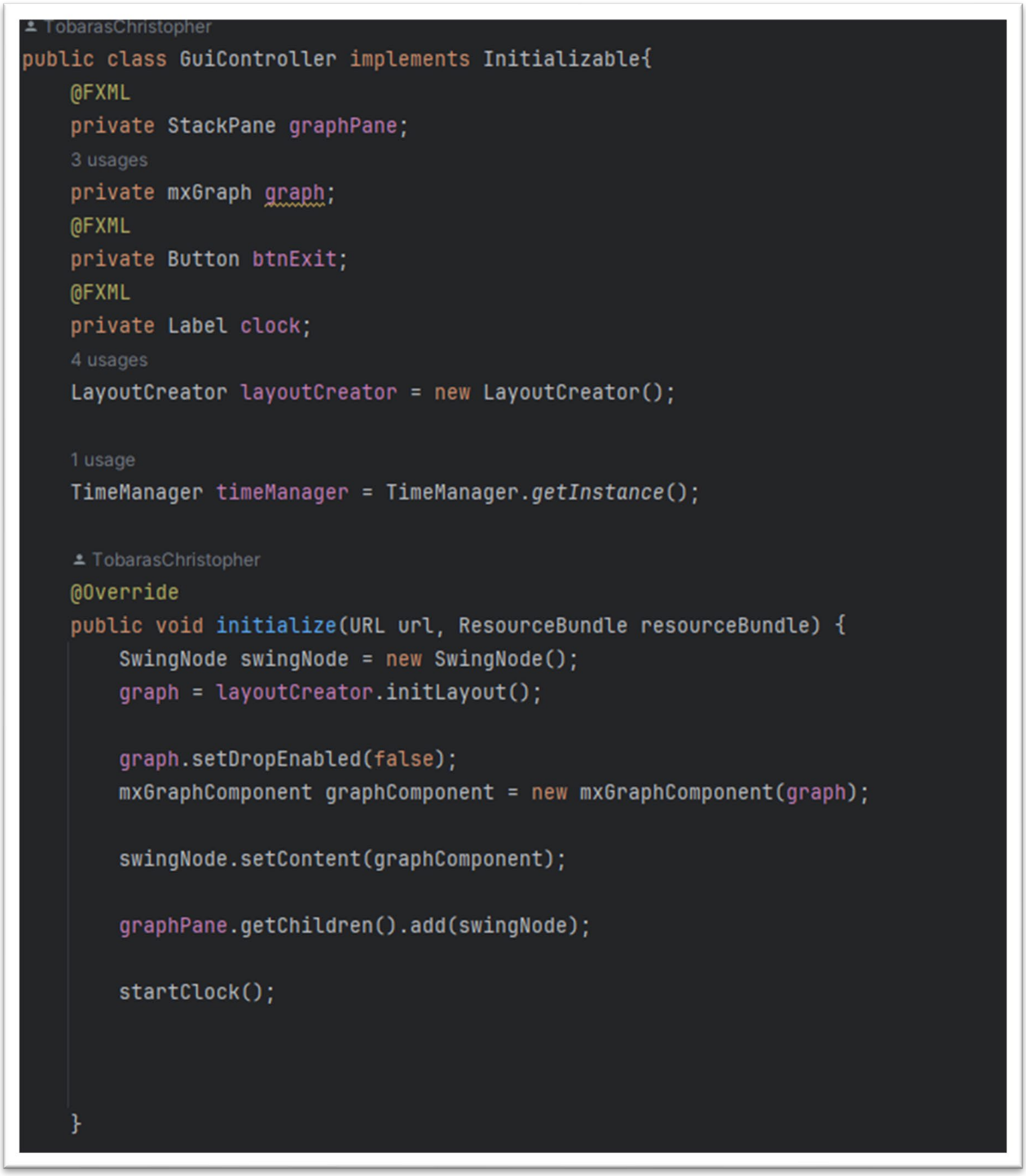


```
fx:controller="org.example.raileader_rewrite.GuiController">
```

R-AI-L Implementation 4: Controller Appendage to FXML Resource.

The above code is the main view that is displayed for the application. FXML files are formatted in XML, which can also be edited in IntelliJ's built in SceneBuilder. Buttons for the user here are created, alongside the graph element and label for a clock, which will be explained later. The root VBox element is stylised in that it can be scaled horizontally at any resolution and contains the controller class needed to utilize the methods specified by the buttons and graph elements.

The GUIController is responsible for providing the displayed main view logic, and data which can be modified on the applications display. It provides the user the ability to open different windows, view a clock implemented to view the time and displays the main graph which is used to show the track layout.



```
TobarasChristopher
public class GuiController implements Initializable{
    @FXML
    private StackPane graphPane;
    3 usages
    private mxGraph graph;
    @FXML
    private Button btnExit;
    @FXML
    private Label clock;
    4 usages
    LayoutCreator layoutCreator = new LayoutCreator();

    1 usage
    TimeManager timeManager = TimeManager.getInstance();

    TobarasChristopher
    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        SwingNode swingNode = new SwingNode();
        graph = layoutCreator.initLayout();

        graph.setDropEnabled(false);
        mxGraphComponent graphComponent = new mxGraphComponent(graph);

        swingNode.setContent(graphComponent);

        graphPane.getChildren().add(swingNode);

        startClock();


    }
```

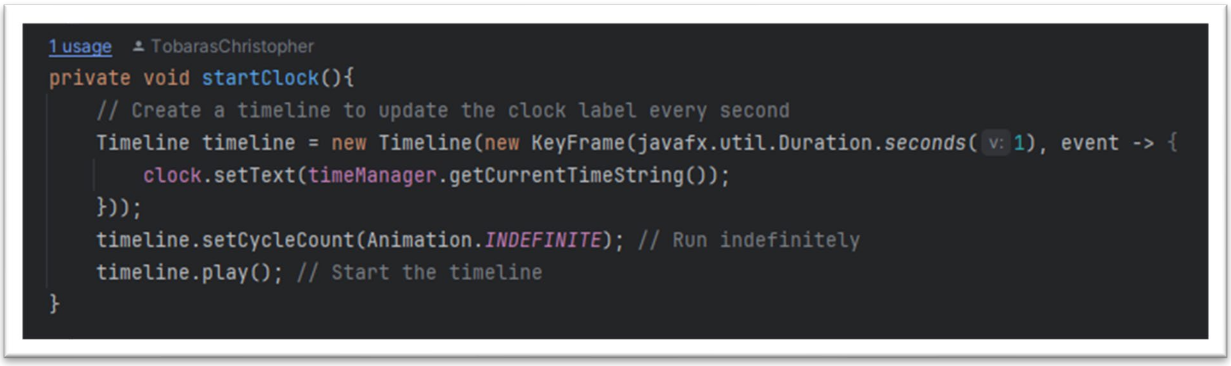
R-AI-L Implementation 5: Initialise Code of GUIController class.

The code shown above shows the initialisation of a few FXML elements in the application logic, alongside the creation of a graph pane and clock element, with the graph itself. A new object is created out of a class called LayoutCreator, which is where the layout creation and train control features are located. Another object is acquired from the TimeManager class, which is a singleton class designed to provide the time.

The initialize method does a couple of things in order to display the graph on the GUI display:

- It creates a Swing Node for the graph object to be laid on to FXML elements.
- Creates the graph by using the initLayout() method by the LayoutCreator class.
- Embeds the graph into the Swing Node, and then adds it to the graphPane element.
- Runs the startClock() method.

The startClock() method uses an animation Timeline which periodically updates every second to set the clock element to the current time provided by the TimeManager.

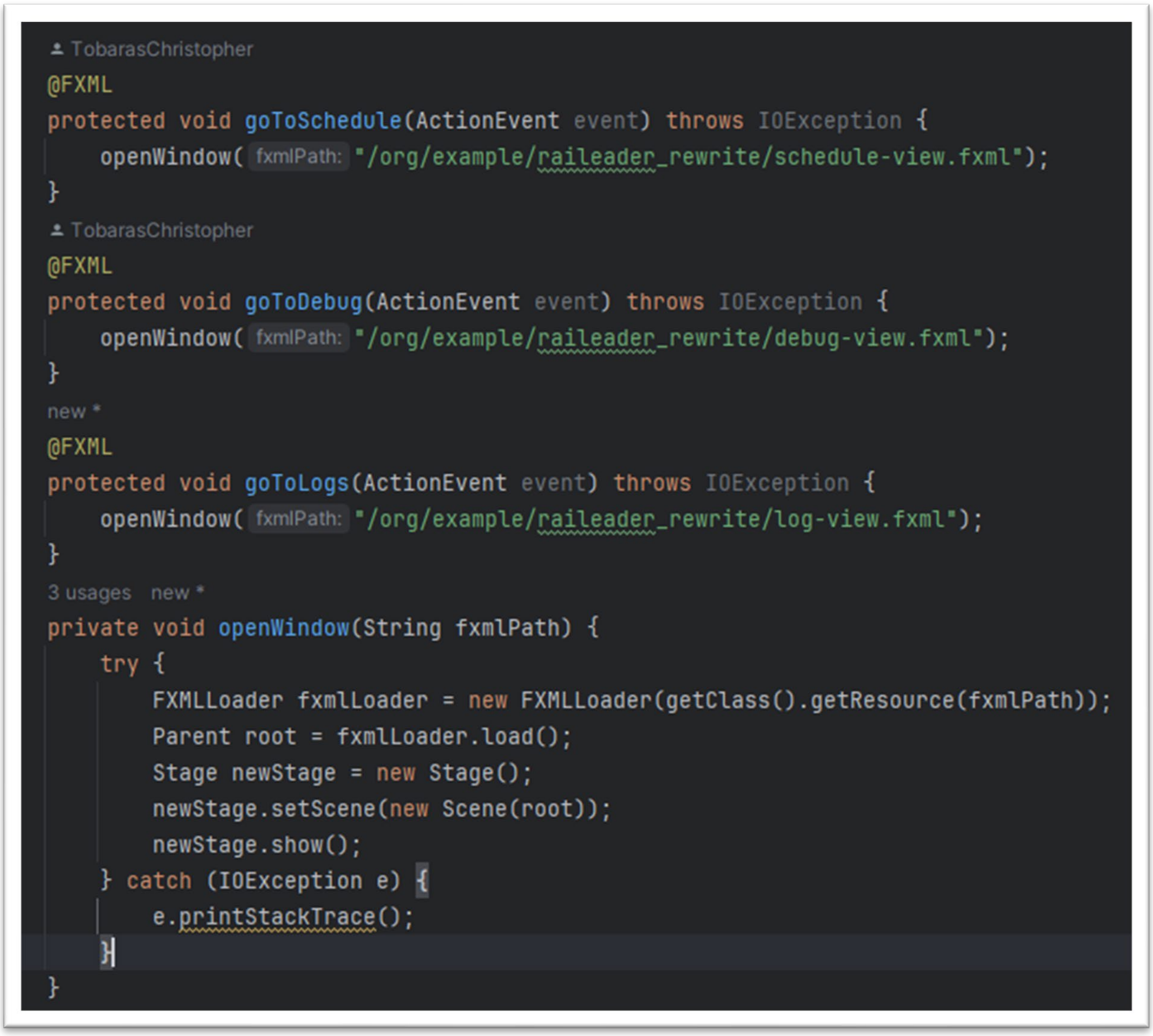


```
1 usage  TobarasChristopher
private void startClock(){
    // Create a timeline to update the clock label every second
    Timeline timeline = new Timeline(new KeyFrame(javafx.util.Duration.seconds( v: 1), event -> {
        clock.setText(timeManager.getCurrentTimeString());
    }));
    timeline.setCycleCount(Animation.INDEFINITE); // Run indefinitely
    timeline.play(); // Start the timeline
}
```

R-AI-L Implementation 6: Simple Clock Function

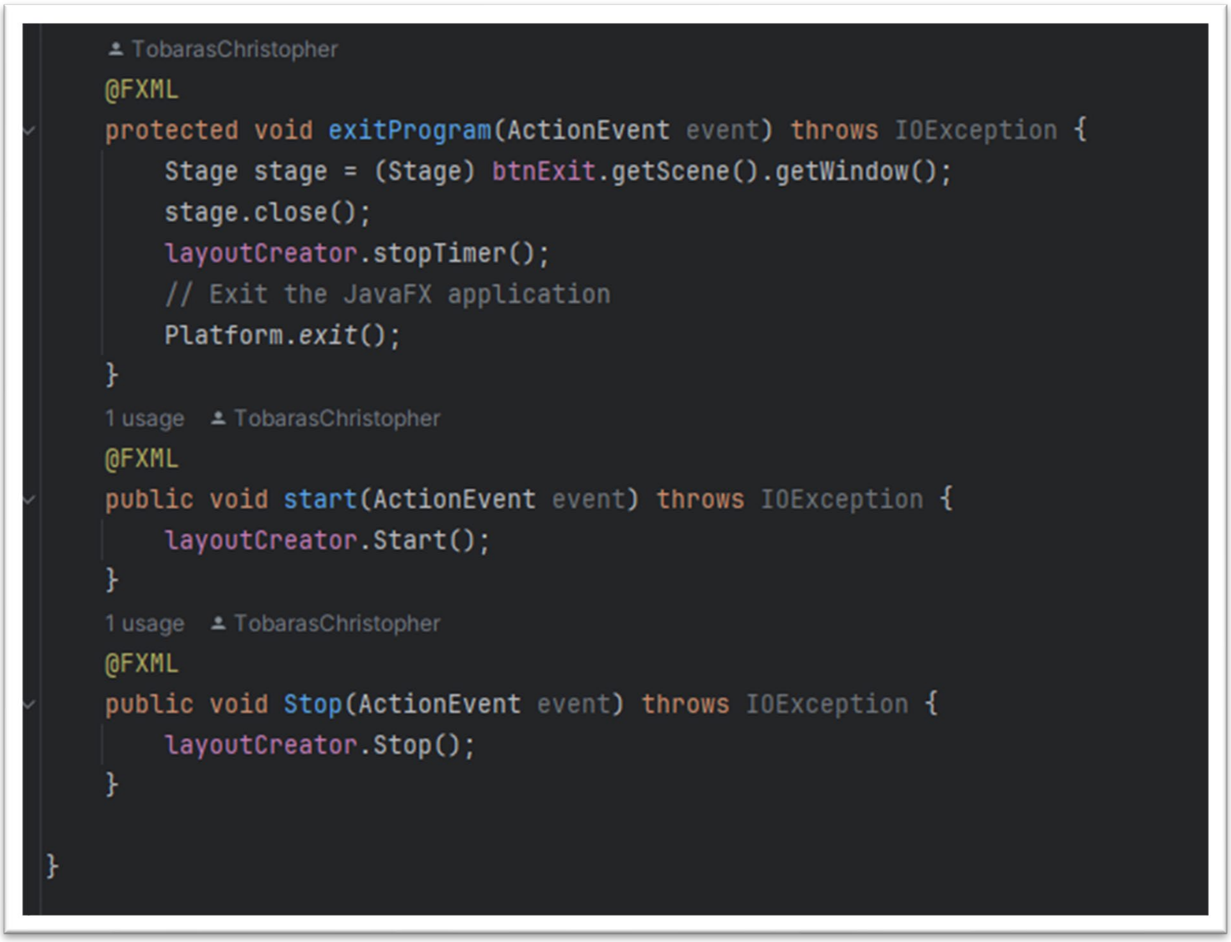
In the code above, the animation will play indefinitely until the application window is closed. This feature helps the user by providing the time during the observation of moving trains.

The GUIController handles the user interaction for the buttons provided in the FXML resource. These buttons are linked to a method in the controller which run the provided code is the button is pressed. To avoid the unnecessary duplication of code, a novel method of opening different windows was introduced.


```
TobarasChristopher
@FXML
protected void goToSchedule(ActionEvent event) throws IOException {
    openWindow( fxmlPath: "/org/example/raileader_rewrite/schedule-view.fxml");
}
TobarasChristopher
@FXML
protected void goToDebug(ActionEvent event) throws IOException {
    openWindow( fxmlPath: "/org/example/raileader_rewrite/debug-view.fxml");
}
new *
@FXML
protected void goToLogs(ActionEvent event) throws IOException {
    openWindow( fxmlPath: "/org/example/raileader_rewrite/log-view.fxml");
}
3 usages   new *
private void openWindow(String fxmlPath) {
    try {
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource(fxmlPath));
        Parent root = fxmlLoader.load();
        Stage newStage = new Stage();
        newStage.setScene(new Scene(root));
        newStage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

R-AI-L Implementation 7: Dynamic Open Window Function

The provided code screenshot shows us multiple methods. Each method that enters a window is marked with the appropriate method name which when run, also runs the openWindow() method with a URL of the associated FXML resource attached. When the openWindow() method is run, it utilizes the same method used to open a window from the MainApplication class and opens the window on top of the main view window.



```
    TobarasChristopher
    @FXML
    protected void exitProgram(ActionEvent event) throws IOException {
        Stage stage = (Stage) btnExit.getScene().getWindow();
        stage.close();
        layoutCreator.stopTimer();
        // Exit the JavaFX application
        Platform.exit();
    }
    1 usage   TobarasChristopher
    @FXML
    public void start(ActionEvent event) throws IOException {
        layoutCreator.Start();
    }
    1 usage   TobarasChristopher
    @FXML
    public void Stop(ActionEvent event) throws IOException {
        layoutCreator.Stop();
    }

}
```

R-AI-L Implementation 8: Start, Stop and Exit Function

There are three final interactions programmed in the main view which do not open a window. These interactions are the start and stop simulation methods, and the exit program method. As the train controller feature is located inside the LayoutCreator Class, the start and stop methods both use the provided object to interact with the train controller, and the exit program method simply uses an exit function to leave the program window, while also closing any running loops inside the application.

2.3.2. Layout Creation

The Layout Creation feature is one of the first methods that is initialised as it is the most important function of the application. It needs to be able to display the track layout, alongside being able to be easily accessible and modifiable if the Train Controller will need to perform its functions. When the GUIController class creates the LayoutCreator object, the layout is first created, and when the initLayout() method is called, the graph is returned using a get method.

The code snippet above is a cut down version of what is implemented in the program to better describe the layout creation process. As the layout is a graph, the graph must contain vertices and edges to denote locations and tracks respectively. A transaction is started on the graph and the vertices v0 and v1 are created as an object from a method similar to the one used for adding vertices to the graph. Following that, an edge is made between the two vertices.

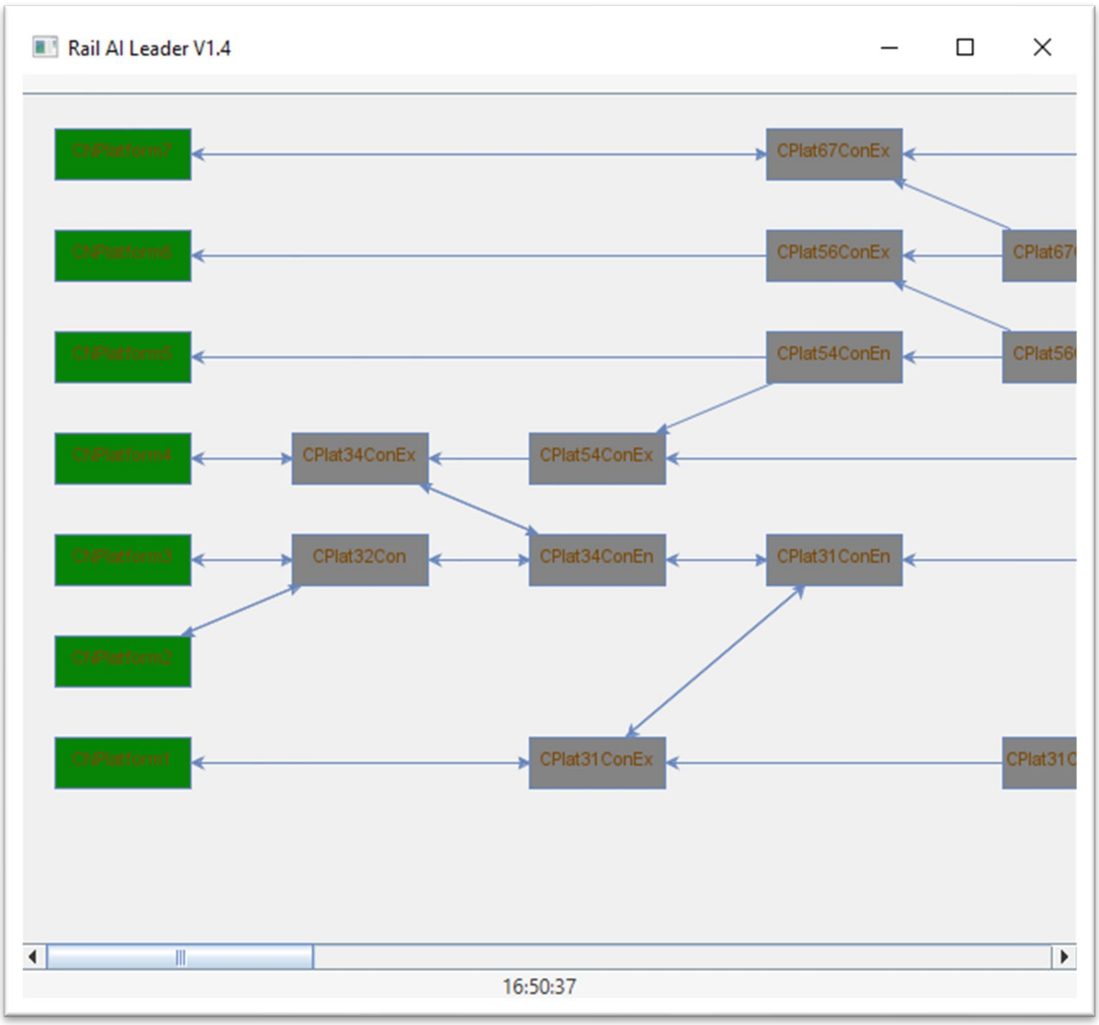
A createVertex() method is used for two functions. The first function is to add the vertex to a graph, using the native library method provided by the JGraphX library. The second function is to add a vertex to a hash map, which stores the vertex object with the key being the name of the vertex. This is done so that any methods or classes that don't have immediate access to the object vertices created by the LayoutCreator class, can access the required vertex by first passing the HashMap, and then finding the object by using the id as the key.

```java
//LayoutCreator.java
public HashMap<String, Object> getVertexList(){
    return vertexMap;
}

//LayoutCreatorTest.java

private LayoutCreator layoutCreator = new LayoutCreator();

public HashMap<String, Object> vertexMap = layoutCreator.getVertexList();

train = new
Train("TestTrain",vertexMap.get("C35ConEn"),vertexMap.get("CNPlatform1"),"
TEST","TEST");
```

R-AI-L Snippet 1: LayoutCreator graph creation

The code snippet above demonstrates the usefulness of the HashMap, as the tests created for the program can readily access the required vertices at any level, provided they create an object of the LayoutCreator class.

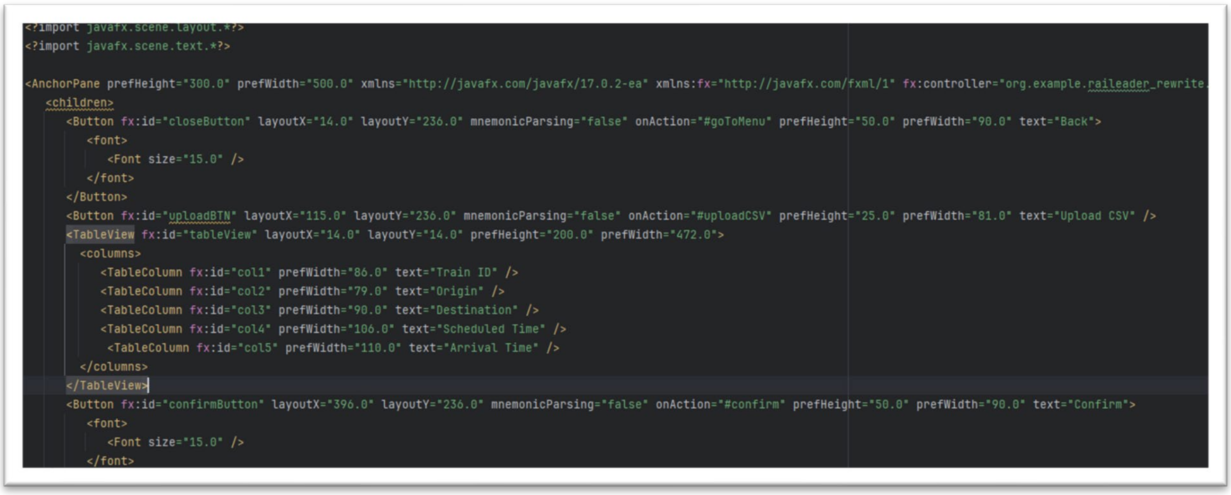
After the transaction is closed, the program returns the graph using the initLayout() get method, and then the program is able to display the graph.



R-AI-L Implementation 9: Generated Layout View

2.3.3. File Upload

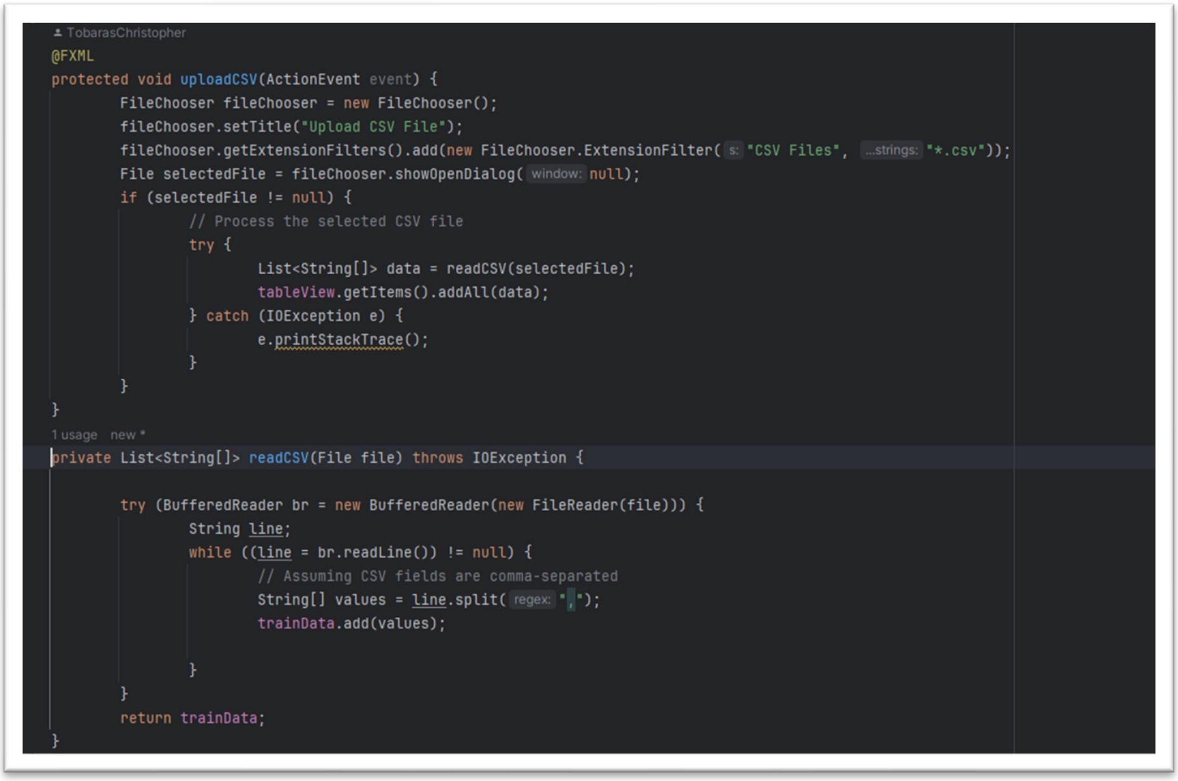
The file upload feature is an important feature as the application does not locally create any trains; instead, it needs to be fed information from a source, which will then create the trains on the layout. This feature is spread out into many components that belong to the ScheduleView class and the ScheduleManager class. The ScheduleView is part of the GUI element that displays a window for the user to input their CSV file and will also show them a preview of the data on a five-column table, with an entry in each column. Once the user is satisfied, they can confirm the choice, which sends the data to the ScheduleManager singleton instance, which is then sent to the LayoutCreator class to allow the Train Controller to process and use the information.



```xml
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<AnchorPane prefHeight="300.0" prefWidth="500.0" xmlns="http://javafx.com/javafx/17.0.2-ea" xmlns:fx="http://javafx.com/fxml/1" fx:controller="org.example.raileader_rewrite.
   <children>
      <Button fx:id="closeButton" layoutX="14.0" layoutY="236.0" mnemonicParsing="false" onAction="#goToMenu" prefHeight="50.0" prefWidth="90.0" text="Back">
         <font>
            <Font size="15.0" />
         </font>
      </Button>
      <Button fx:id="uploadBTN" layoutX="115.0" layoutY="236.0" mnemonicParsing="false" onAction="#uploadCSV" prefHeight="25.0" prefWidth="81.0" text="Upload CSV" />
      <TableView fx:id="tableView" layoutX="14.0" layoutY="14.0" prefHeight="200.0" prefWidth="472.0">
        <columns>
          <TableColumn fx:id="col1" prefWidth="86.0" text="Train ID" />
          <TableColumn fx:id="col2" prefWidth="79.0" text="Origin" />
          <TableColumn fx:id="col3" prefWidth="90.0" text="Destination" />
          <TableColumn fx:id="col4" prefWidth="106.0" text="Scheduled Time" />
           <TableColumn fx:id="col5" prefWidth="110.0" text="Arrival Time" />
        </columns>
      </TableView>
      <Button fx:id="confirmButton" layoutX="396.0" layoutY="236.0" mnemonicParsing="false" onAction="#confirm" prefHeight="50.0" prefWidth="90.0" text="Confirm">
         <font>
            <Font size="15.0" />
         </font>
```

R-AI-L Implementation 10: FXML Resource of File Upload.

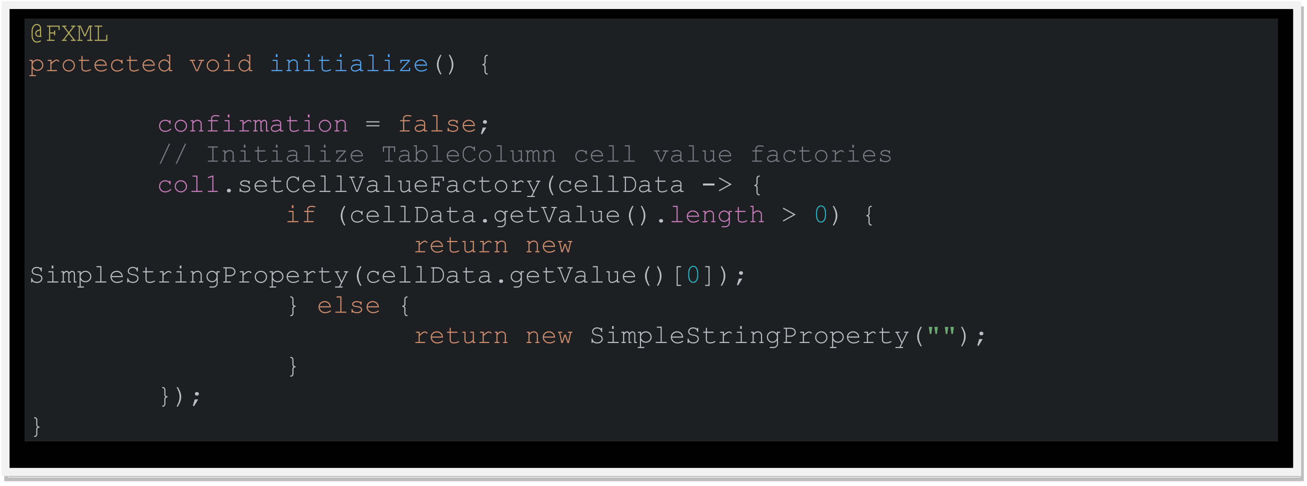
The ScheduleView FXML resource is shown above. A table is added, with five columns, and three buttons. Each button corresponds to a function in the ScheduleView class, and the columns are used as visualisation.



```java
TobarasChristopher
@FXML
protected void uploadCSV(ActionEvent event) {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Upload CSV File");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter( s: "CSV Files",  ...strings: "*.csv"));
        File selectedFile = fileChooser.showOpenDialog( window: null);
        if (selectedFile != null) {
                // Process the selected CSV file
                try {
                        List<String[]> data = readCSV(selectedFile);
                        tableView.getItems().addAll(data);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}
1 usage  new *
private List<String[]> readCSV(File file) throws IOException {

        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
                String line;
                while ((line = br.readLine()) != null) {
                        // Assuming CSV fields are comma-separated
                        String[] values = line.split( regex: ",");
                        trainData.add(values);

                }
        }
        return trainData;
}
```

R-AI-L Implementation 11: Upload and Read CSV functions.

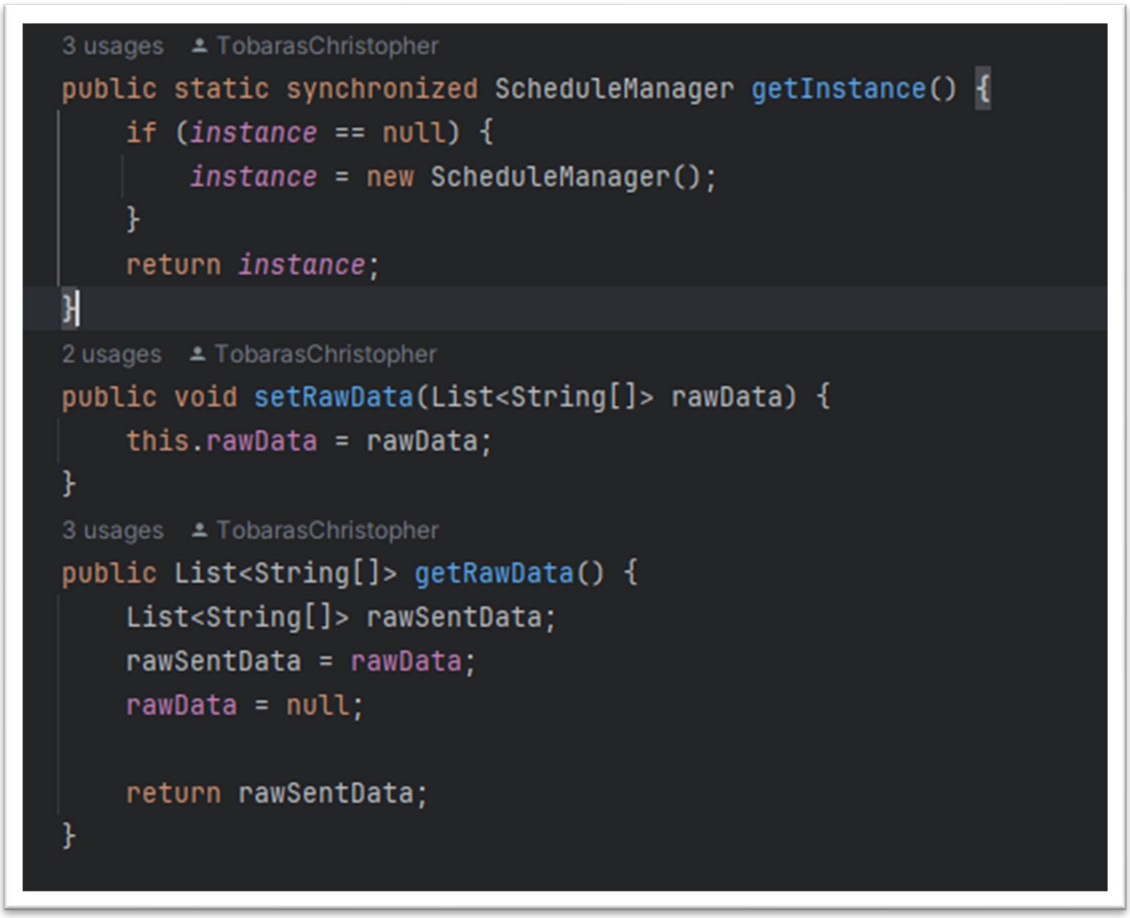
The methods outlined above are the upload and read methods for the ScheduleView class. The uploadCSV() method is called by the button on the FXML, which opens up a window to let the user choose the appropriate file. While the window is open, the only files visible are files with a .CSV extension, and folders with .CSV files inside. Once selected, the raw data is passed to the readCSV() method where it is then put back into a list, and returned to the original UploadCSV() method, finally adding it to the table.



```java
@FXML
protected void initialize() {

        confirmation = false;
        // Initialize TableColumn cell value factories
        col1.setCellValueFactory(cellData -> {
                if (cellData.getValue().length > 0) {
                        return new
SimpleStringProperty(cellData.getValue()[0]);
                } else {
                        return new SimpleStringProperty("");
                }
        });
}
```

R-AI-L Snippet 2: CSV Table Initialise function.

The code snippet above is how each column in the table dynamically updates itself when a CSV file's data is uploaded to the UploadCSV() method. Once the user has observed the data and deemed it is correct, they will press the confirm button which will start the transfer process.



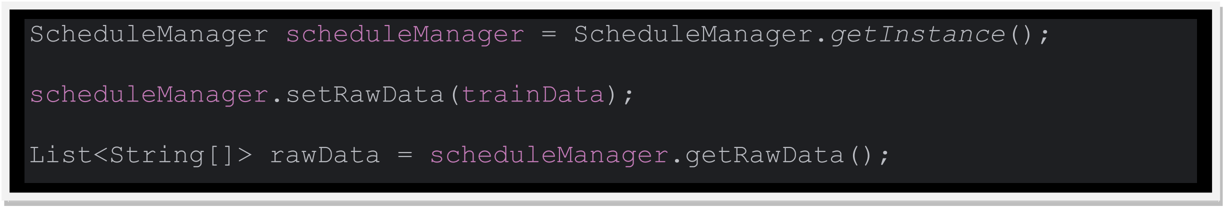
```java
3 usages   TobarasChristopher
public static synchronized ScheduleManager getInstance() {
    if (instance == null) {
        instance = new ScheduleManager();
    }
    return instance;
}
2 usages   TobarasChristopher
public void setRawData(List<String[]> rawData) {
    this.rawData = rawData;
}
3 usages   TobarasChristopher
public List<String[]> getRawData() {
    List<String[]> rawSentData;
    rawSentData = rawData;
    rawData = null;

    return rawSentData;
}
```

R-AI-L Implementation 12: ScheduleManager get and set methods.

The code displayed above is the ScheduleManager singleton instance which is used to transfer the data from the ScheduleView class to the LayoutCreator class. The ScheduleManager makes an instance of itself and then contains a simple set method for incoming data, and a modified get method that sends the data once and then deletes it completely.

The reason this middleman-style method is used, is that when the program opens, an object of LayoutCreator is created by default. When the Schedule button is pressed, the object of the ScheduleView's controller is also made, which makes both unable to inherit the others session.

The ScheduleManager however, makes a session of their own, without the interaction of another class. This means that classes can only grab the current running instance, rather than make a new one. This means that LayoutCreator and ScheduleView can both transmit data over the same instance of ScheduleManager, which allows for the transfer of data to happen between different windows.



```
ScheduleManager scheduleManager = ScheduleManager.getInstance();

scheduleManager.setRawData(trainData);

List<String[]> rawData = scheduleManager.getRawData();
```

R-AI-L Snippet 3: Singleton Instance object acquisition and usage.

This code above describes how a program would interact with the class. Both get and set commands could be in different classes or objects without any inheritance, which fits the situation of the application.

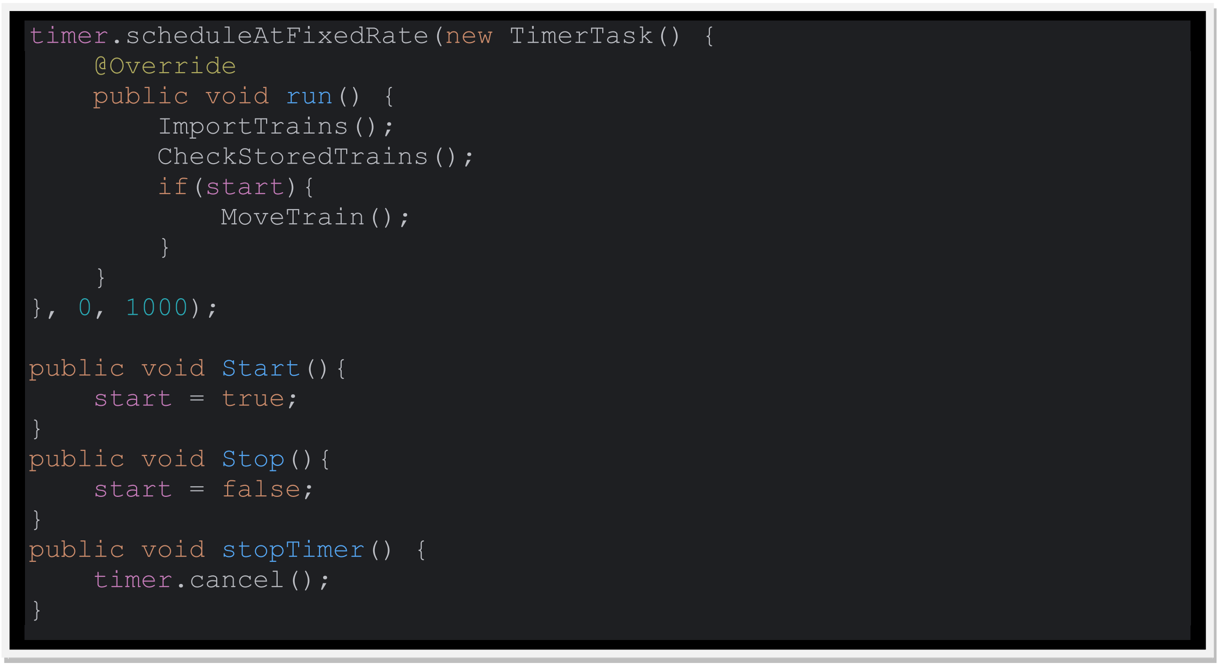
Once the data is set, the window is then closed. Data processing of the trains is done as part of the Train Controller feature.

2.3.4. Train Controller

The Train Controller is one of the core features of the Rail-AI-Leader application and is the biggest feature present in terms of code, and functionality. It compromises a variety of subsystems which help provide the following:

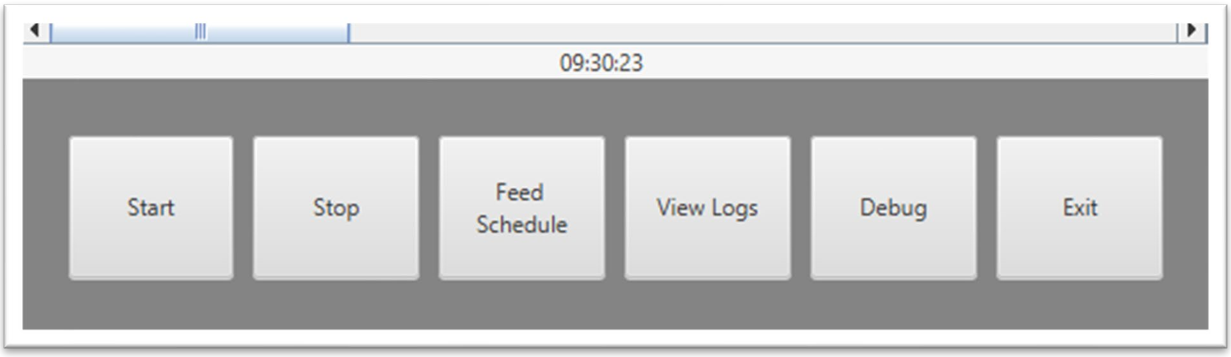
1. Turn the data received from the user into Trains.
2. Handles the time in which trains enter and leave the layout.
3. Assigns a slot for a train and finds a path for them.
4. Checks if the chosen path is occupied by another train.
5. Handling the movement of the train.
6. Submit performance data to the Performance functionality.

These features, especially feature one, two, three and five, all rely on the time for dynamically updating the layout. Therefore, a time system was implemented which focuses on running these features until the main program window is closed.



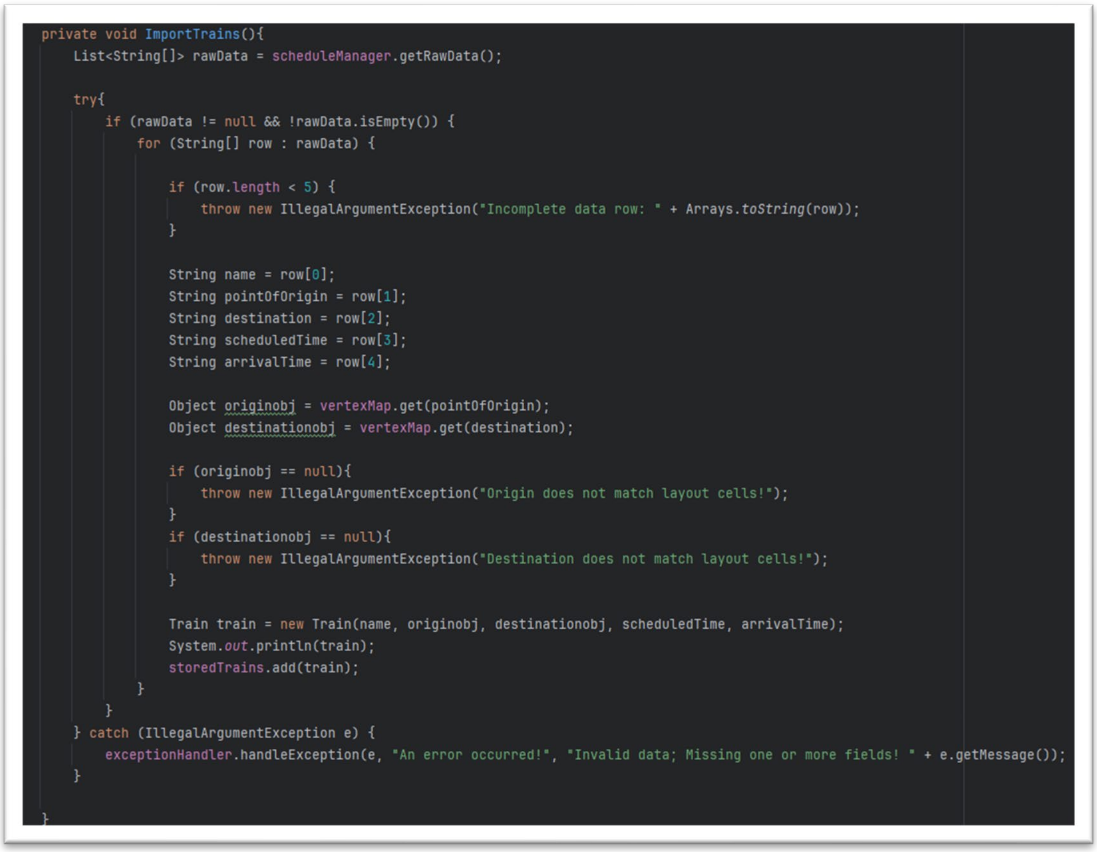
```java
timer.scheduleAtFixedRate(new TimerTask() {
    @Override
    public void run() {
        ImportTrains();
        CheckStoredTrains();
        if(start){
            MoveTrain();
        }
    }
}, 0, 1000);

public void Start(){
    start = true;
}
public void Stop(){
    start = false;
}
public void stopTimer() {
    timer.cancel();
}
```

R-AI-L Snippet 4: LayoutCreator Timer Function



R-AI-L Implementation 13: Menu Button View

The following code snippet above describes how this dynamic system functions. A timer is created upon the LayoutCreator being created as an object, set to restart every 1000 milliseconds. As of every second, the method used to import and process trains, handle the entry of trains, and if the simulation has started, start processing the movement of the trains will be run. Note that only when start is deemed true will the movement process begin, which is controlled by the Start and Stop buttons on the main menu. The exit button also calls the stopTimer() method which cancels the running timer and properly exits the program.

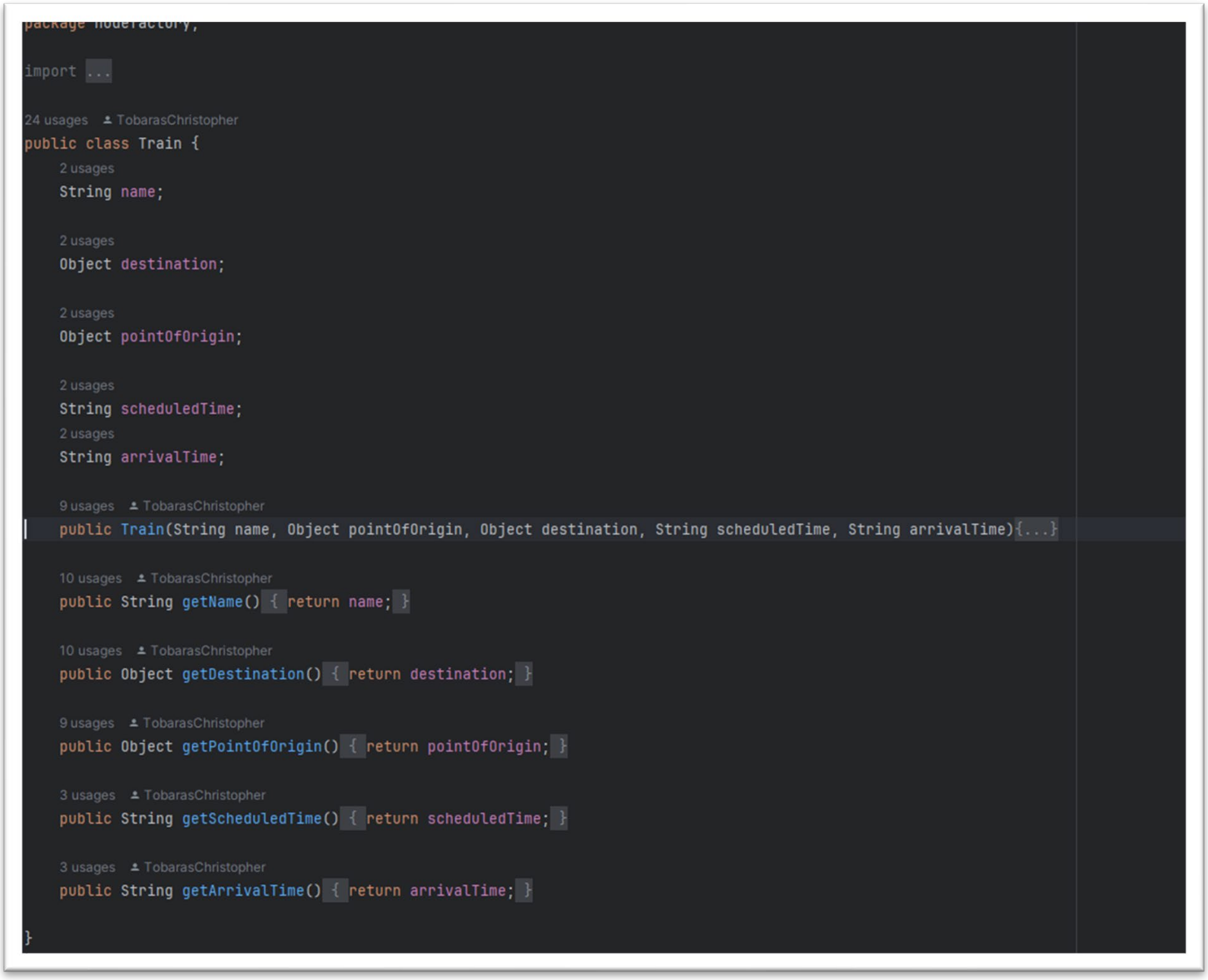


```
private void ImportTrains(){
    List<String[]> rawData = scheduleManager.getRawData();

    try{
        if (rawData != null && !rawData.isEmpty()) {
            for (String[] row : rawData) {

                if (row.length < 5) {
                    throw new IllegalArgumentException("Incomplete data row: " + Arrays.toString(row));
                }

                String name = row[0];
                String pointOfOrigin = row[1];
                String destination = row[2];
                String scheduledTime = row[3];
                String arrivalTime = row[4];

                Object originobj = vertexMap.get(pointOfOrigin);
                Object destinationobj = vertexMap.get(destination);

                if (originobj == null){
                    throw new IllegalArgumentException("Origin does not match layout cells!");
                }
                if (destinationobj == null){
                    throw new IllegalArgumentException("Destination does not match layout cells!");
                }

                Train train = new Train(name, originobj, destinationobj, scheduledTime, arrivalTime);
                System.out.println(train);
                storedTrains.add(train);
            }
        }
    } catch (IllegalArgumentException e) {
        exceptionHandler.handleException(e, "An error occurred!", "Invalid data; Missing one or more fields! " + e.getMessage());
    }

}
```

R-AI-L Implementation 14: Train Creation

The following screenshot shows the first method run on the timer, which handles the import and processing of the data received from the File Upload function. It first receives the raw data from the ScheduleManager class, and checks if the list received is empty or not. As the method is run every second, only one iteration of the data is needed, and that is why the ScheduleManager class clears the data once it is sent.

The method then checks if the list is the correct size and gives each variable a value from the list. It then checks to see if the vertices referenced by the data exists, and if not found, will throw an exception which will be displayed by the error handling function. Once the vertices has been found in the layout, an object is made using a factory described below, and put into a storedTrains list.

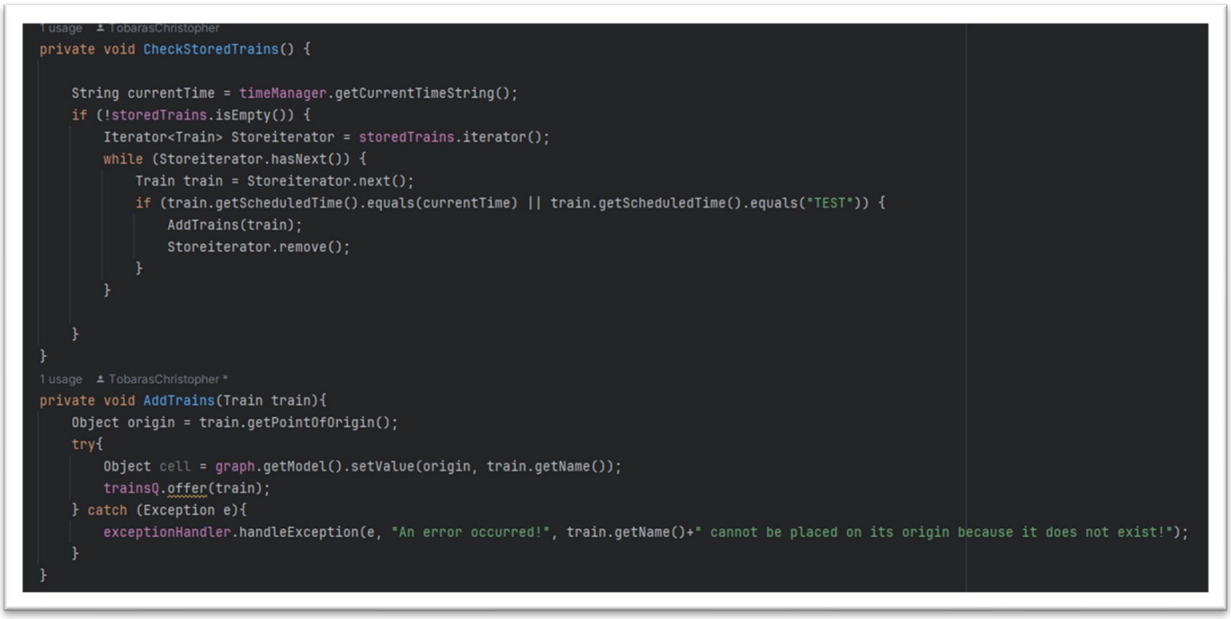


```java
package nodeFactory;

import ...

24 usages  TobarasChristopher
public class Train {
    2 usages
    String name;

    2 usages
    Object destination;

    2 usages
    Object pointOfOrigin;

    2 usages
    String scheduledTime;
    2 usages
    String arrivalTime;

    9 usages  TobarasChristopher
    public Train(String name, Object pointOfOrigin, Object destination, String scheduledTime, String arrivalTime){...}

    10 usages  TobarasChristopher
    public String getName() { return name; }

    10 usages  TobarasChristopher
    public Object getDestination() { return destination; }

    9 usages  TobarasChristopher
    public Object getPointOfOrigin() { return pointOfOrigin; }

    3 usages  TobarasChristopher
    public String getScheduledTime() { return scheduledTime; }

    3 usages  TobarasChristopher
    public String getArrivalTime() { return arrivalTime; }

}
```

R-AI-L Implementation 15: Train Object Factory

The code describes a simple object factory, allowing an object to be created by using the details supplied by the user. This object allows all the data to be focused on a single point which allows the program to easily find and sort the train. Get methods are also supplied to retrieve data quickly.

The next method run on the timer is the CheckStoredTrains() method. The purpose of this functionality is to find the scheduledTime of the train, which is the time it is supposed to enter the layout. Once this time is met with the current time, the train is added using another AddTrains() method and removed from the storedTrains list.



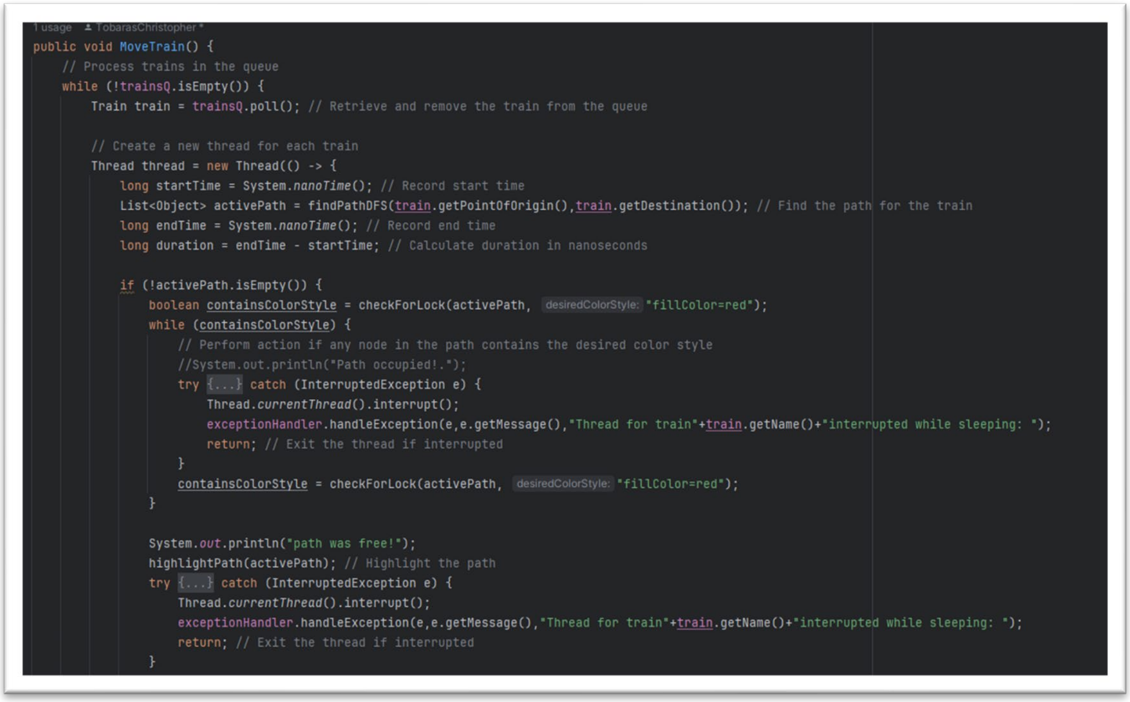
```
1 usage   TobarasChristopher
private void CheckStoredTrains() {

    String currentTime = timeManager.getCurrentTimeString();
    if (!storedTrains.isEmpty()) {
        Iterator<Train> Storeiterator = storedTrains.iterator();
        while (Storeiterator.hasNext()) {
            Train train = Storeiterator.next();
            if (train.getScheduledTime().equals(currentTime) || train.getScheduledTime().equals("TEST")) {
                AddTrains(train);
                Storeiterator.remove();
            }
        }

    }
}
1 usage   TobarasChristopher *
private void AddTrains(Train train){
    Object origin = train.getPointOfOrigin();
    try{
        Object cell = graph.getModel().setValue(origin, train.getName());
        trainsQ.offer(train);
    } catch (Exception e){
        exceptionHandler.handleException(e, "An error occurred!", train.getName()+" cannot be placed on its origin because it does not exist!");
    }
}
```

R-AI-L Implementation 16: Train time addition

The screenshot above describes both the CheckStoredTrains() and AddTrains() method. Once the train has met the time, the AddTrains method opens a transaction with the graph, and uses the setValue() function to grab the vertex referenced by the train and set it with the name of the train. If the vertex is still not existent, an exception will be thrown in this case.

The train is then offered into a queue, which follows the First-In-First-Out asset management rule. This means the last train to be added will be the last sorted, as the queue will always pick out the first one in.

The MoveTrain() method is the most substantial method of the rest, which will take multiple screenshots to elaborate on. It relies on a multithread system where each train present in the queue is given a thread. On that thread, instructions for finding a path, finding out if it is occupied, alongside moving the train are specified, then run. Alongside these processes, performance metrics are recorded, such as comparing the expected arrival time to the actual arrival time, duration of finding a path, and exception handling functionality is included.

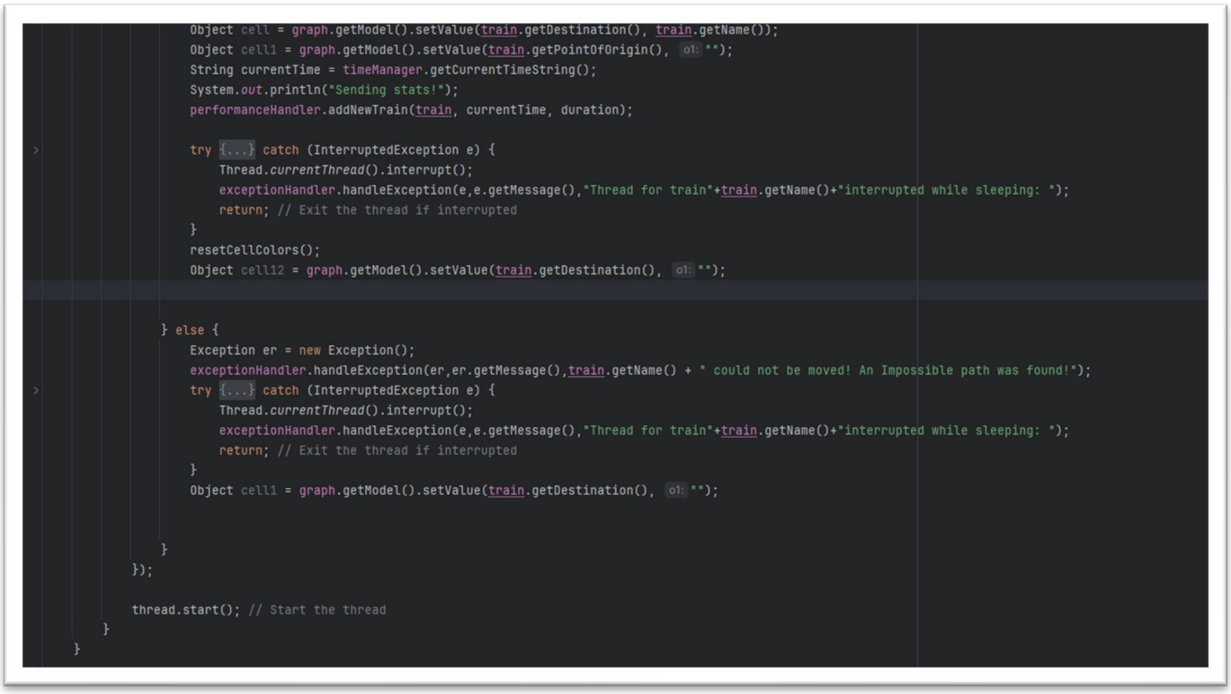


```java
1 usage   TobarasChristopher *
public void MoveTrain() {
    // Process trains in the queue
    while (!trainsQ.isEmpty()) {
        Train train = trainsQ.poll(); // Retrieve and remove the train from the queue

        // Create a new thread for each train
        Thread thread = new Thread(() -> {
            long startTime = System.nanoTime(); // Record start time
            List<Object> activePath = findPathDFS(train.getPointOfOrigin(),train.getDestination()); // Find the path for the train
            long endTime = System.nanoTime(); // Record end time
            long duration = endTime - startTime; // Calculate duration in nanoseconds

            if (!activePath.isEmpty()) {
                boolean containsColorStyle = checkForLock(activePath, desiredColorStyle: "fillColor=red");
                while (containsColorStyle) {
                    // Perform action if any node in the path contains the desired color style
                    //System.out.println("Path occupied!.");
                    try {...} catch (InterruptedException e) {
                        Thread.currentThread().interrupt();
                        exceptionHandler.handleException(e,e.getMessage(),"Thread for train"+train.getName()+"interrupted while sleeping: ");
                        return; // Exit the thread if interrupted
                    }
                    containsColorStyle = checkForLock(activePath, desiredColorStyle: "fillColor=red");
                }

                System.out.println("path was free!");
                highlightPath(activePath); // Highlight the path
                try {...} catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    exceptionHandler.handleException(e,e.getMessage(),"Thread for train"+train.getName()+"interrupted while sleeping: ");
                    return; // Exit the thread if interrupted
                }
```

R-AI-L Implementation 17: First Half of MoveTrain() method

The following screenshot shows the first half of the method, alongside the first instructions for the thread. If the queue is found to be not empty, the first train is then removed from the queue. A new thread is then created, which starts the instructions. First, the FindPathDFS() method is called, alongside recording the time before and after the method has been called, to compare the difference and record it as the duration of the pathfinding system.

If a path has been found, it will be passed into the checkForLock() method, to analyse the path and find out if it has been locked. This lock is found if any of the vertices are detected to be coloured red. If the path is found to be occupied, the method will try again after a short period of time. If the path is clear, the path is then locked by the thread by using the HighlightPath() method. This first saves the vertices current styles in a list, then colours the vertices red to indicate the path is locked logically and visually.

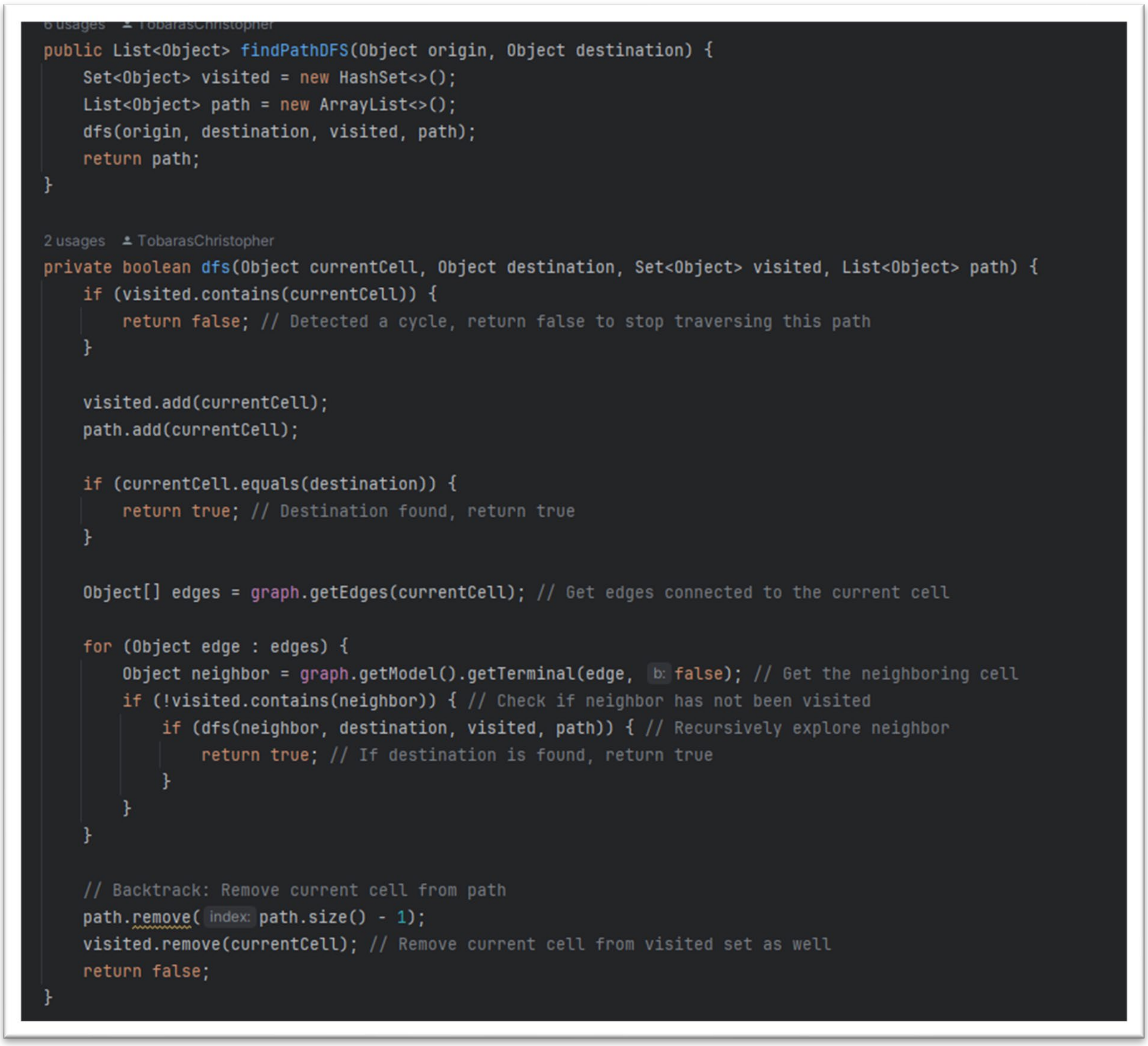


```java
                Object cell = graph.getModel().setValue(train.getDestination(), train.getName());
                Object cell1 = graph.getModel().setValue(train.getPointOfOrigin(),  o1: "");
                String currentTime = timeManager.getCurrentTimeString();
                System.out.println("Sending stats!");
                performanceHandler.addNewTrain(train, currentTime, duration);

                try {...} catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    exceptionHandler.handleException(e,e.getMessage(),"Thread for train"+train.getName()+"interrupted while sleeping: ");
                    return; // Exit the thread if interrupted
                }
                resetCellColors();
                Object cell12 = graph.getModel().setValue(train.getDestination(),  o1: "");


            } else {
                Exception er = new Exception();
                exceptionHandler.handleException(er,er.getMessage(),train.getName() + " could not be moved! An Impossible path was found!");
                try {...} catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    exceptionHandler.handleException(e,e.getMessage(),"Thread for train"+train.getName()+"interrupted while sleeping: ");
                    return; // Exit the thread if interrupted
                }
                Object cell1 = graph.getModel().setValue(train.getDestination(),  o1: "");


            }
        });

        thread.start(); // Start the thread
    }
}
```

R-AI-L Implementation 18: Second half of MoveTrain() method

The above screenshot shows the end of the instructions for the thread, with the last line of code starting the thread. After locking the path, a transaction is opened on the graph, clearing the name of the train from the point of origin, and moving the name to the destination. This means the train has now moved successfully. The time is then recorded, and all statistics passed to the PerformanceHandler.

After a set amount of time, the resetCellColors() method is called, which takes the saved cell styles from the HighlightPath() method and reverts the cell colours from red to their original colours. The destination vertex is then wiped of the trains name, indicating that the vertex is no longer occupied. The rest of the instructions specify an error handling case for when the train could not find a path and is therefore removed from the graph.



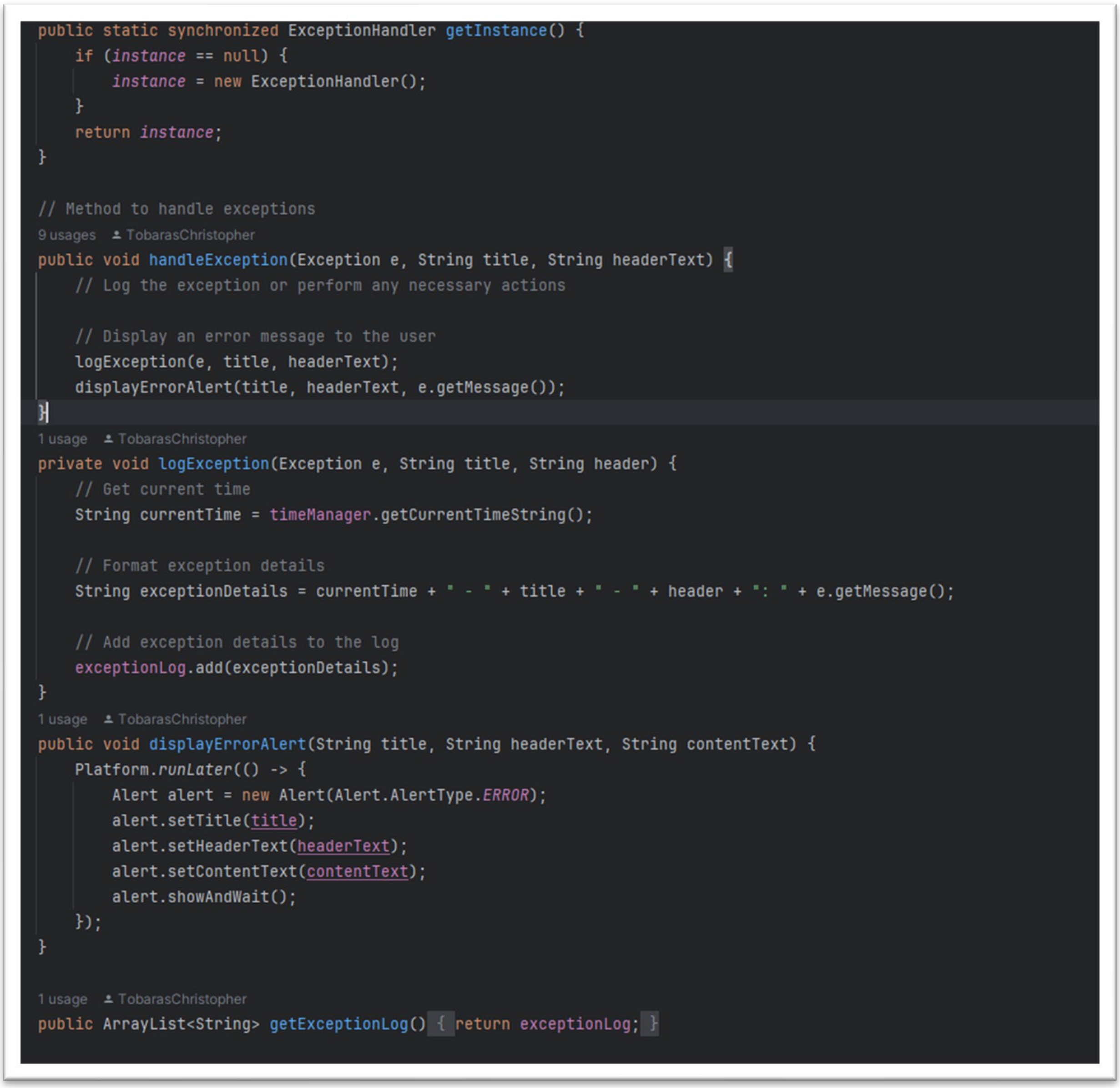
```java
6 usages  TobarasChristopher
public List<Object> findPathDFS(Object origin, Object destination) {
    Set<Object> visited = new HashSet<>();
    List<Object> path = new ArrayList<>();
    dfs(origin, destination, visited, path);
    return path;
}

2 usages  TobarasChristopher
private boolean dfs(Object currentCell, Object destination, Set<Object> visited, List<Object> path) {
    if (visited.contains(currentCell)) {
        return false; // Detected a cycle, return false to stop traversing this path
    }

    visited.add(currentCell);
    path.add(currentCell);

    if (currentCell.equals(destination)) {
        return true; // Destination found, return true
    }

    Object[] edges = graph.getEdges(currentCell); // Get edges connected to the current cell

    for (Object edge : edges) {
        Object neighbor = graph.getModel().getTerminal(edge,  b: false); // Get the neighboring cell
        if (!visited.contains(neighbor)) { // Check if neighbor has not been visited
            if (dfs(neighbor, destination, visited, path)) { // Recursively explore neighbor
                return true; // If destination is found, return true
            }
        }
    }

    // Backtrack: Remove current cell from path
    path.remove( index: path.size() - 1);
    visited.remove(currentCell); // Remove current cell from visited set as well
    return false;
}
```

R-AI-L Implementation 19: Pathfinding Algorithm

The following screenshot describes the pathfinding algorithm used to figure out a path between the point of origin vertex and the destination. The algorithm is a Depth-First-Search algorithm, with a backtracking mechanism, to remove vertices that don't belong to the correct path. It also features neighbour recognition, which means that it can detect previously visited nodes, and backtrack to a previous point. These functions are necessary as the method is designed to return the path as a list of vertexes, which will then be coloured and used for locking.

2.3.5. Error Handling

When an error is encountered in the software, the program uses a unique function to make sure the exception does not disrupt the features of the program, and also provide the user a view of the error, alongside any means to export it. This is a much smaller feature but allows the usage of previous methods shown in this implementation to give a better user experience. The ExceptionHandler class is a singleton instance that can be used to both notify and help transfer the data to different classes.



```java
public static synchronized ExceptionHandler getInstance() {
    if (instance == null) {
        instance = new ExceptionHandler();
    }
    return instance;
}

// Method to handle exceptions
9 usages  TobarasChristopher
public void handleException(Exception e, String title, String headerText) {
    // Log the exception or perform any necessary actions

    // Display an error message to the user
    logException(e, title, headerText);
    displayErrorAlert(title, headerText, e.getMessage());
}
1 usage  TobarasChristopher
private void logException(Exception e, String title, String header) {
    // Get current time
    String currentTime = timeManager.getCurrentTimeString();

    // Format exception details
    String exceptionDetails = currentTime + " - " + title + " - " + header + ": " + e.getMessage();

    // Add exception details to the log
    exceptionLog.add(exceptionDetails);
}
1 usage  TobarasChristopher
public void displayErrorAlert(String title, String headerText, String contentText) {
    Platform.runLater(() -> {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(headerText);
        alert.setContentText(contentText);
        alert.showAndWait();
    });
}

1 usage  TobarasChristopher
public ArrayList<String> getExceptionLog() { return exceptionLog; }
```

R-AI-L Implementation 20: ExceptionHandler Singleton instance

The screenshot of the following code shows the methods of the ExceptionHandler class. If an error is found in code, a catch block in that code will call upon the handleException() method, which calls on both displaying and storing the error message. The logException() method will store the entire error message, alongside the time in a string and put it inside a list, meanwhile the displayErrorAlert() method will create a new JavaFX window and alert the user of the error. The following code will display the functionality being used by a catch block.

```java
catch (NullPointerException ex) {
    exceptionHandler.handleException(ex,ex.getMessage(),"Example Error
Found!");

}
```

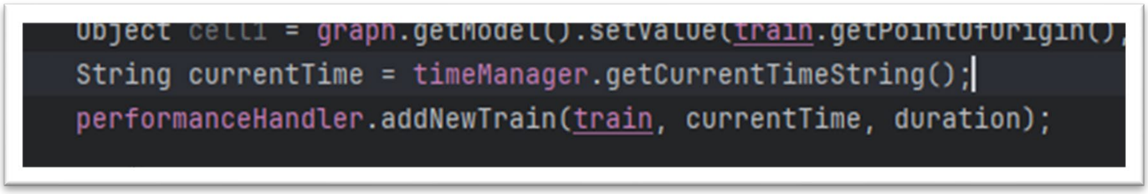
R-AI-L Snippet 5: Example usage of ExceptionHandler class

The user can now observe the error that has been displayed to then, and proceed to the Logs section of the GUI display to see a timeline of any error that has been displayed in the session.

The log display is very similar to the file upload functionality that was implemented earlier. It consists of a simple method of acquiring the data upon initialisation and displays it on a table with a single column. The user can observe the data and if it is deemed necessary, they can export it to a CSV file. It then uses a similar function to the data upload method but instead writes the data in a newly created CSV file. The user is then able to select where it is stored and how it is named.

2.3.6. Performance Metrics

The performance metrics functionality is remarkably similar to the previous feature described, as it is a singleton instance that handles the transfer of data to a different window which can then be exported to data. The saving of data is used by a class to store it in a list by the PerformanceHandler class, which then gets imported by the DebugController class.



```
Object cell1 = graph.getModel().setValue(train.getPointOfOrigin()
String currentTime = timeManager.getCurrentTimeString();
performanceHandler.addNewTrain(train, currentTime, duration);
```

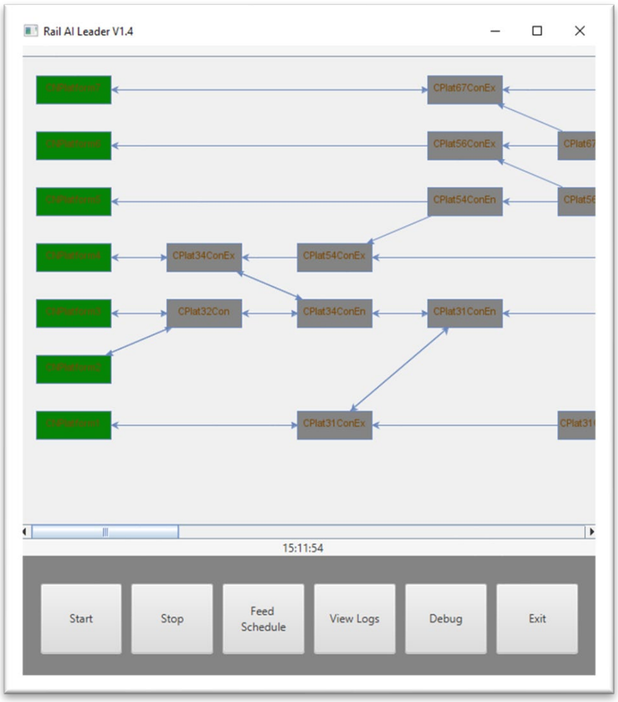
R-AI-L Implementation 21: Usage of PerformanceHandler class

The performance metrics are then displayed on a five column table, which can then be exported to a CSV file in the same fashion as the error handling functionality.

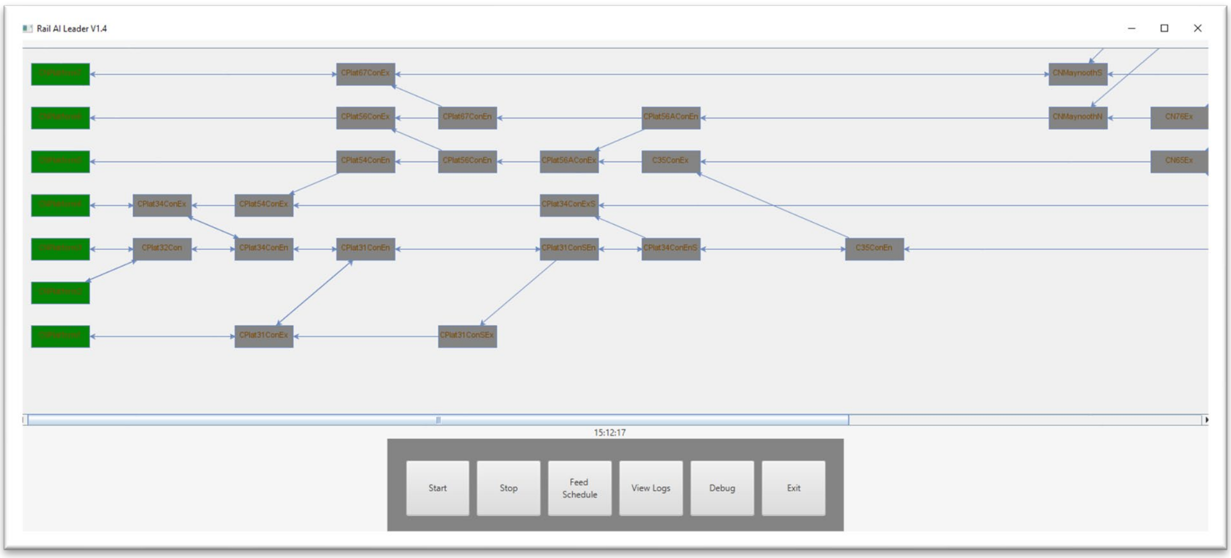
2.4. Graphical User Interface (GUI)

This section will showcase each of the graphical elements that can be seen within the Rail-AI-Leader application.

When the user first opens the program, the application brings up the Main Menu.

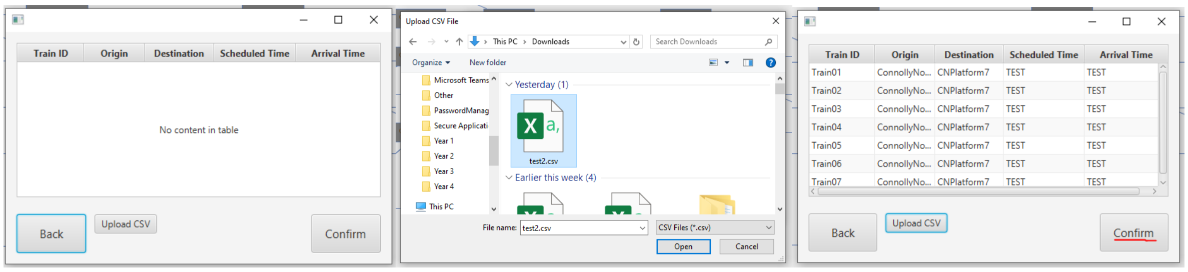


R-AI-L Graphical Element 1: Main Menu



R-AI-L Graphical Element 2: Main Menu Scaling Feature

As shown in the second diagram, the program is able to be scaled to provide a better view of the track layout. To add train details, the user will navigate using the menu shown at the bottom of the window, and press "Feed Schedule".

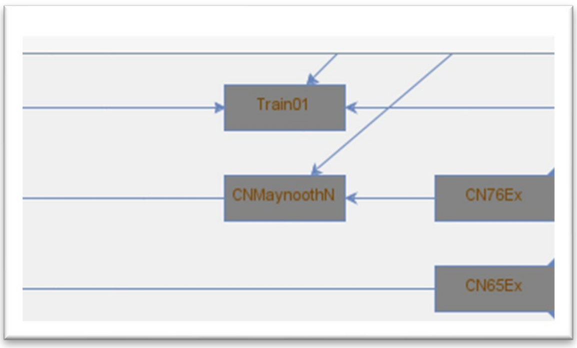


Train ID	Origin	Destination	Scheduled Time	Arrival Time
Train01	ConnollyNo...	CNPlatform7	TEST	TEST
Train02	ConnollyNo...	CNPlatform7	TEST	TEST
Train03	ConnollyNo...	CNPlatform7	TEST	TEST
Train04	ConnollyNo...	CNPlatform7	TEST	TEST
Train05	ConnollyNo...	CNPlatform7	TEST	TEST
Train06	ConnollyNo...	CNPlatform7	TEST	TEST
Train07	ConnollyNo...	CNPlatform7	TEST	TEST

R-AI-L Graphical Element 3, 4 and 5: Process of uploading a CSV File and Tableview before confirmation.

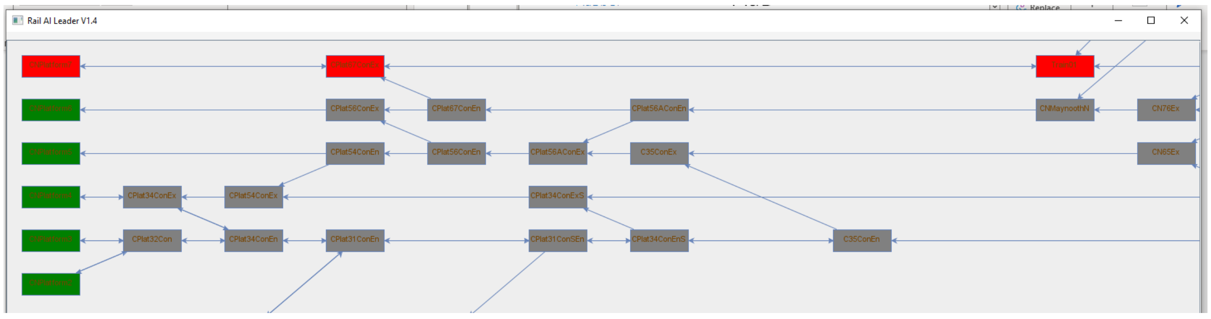
This is the process of uploading a schedule to the Schedule View window. The window starts out with an empty table, but when supplied with a CSV file, the information on the CSV file will be displayed to the user using the table. If the user is satisfied, they can press confirm which closes the window and brings them to the main menu.

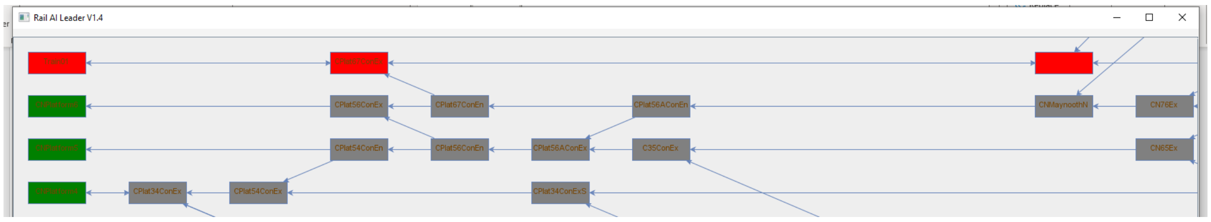
In our CSV file, we have specified the trains point of origin being the CNMaynoothS vertex. As we have put "TEST" in our scheduled time, the train will appear instantly on the track layout where the node CNMaynoothS is.

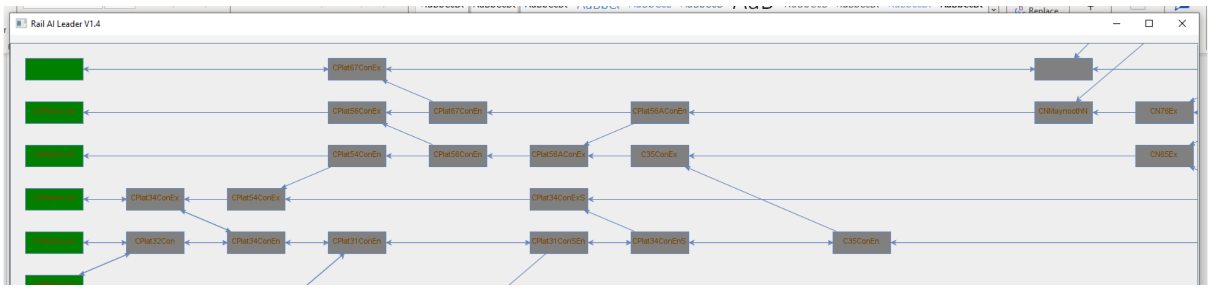


R-AI-L Graphical Element 6: Inserted train on the menu graph

The train will not move unless the simulation is run. This can be done by pressing the start button on the navigation panel. If the user would like to stop the simulation at any time, pressing the stop button will prevent any other trains without a path from creating one.

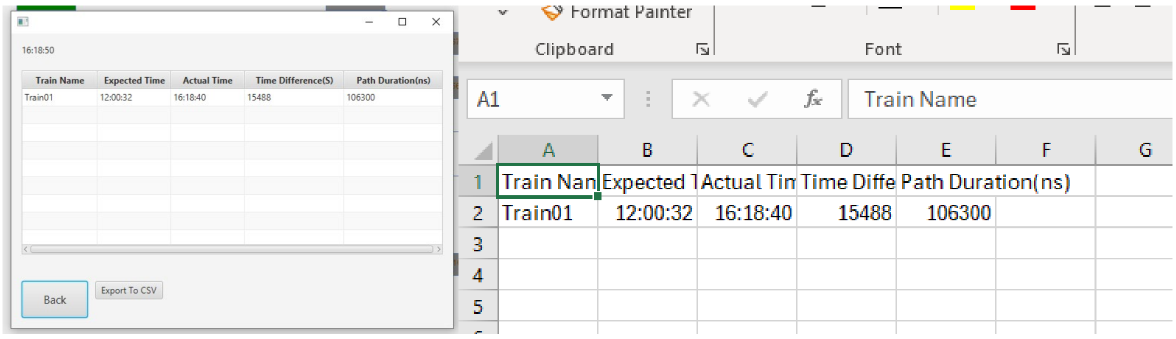






R-AI-L Graphical Element 7, 8 and 9: View of train locking a path, moving to destination, and subsequent unlocking.

This trio of screenshots demonstrates the trains movement throughout the track layout. It first finds a path, and then colours the path vertices red. After a set amount of time, the train moves to the destination vertex, and after some more time, the path vertices are coloured back to normal.


16:18:50

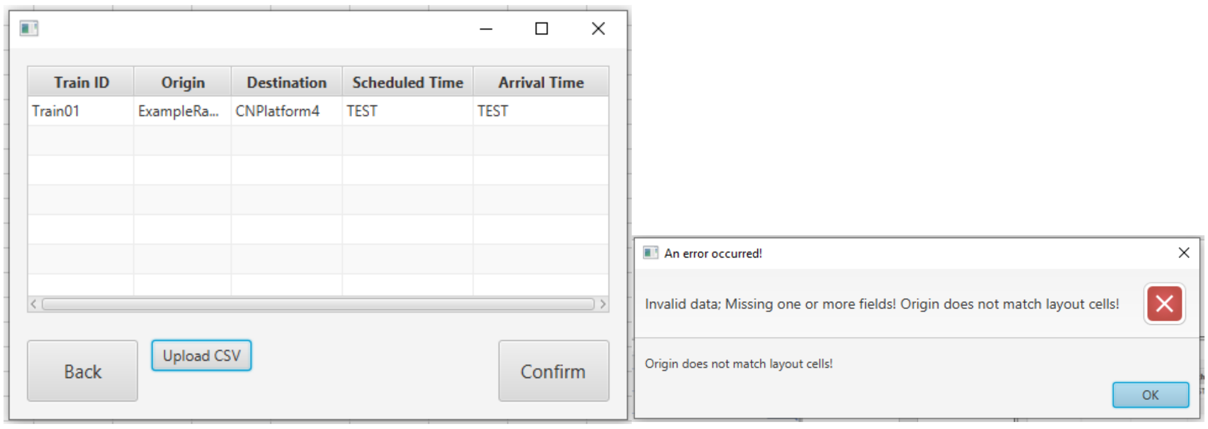
Train Name	Expected Time	Actual Time	Time Difference(S)	Path Duration(ns)
Train01	12:00:32	16:18:40	15488	106300

	A	B	C	D	E	F	G
1	Train Nan	Expected T	Actual Tin	Time Diffe	Path Duration(ns)		
2	Train01	12:00:32	16:18:40	15488	106300		
3							
4							
5							

R-AI-L Graphical Element 10 and 11: Export Performance View and CSV File

Once the train has been sorted, the user can then navigate to the debug view, where they can view their sorted trains and the performance statistics. If the user would like to export the information, they can click the "Export to CSV" button which will give them a new CSV file with the values.

Exception handling will appear in most cases where information is input into the program, especially where the file upload is. Here is a case where an origin is specified that does not exist in the track layout.



Train ID	Origin	Destination	Scheduled Time	Arrival Time
Train01	ExampleRa...	CNPlatform4	TEST	TEST

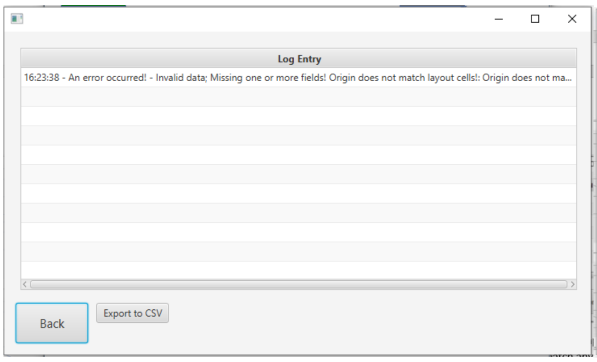
An error occurred!

Invalid data; Missing one or more fields! Origin does not match layout cells!

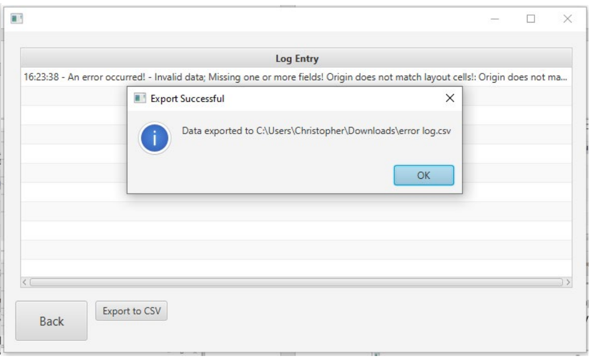
Origin does not match layout cells!

R-AI-L Graphical Element 12 and 13: Log view and Error alert message

As the incorrect information is accidentally input into the program, the alert box shows up and notifies us that the origin does not match any of the track layout vertices. Errors like this are embedded with a timestamp in the logs view of the program.


Log Entry

```
16:23:38 - An error occurred! - Invalid data; Missing one or more fields! Origin does not match layout cells!: Origin does not ma...
```


Log Entry

```
16:23:38 - An error occurred! - Invalid data; Missing one or more fields! Origin does not match layout cells!: Origin does not ma...
```

Export Successful

Data exported to C:\Users\Christopher\Downloads\error log.csv

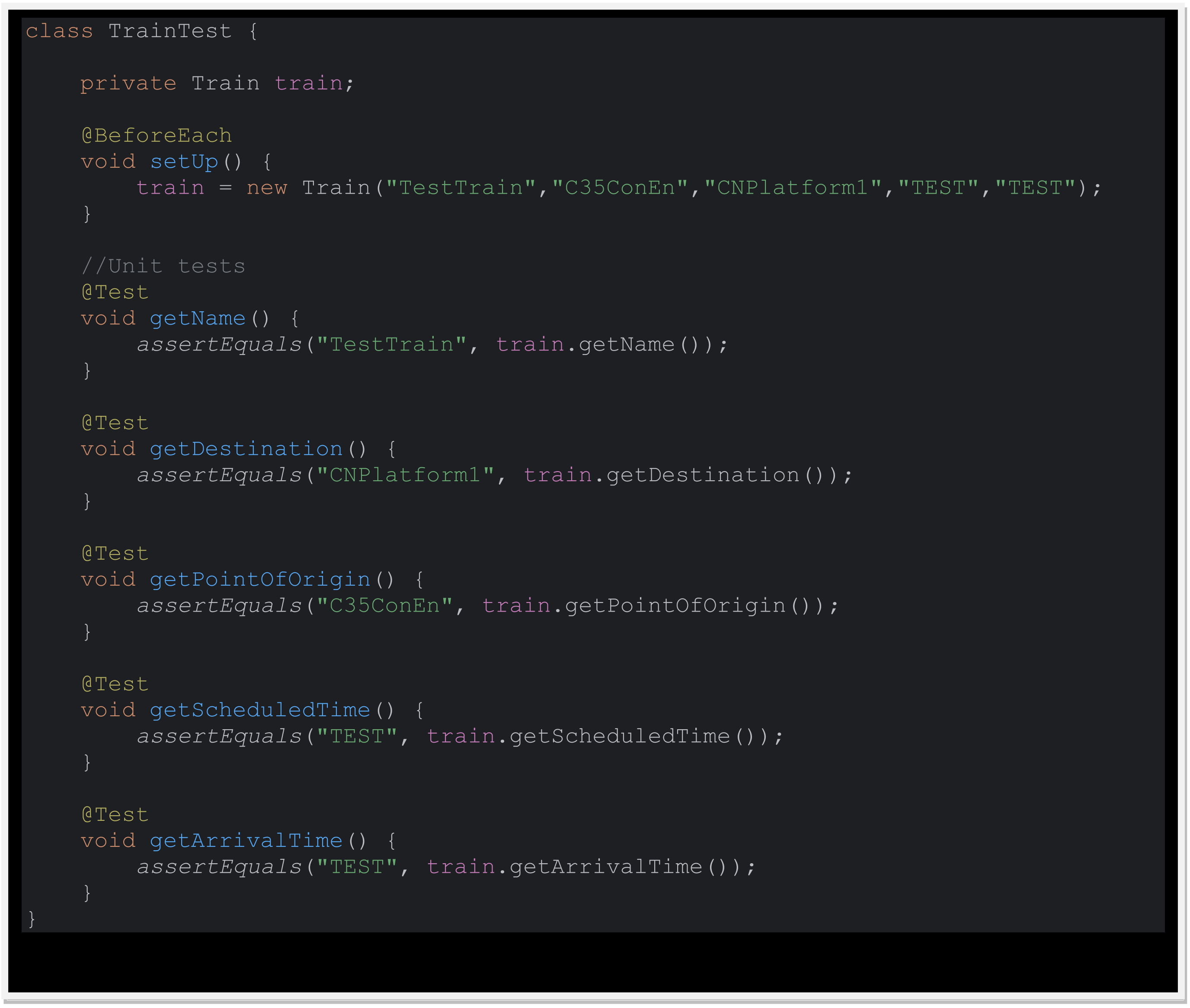
R-AI-L Graphical Element 14 and 15: Process of exporting log file

Much like how the performance metrics are exported, if the person would like to record and store the error, they can export it to a CSV file which can be named by them and stored anywhere on the hardware.

2.5. Testing & Evaluation

A variety of testing was conducted during and after the development of the features in the Rail-AI-Leader application. These included programmed unit and integration tests using the Junit5 library, a testing framework for Java which allows a more flexible and modular approach to testing methods and classes.

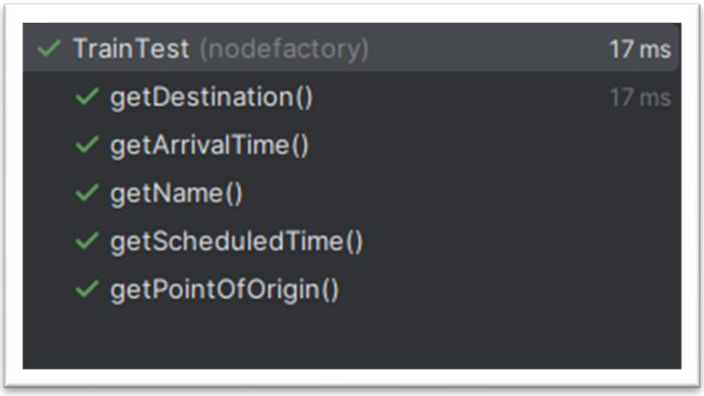
Tests were created and split into three distinct categories. The first class of tests were unit tests designed to make sure the train object factory worked as needed. The second class focused on the LayoutCreator class, which included a number of unit and integrational tests in order to make sure the systems worked together seamlessly. The third and final class of tests were designed to ensure the data transfer between the ScheduleManager singleton class, as the importance of importing trains is a high priority of the application.



```
class TrainTest {

    private Train train;

    @BeforeEach
    void setUp() {
        train = new Train("TestTrain","C35ConEn","CNPlatform1","TEST","TEST");
    }

    //Unit tests
    @Test
    void getName() {
        assertEquals("TestTrain", train.getName());
    }

    @Test
    void getDestination() {
        assertEquals("CNPlatform1", train.getDestination());
    }

    @Test
    void getPointOfOrigin() {
        assertEquals("C35ConEn", train.getPointOfOrigin());
    }

    @Test
    void getScheduledTime() {
        assertEquals("TEST", train.getScheduledTime());
    }

    @Test
    void getArrivalTime() {
        assertEquals("TEST", train.getArrivalTime());
    }
}
```

R-AI-L Snippet 6: Train object unit tests

The following screenshot shows the unit testing of the train object, which shows an example train being made, and the assertion that the correct information is being returned from all get methods. The train object creation will appear in the next two classes of tests as the train is important for the operation of the application.



```
✓ TrainTest (nodefactory)        17 ms
    ✓ getDestination()           17 ms
    ✓ getArrivalTime()
    ✓ getName()
    ✓ getScheduledTime()
    ✓ getPointOfOrigin()
```

R-AI-L Implementation 22: Train Test Visualised

This image above shows us that the system was able to create an object and access any of its values at any point, which is satisfactory.

```
class LayoutCreatorTest {


    private Train train;
    private LayoutCreator layoutCreator = new LayoutCreator();
    public mxGraph graph = new mxGraph();
    public HashMap<String, Object> vertexMap =
layoutCreator.getVertexList();

    //Functional Tests
    @Test
    void findPath() {
        train = new
Train("TestTrain",vertexMap.get("C35ConEn"),vertexMap.get("CNPlatform1")
,"TEST","TEST");
        List<Object> path =
layoutCreator.findPathDFS(train.getPointOfOrigin(),train.getDestination(
));
        assertNotNull(path, "The path returned must exist");
        assertFalse(path.isEmpty(), "The path returned must have
entries");
    }
    @Test
    void findNoPath() {
        train = new
Train("TestTrain",vertexMap.get("C35ConEn"),vertexMap.get("TestNode
"),"TEST","TEST");
        List<Object> path =
layoutCreator.findPathDFS(train.getPointOfOrigin(),train.getDestination(
));
        assertNotNull(path, "The path returned must exist");
        assertTrue(path.isEmpty(), "The path returned must not contain
anything");
    }
```

R-AI-L Snippet 7: LayoutCreator Functional Tests

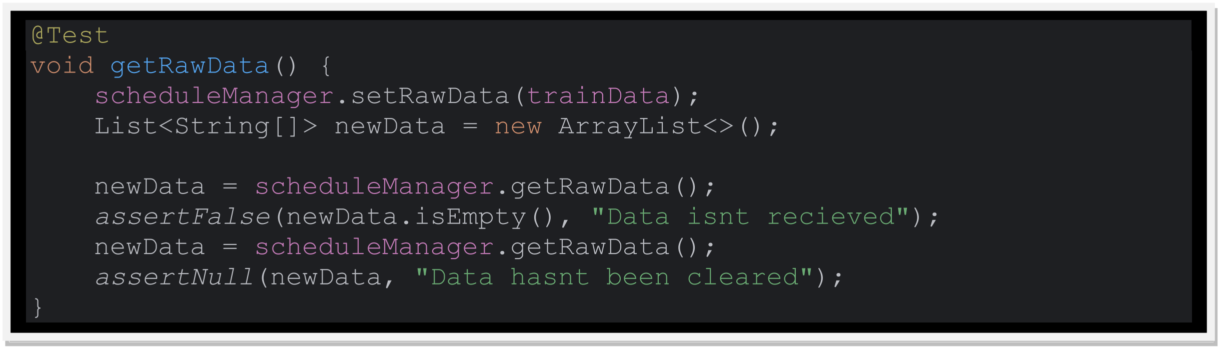
The snippet of code shown is the initialisation of the LayoutCreator test class, and the first batch of tests, which test the findpath method. The first test asserts that a path should be returned, as the path is easily findable, and the second tests asserts the opposite, returning no path as it is impossible to.

```
@Test
    void checkForLock() {
        graph = layoutCreator.initLayout();
        graph.getModel().setStyle(vertexMap.get("CPlat31ConEx"),
"fillColor=red");
        train = new
Train("TestTrain",vertexMap.get("CPlat31ConSEx"),vertexMap.get("CNPlatfo
rm1"),"TEST","TEST");
        List<Object> path =
layoutCreator.findPathDFS(train.getPointOfOrigin(),train.getDestination(
));
        boolean check = layoutCreator.checkForLock(path,
"fillColor=red");
        assertTrue(check, "The path must be detected as occupied");
    }
    @Test
    void checkForNoLock() {
        train = new
Train("TestTrain",vertexMap.get("C35ConEn"),vertexMap.get("CNPlatform1")
,"TEST","TEST");
        List<Object> path =
layoutCreator.findPathDFS(train.getPointOfOrigin(),train.getDestination(
));
        boolean check = layoutCreator.checkForLock(path,
"fillColor=red");
        assertFalse(check, "The path must be detected as free");
    }

    //Unit Tests
    @Test
    void PathHighlightTest() {
        boolean check = true;

        train = new
Train("TestTrain",vertexMap.get("C35ConEn"),vertexMap.get("CNPlatform1")
,"TEST","TEST");
        List<Object> path =
layoutCreator.findPathDFS(train.getPointOfOrigin(),train.getDestination(
));
        layoutCreator.highlightPath(path);
        for (Object cell : path) {
            if(!Objects.equals(graph.getModel().getStyle(cell),
"fillColor=red")){
                //System.out.println("Red Lock Detected!");
                check = false;
            }
        }

        assertTrue(check, "Path must display red when path found");
    }
}
```

R-AI-L Snippet 8: LayoutCreator Path find and Highlight tests.

The second half of the tests check for if the path was highlighted by a train, and asserts that the path is locked and returns a true value from checking the lock. The next text asserts the opposite, once the two trains do not intersect with each others paths. The final test tests to see that the highlight method correctly displays to the user that the path turns red by taking the style and asserting it red.

The final class of tests belong to the ScheduleManager class, and use Junit5's setUp() feature, which initialises the environment before testing. Two trains are added and are used in the following test.



```
@Test
void getRawData() {
    scheduleManager.setRawData(trainData);
    List<String[]> newData = new ArrayList<>();

    newData = scheduleManager.getRawData();
    assertFalse(newData.isEmpty(), "Data isnt recieved");
    newData = scheduleManager.getRawData();
    assertNull(newData, "Data hasnt been cleared");
}
```

R-AI-L Snippet 9: Singleton Instance Tests

As shown in the test, there are two assertions in the same test. The data created in the setUp() method is set in the ScheduleManager instance, received and asserted to not be empty. Afterwards, it repeats this in the same test. The reason for this is that the timer system runs the get method every second, potentially duplicating data if no clearing of the data once sent is made. The second assertion is for the same data to be empty, as it will show the instance has cleared it after sending it once.

Other testing methods used are regression testing, where the tests created during development are run again after each change, specifically when an update to the GitHub repo is made. After each feature was implemented, the GitHub was updated to reflect the new changes, and tests were run to ensure that the core functionality remained the same and did not break. Examples of regression testing was the rewrite of the pathfinding algorithm, which could have broken some of the functionality, specifically the highlight path function as a path is needed to highlight the trains movement.

The Rail-AI-Leader application was also evaluated using black box testing techniques, such as inputting and outputting types regardless of what the program expects it to be. Inputting non-CSV files in the file upload section, exporting data in a different format, or leaving out data upon user input produced interesting results which allowed for better exception handling and programming techniques.

3.0 Conclusions

Rail-AI-Leader has been designed to implement the machine automation of train movements, inside an application that can be utilized to observe, study and train movements, alongside exporting data which could be used to tweak algorithms. The GUI graph interface presents a useful way of observing the movements of trains, generating new trains, and properly handling common exceptions which may arise from managing a track layout, such as pathfinding and locking. The main advantage of this project is that the application can optimise the flow of traffic on railways. By automating the train movements, it removes the logical and physical aspect from the user and allows them to potentially focus on other tasks, improving productivity.

Another advantage of the application is the safety that is implemented in the form of a path locking system. This helps trains ensure that once a path is found, they are only able to cross it if it is not being used by another train, which can prevent the possibility of a collision. As the graph is transaction based, two trains are not able to access the graph at the same time and cause this issue.

A final advantage of the program is that it follows a modular architecture. Utilizing JavaFX and JGraphX, the system is easily modifiable and can be added with new features and algorithms that can allow a choice of different pathfinding software or other track layouts.

However, the application does come with its limitations. The application is inflexible, and any changes to the track layout in real life would require the user to be proficient in JGraphX in order to implement changes. For the file upload system, only CSV files are accepted, which means more popular types are not supported. Another problem encountered is that on a completely undirected graph, trains are seen finding impossible paths which do not make sense.

Despite the limitations addressed, the Rail-AI-Leader application is an important tool for the current, and future development of machine learning algorithms in the railway industry. The ease of access, simplified operation of the program and ability to export output data measured from the application help provide a useful tool for both novices learning about scheduling with algorithms and signallers for the ease of workload.

4.0 Further Development or Research

If given more development time, numerous features would have been highlighted for further development, increasing both the flexibility and functionality of the program. The biggest feature earmarked for future development would have been the ability to load different track layouts by the use of a template. This would have been a feature that would eliminate the need for manually implementing and modifying a track layout, as being able to take a template from the user and generate it dynamically would allow for much more widescale use. Another important feature would be redesigning the graph to include new and improved features in the track layout, such as junctions with set alignments, signals that change colour, stations with much more substantial graphics. This could also lead into a dynamic pathing feature which would see an algorithm pick a platform in a station based on the number of people and trains inside each platform.

A feature which would have dramatically increased the workload of the user but given much more flexibility in the case of an emergency would have been the ability to manually control a train and dynamically choose a custom destination while it is waiting to be sorted. This would allow on-the-fly sorting if there happened to be a problem with the train, tracks or any features on the graph proving a section of track inoperable.

A feature with a smaller priority but with the potential for expanding the scope of the application would be a scroll and zoom feature, allowing the track layout to be expanded and easily navigated. This would in turn allow for much more substantial track layouts, possibly for an entire train system, which would see trains guided by an algorithm from the start of their journey to the very end.

5.0 References

[2] Gilmore, J.F. and Elibiary, K.J., 1993. *AI in advanced traffic management systems.* Association for the Advancement of Artificial Intelligence (AAAI) Technical Report WS-93-04.

[3] Iarnrod Eireann, 2007, *RULE BOOK ISSUE 09/13.* Available at: https://www.irishrail.ie/Admin/IrishRail/media/Imported/Rule-Book-Complete-updated-January-2021.pdf [Accessed 05 May 2024]

[1] European Parliament, 2023, *What is artificial intelligence and how is it used?*, Available at: https://www.europarl.europa.eu/topics/en/article/20200827STO85804/what-is-artificial-intelligence-and-how-is-it-used [Accessed 05 May 2024]

[4] International Journal of Creative Research Thoughts (IJCRT), 2017, *AI Automation and It's Future in the United States*, Available at :http://www.ijcrt.org/papers/IJCRT1133935.pdf [Accessed 05 May 2024]

6.0 Appendices

6.1. Project Proposal

National College of Ireland

Project Proposal

AI Rail Signaller

27/10/23

Bachelors(Hons) in Computing

Software Development

2023/2024

Christopher Tobaras

X20324573

X20324573@student.ncirl.ie

Contents

1.0 Objectives

The projects’ goal is to achieve an AI based alternative to a role that is very crucial in the running of railways, the signaller. As the project aims to replace the human operator in this role it will have to take over its tasks, such as operating switches and signals in order to direct trains to their destination, optimizing train schedules by efficiently routing multiple trains at once, adhering to modern train operating rules and procedures, and deal with situations such as handling large amounts of traffic, delays, shunting and emergencies.

While the artificial intelligence aims to adhere to the rules set out by various regulations and operating procedures, it can also utilise machine learning to track certain trends in the network, such as predictive maintenance on railway infrastructure, times where traffic would get increasingly difficult to manage(I.E rush hour), and allowing the AI to relearn the network if the layout of track changes, such as a new railway line or improvements to existing lines.

Using artificial intelligence, we can further minimise the risk of human error in a role so vital for the running of the world railways. Being able to manage train operations, while being scalable and always improving can help improve safety, reliability, and cost effectiveness.

2.0 Background

As railways are a personal hobby of mine, I am quite knowledgeable about the history surrounding railway infrastructure around the world, and throughout the century, multiple technologies would revolutionise the way rail transport would be handled. Mechanical semaphores to modern LED signals, the transition from steam to diesel and afterwards electric, and the introduction of numerous safety systems have drastically improved and escalated railways into a competitive form of transport.

As we enter the modern age, with AI using machine learning to adapt to today’s needs, I was curious about the applicability of a scalable AI in an environment like this. In order to achieve this challenging task, I will employ machine learning algorithms, creating, and deploying AI models, data analysis and real-time processing to develop an intelligent signalling system capable to adapting to dynamic train schedules.

3.0 State of the Art

Existing applications such as traffic management systems have been experimented and tested using artificial intelligence and machine learning, however an application that has been designed to handle rail traffic has not been done before. Management applications that use predictive analytics to predict maintenance and cleaning schedules exist alongside models that fine tune schedules to be more efficient, however a rail traffic management system has not been implemented yet.

4.0 Technical Approach

The project will adopt an iterative development process. It will begin with requirement identification and gathering through research of operating procedures, and collaboration with experts. Tasks of the project will be broken down into phases which will start in the order of data collection, model development, simulation, and real-time testing. Activities relevant to each of these tasks may include data preprocessing, model training, system development and integration, and performance checks.

Identifying requirements will require the comprehensive research of existing rail infrastructure and operating procedures along with the collaboration of train experts. Using this information, I can better prioritize specific features and capabilities the AI must prosses in order to carry out its task.

Milestones will be defined by the integration of critical features in the system. For example, the working model will constitute a milestone, and the implementation of real-time processing could also count as another. The aforementioned project tasks at the beginning can allow for tracking the project more closely.

Another requirement worthy of note is the testing environment needed to measure the performance of the model. Research will be put in to find a signalling software capable of being controlled by the AI, or to create a new signalling software from scratch to simply demonstrate the AI's capabilities. This would require its own requirement gathering but will be easier to implement with previous research on the model being conducted already.

5.0 Technical Details

The decision is not final but the language most likely to be used in data preprocessing and creation of the model will be Python, alongside PyTorch and scikit-learn which will be used for the machine learning framework. The model will use a discrete rule-based system that will prioritize safety, which will also incorporate machine learning such as neural networks OR decision tree models in order to minimize risk, improve reliability and timeliness.

The program used to display the model's functionality will use Java and a graphics library to create a display to where the model can show its processing.

6.0 Special Resources Required

As mentioned previously, signaling software that human signalers use to route trains will be needed but that will either be found during the initial research of the project, or will be custom made for the model itself.

7.0 Project Plan

The project plan will be split into multiple phases, which follows a structured approach, focusing on iterative development and aforementioned milestones.

Weeks 1 and 2 start with the first phase which consists of researching technologies being used for the project, alongside refining requirements for features and the model being used. This should happen alongside research into signalling systems and the basics of machine learning systems in traffic management.

Week 3 will go deeper into a literature review, exploring scholarly articles, industry reports, and academic papers about signalling systems and machine learning in transportation.

Week 4 is the start of the second phase which involves the dataset collection and preprocessing of the data. Keep in mind that the real-time processing function of the AI will not be implemented until much later in 2024.

Week 5 signals the start of the third phase which will consist of the model selection and implementation of said model. It will use a small example of the dataset for use as training material. Week 6 and 7 follows suit with exploring the testing environment for the model either by adapting an existing software or creating a new one from scratch. This will be around the time the midpoint presentations will happen.

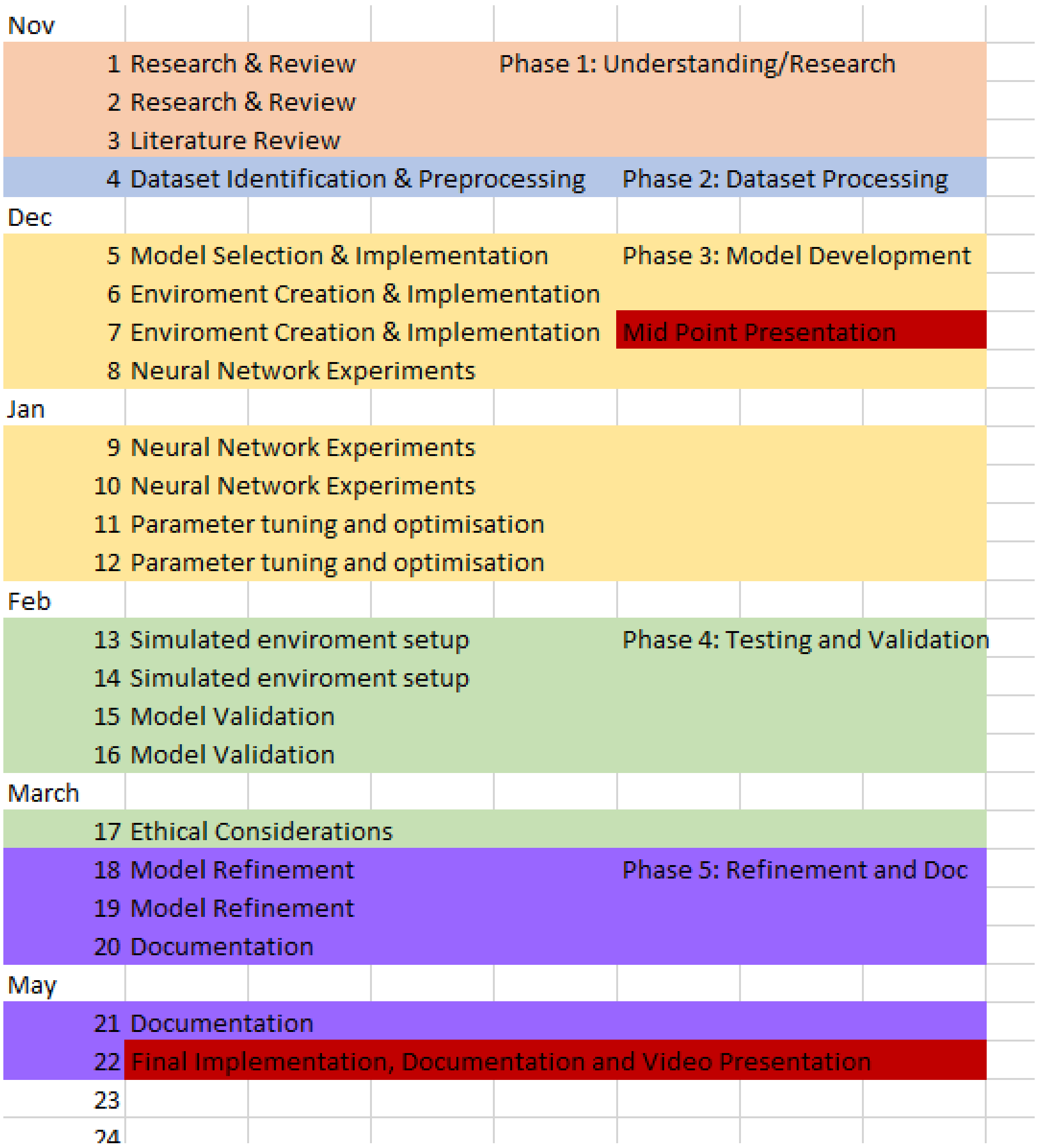
Week 8 will start with experiments in neural networks, which will continue to week 11 where we switch to parameter tuning and optimisation of the neural model for two weeks.

Weeks 13 and 14 starts the 4th phase and will try to merge the real-time data with the model, alongside merging it with the environment. This will be a big milestone as we have paired all of the necessary technologies together to make our first working prototype.

Weeks 15 and 16 will consider model validation, to make sure that the decision-making aspect of the model fits well within expectations. Week 17 will explore the ethical considerations of applying AI to rail management.

Weeks 18 and 19 will start the last phase and will refine the model itself and polish the environment, dataset and prepare anything else for the final product. This is also the part where documentation starts to be made and written down. Week 20 which is the final week of the final project will include polishing, bug fixes and further documentation, if not used for catching up.

The picture below is a graphic better explaining the structure of the plan.



Nov			
	1	Research & Review	Phase 1: Understanding/Research
	2	Research & Review	
	3	Literature Review	
	4	Dataset Identification & Preprocessing	Phase 2: Dataset Processing
Dec			
	5	Model Selection & Implementation	Phase 3: Model Development
	6	Enviroment Creation & Implementation	
	7	Enviroment Creation & Implementation	Mid Point Presentation
	8	Neural Network Experiments	
Jan			
	9	Neural Network Experiments	
	10	Neural Network Experiments	
	11	Parameter tuning and optimisation	
	12	Parameter tuning and optimisation	
Feb			
	13	Simulated enviroment setup	Phase 4: Testing and Validation
	14	Simulated enviroment setup	
	15	Model Validation	
	16	Model Validation	
March			
	17	Ethical Considerations	
	18	Model Refinement	Phase 5: Refinement and Doc
	19	Model Refinement	
	20	Documentation	
May			
	21	Documentation	
	22	Final Implementation, Documentation and Video Presentation	
	23		
	24		

8.0 Testing

Testing will be done in its environment and will be subjected to multiple schedules which all incorporate their own challenges and trends. The model will be graded on its ability to calculate paths in a fast enough time, reliability of the path chosen, the timekeeping of the system, the accuracy of the decision, safety, among other things. Whitebox testing can also be used to verify the workflow of the

6.2 Ethics Approval Application (only if required)

No ethics approval was required.

6.2. Reflective Journals

October:

Supervision & Reflection Template

Student Name	Christopher Tobaras
Student Number	X20324573
Course	Bachelors(Hons) in Computing
Supervisor	William Clifford

Month: October

What?
Reflect on what has happened in your project this month?
At the beginning of the month, I initially had an idea of creating a testing program that creates and manipulates a second cursor, that will test programs as a lightweight easy to download package and include performance statistics. However, during the week leading up to the video submission, I contacted a lecturer who agreed to become my supervisor, and advised me that the idea is not unique, and would take a lot of skill to create at its base form, which sounded too challenging. Instead, I decided to form another idea. The idea came when I was stuck outside Connolly station in a train as it was approaching the station. Humans still control switches and signals that let trains arrive in busy transit hubs like Connolly and Heuston every day. So, what it I was able to replace them using a Machine Learning model? Using this, I submitted my video proposal, talking about using AI to signal train traffic and control their movements using a strict rule system to ensure safety, but to allow Machine Learning elements to improve reliability and timeliness.
In the middle of the month, I discussed the idea with my supervisor further to find some common ground, as I knew a lot about the signalling protocols and practices of railways, while he knew about AI and Machine Learning. We agreed that a simple AI can be used along with rules to enforce a 100% safety rate, and that we could allow the AI to train itself on test data to see if it could get trains quicker into stations, improving timeliness. We also discussed the environment this AI would have to test itself in, as I needed to show this during the final project expo. Finally, we discussed the languages and technologies that we could utilise to make this project a reality.
At the end of the month, I spent the time creating the written project proposal, writing about the clarifications learnt during my supervisor meetup, and elaborating on the technologies used and the environment that is planned to be used on the project. The feedback for the idea was great, although my explanation about the projects goal and motivations were not well explained.

So What?
Consider what that meant for your project progress. What were your successes? What challenges still remain?
I consider this a huge step in the right direction, as I was not completely on board with my last idea, but since railways are a hobby to me, this new idea took off way quicker than expected, which lead to more streamlined ways of research and development. Early participation with my supervisor also proved helpful, as the feedback earned throughout the meetings helped me hone and tweak my idea to allow it to meet bigger goals and focus my scope. The project proposal also allowed me to create a weekly plan for the final project, on what I should do every week in order to move the project along and keep myself on track.

Now What? What can you do to address outstanding challenges? The only challenge to address right now is research. Searching academic papers on this subject is hard, as there is little to no papers on subjects like this in particular, although similar things do exist, such as the use of AI in road traffic. The first two weeks of November are planned to be full of academic research about the topics my project is based on so that should tackle the issue.

Student Signature	[illegible]

November:

Supervision & Reflection Template

Student Name	Christopher Tobaras
Student Number	X20324573
Course	Bachelors(Hons) in Computing
Supervisor	William Clifford

Month: November

What? Reflect on what has happened in your project this month? In the month of November, a lot of research was conducted on how the program would function, and what its true purpose might be compared to other different approaches. At this time, I also decided on the technologies used in the development of the project. The first week I contacted my supervisor and we both agreed that this application can be used as a tool for the signaller in order to free up tasks for a rather demanding job. This was a good idea which meant that AI could have a place in railway signalling but also the human operator can deal with aspects not covered by the artificial intelligence such as communicating to trains and handling complex manoeuvres. At the same time, during the middle of November I found a new signalling software that proved compatible with the project, called JRMI. This was a java-based railway modelling software that was open source and perfect for use. With this I started work on some simple flows and architecture diagrams, to understand how the application would interact with the user and the environment. By the end of November, I had finalised the technology needed for the project to happen. The application would be coded in Python that will utilise a control panel for the user to use. The model is not visible but can be interacted with the control panel to either start or stop the system, print logs, feed schedules, and debug the performance of the algorithm. With this I started on the wireframes, and then started work on the Use Case Diagrams.
So What? Consider what that meant for your project progress. What were your successes? What challenges

still remain? This month was a huge leap in research and development for the project. Weeks of theorising and investigating showed that artificial intelligence did really have a place in railway signalling and that the technology to implement it was possible. Using the test environment and various diagrams drawn over the month, the development and deployment of a prototype for the midpoint seems within reach.
Now What? What can you do to address outstanding challenges? In terms of challenges, the development of the prototype itself is the biggest challenge. As the API and the algorithmic model are very complex, they most likely would not be able to fit the development timeframe I have until the midpoint. So, there must be a compromise to allow as much functionality while also allowing the program to function without these features. Perhaps a bigger focus on the visual aspects of the program could be satisfactory.

Student Signature	

December:

Supervision & Reflection Template

Student Name	Christopher Tobaras
Student Number	X20324573
Course	Bachelors(Hons) in Computing
Supervisor	William Clifford

Month: December

What? Reflect on what has happened in your project this month? In December, work started on the prototype used for the demonstration in the project midpoint for Christmas. Before commencing work, I had to identify the full list of requirements and decide what features the prototype should include in order to meet at least one main user requirement. As the program is designed to incorporate a model and use APIs to decipher the track layout of another program, this would not be included as the time constraints would not allow for them. Instead, the prototype would feature a working GUI, with plans to incorporate basic user features and create mock-ups of features that are not present in the program. The application would simulate the turning on/off of the program and checks to see if an example window(acting as the environment) is open. The ability to feed a schedule and error popups to notify the user of an exception was also implemented, however viewing logs, using the debug menu, and manually controlling the application was not able to be added. Outside of programming, further research was considered in respect of the technical report. User, environmental, usability and data requirements were identified and elaborated upon. Diagrams such as architecture diagrams to

identify layout structure, use case diagrams to describe flows and wireframes to elaborate on system features were created and added into the document.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? So far, the progress has been at a steady pace. The research has not yet been fully completed however progress can be seen in other areas such as implementation, model choice, how the applications will communicate between each other and rules on how to safely find a path for trains. The challenge that remains now is to continue into incorporating and training a model to find a path to a certain node.
Now What? What can you do to address outstanding challenges? In the new year, work will be done with PyTorch to implement a neural network that will be trained with example data in finding a path to a certain object, afterwards the rules researched will be implemented into the model and further training and refinement will commence.

Student Signature	

January:

Supervision & Reflection Template

Student Name	Christopher Tobaras
Student Number	X20324573
Course	Bachelors(Hons) in Computing
Supervisor	William Clifford

Month: January

What? Reflect on what has happened in your project this month? In January, work was halted due to the upcoming January assignments and exams, including the Christmas and New years holiday. With the time taken away to these events, this left me with a more than a week's worth of time to work on the project. I started by reading more into the documentation surrounding the JRMI software in which my project was to be involved with. With this, I learnt the ins and outs of how to operate the program, such as to build custom layouts, build and place signals, waypoints, junctions, and other important items such as junctions. Afterwards, I researched the topic of PyTorch and began to slowly understand the process of setting up and running a model. As the time frame for this month was shorter than the other months, Integration between the software and JRMI, and the inclusion of a model had to be delayed to the middle of this semester.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? At the moment, I consider the progress done this month to be a good foundation for what will happen in February. Grabbing a good knowledge about the software I plan to integrate, alongside

information on how to develop and deploy an AI model has definitely risen some ideas and issues about different topics that I would have to tackle.
Now What? What can you do to address outstanding challenges? In February, I plan on diving into the Java Railway Modellers Interface software and begin implementing APIs to control switches, signals, and different objects. From this, I will also research further about AI models and begin implementing one by the end of the month.

Student Signature	

February:

Supervision & Reflection Template

Student Name	Christopher Tobaras
Student Number	X20324573
Course	Bachelors(Hons) in Computing
Supervisor	William Clifford

Month: February

What? Reflect on what has happened in your project this month? In the month of February, work was done towards the future priorities and potential pathways the programming section of the project could take. Initially work was done in researching how a machine learning algorithm could interpret track layouts and based off training data, come up with an answer which would fulfil the reward conditions. However, it was then decided that work would be done towards the framework of the project, as this section was deemed more important. A decision was made to change the UI interface to have the layout displayed on the screen itself, and potentially manually controlled by the user to allow training of the AI model.
So What? Consider what that meant for your project progress. What were your successes? What challenges still remain? I consider this great progress, as much information and research on the implementation of the application has been done, and due to holidays and outside factors, this has hampered development quite a bit. Research and discussion surrounding the application has been intriguing as different aspects have been considered and multiple pathways have been either put forward or shut down.
Now What? What can you do to address outstanding challenges? To overcome the challenges, more work must be done in terms of the framework, and researching the open-source software, JRMI. With these two pathways, a decision could be made on whether to modify the existing JRMI software to train the model, or to pursue development on a standalone application that would connect with JRMI to create a custom layout the model could control.

Student Signature	

March:

Supervision & Reflection Template

Student Name	Christopher Tobaras
Student Number	X20324573
Course	Bachelors(Hons) in Computing
Supervisor	William Clifford

Month: March

What?
Reflect on what has happened in your project this month?
In the month of March, substantial programming work was started on the application. The program was redeveloped from scratch as the prototype had design flaws which were not able to fit within the time frame given. The software is now written in java, and the libraries used to show the layout are JGraphX, and JavaFX. Work was done in generating the GUI, its associated windows and with the generation of the layout. Restructuring work in the form of how the GUI looks and the file structure was done, and work was done to implement up to date coding standards such as abstraction, and keeping methods simple.

So What?
Consider what that meant for your project progress. What were your successes? What challenges still remain?
This was great progress the project. As the rewritten version of the prototype has already been completed and features are being developed, the application is slowly becoming a complete product with the numerous features being added. The logic to add trains within the layout are still being implemented, and will be for the next week while also making sure to implement a traversal algorithm in order to successfully complete the applications purpose.

Now What?
What can you do to address outstanding challenges?
To overcome the challenges, work will be done to address the currently missing features in the application. As assignments have been completed, development time will be significantly increased, which will help towards the implementation of more features which will make the application a more complete product.

Student Signature	

April:

Supervision & Reflection Template

Student Name	Christopher Tobaras
Student Number	X20324573
Course	Bachelors(Hons) in Computing
Supervisor	William Clifford

Month: April

What?
Reflect on what has happened in your project this month?
In the month of April, more substantial work was done in implementing the fundamental features of this project. As the GUI and graphing interface was created last month, work was done in generating a transactional model which would handle trains on this graph. This followed by implementing a pathfinding algorithm so that the train would be able to find a path to the correct node without breaking safety rules. After this, a time system was implemented which added a clock to the program, and a method to add trains when a certain time was met. From this, the schedule view was also implemented, using a singleton instance as a messenger, which could allow the user to add trains from a CSV file. Finally, work was done in showing the necessary highlighting of paths for the trains and adding a locking system in order for trains to respect already highlighted paths by waiting a few seconds.

So What?
Consider what that meant for your project progress. What were your successes? What challenges still remain?
This has been excellent progress in my opinion as compared to last month, more features that implemented the necessary use-cases defined in the mid-point were achieved in this month compared to any other month in the final project. Much needed experience was gained, both in coding practices, usage of different topics in code, and interaction with testing and continuous integration such as using GitHub to store my code. The features implemented represent an almost fully working version of the final result and only three features remain as of now. These challenges include bi-directional pathfinding, train persistence when entering/exiting the layout, and the exporting/importing of algorithms by using the cloud.

Now What?
What can you do to address outstanding challenges?
These challenges mentioned before will be attempted to be fixed by the end of the first week, as the priority of the final project will be moved over to the documentation/academia section, which will involve research and explanation of methods used during the final project. Once the first week ends, work will be done in researching articles and documenting the design process of the application in order to present it in a suitable form for submission.

Student Signature	