OS Documentation

Rat in a maze

مصطفي محمود ابراهيم محمد :20210946-1

معاذ محمود حامد :20210952-2

ندى كامل عبدالعال احمد : 20210999 -3

منة الله شريف مصطفي :20210958-4

منة الله منصور السيد :20210963-5

ياسمين احمد ابراهيم مصطفي :20211044-6

Project Overview:

The Rat in a Maze problem involves finding a path through a binary matrix from the upper left cell to the lower right cell. The maze contains blocked (0) and open (1) cells, and the rat can only move forward or down. The challenge is to solve this problem using multithreading.

Project Features:

1-input Dimension: The program allows the user to input the dimensions of the maze (𝑁).

2-Dynamic Grid Generation: A grid of dimensions 𝑁 × 𝑁 is generated based on user input.

3-Runtime Dead Blocks Specification: The user can dynamically specify dead blocks in the maze during runtime.

4-Problem Modeling:

-The maze is modeled as a matrix with the start at the upper-left cell and the end at the lower-right cell.

-The rat can only move forward or down in the maze.

5-Multithreading:

Solution Threads are used to explore possible directions in the maze concurrently. If a thread finds two possible directions, it continues in one direction and creates a new thread for the second direction. The number of threads is limited based on the number of processors.

6-GUI:

Input:

Maze size (𝑁)

Output:

Real-time visualization of the rat's path in the maze.

The GUI updates in real-time as the rat explores the maze.

Implementation Details

1-Thread Creation:

Threads are created to handle the exploration of different paths concurrently.

The number of threads is managed based on the available processors.

2-Dynamic Grid Modification:

The grid is dynamically modified during runtime as the user specifies dead blocks.

1-Real-time GUI Update:

The GUI is updated in real-time to show the progress of the rat in the maze.

Paths explored by different threads are visualized as the rat traverses the maze.

Code and Structure of Classes:

1-RatMazeDriver

2-MainFrame

3-MainView

4-GridFrame

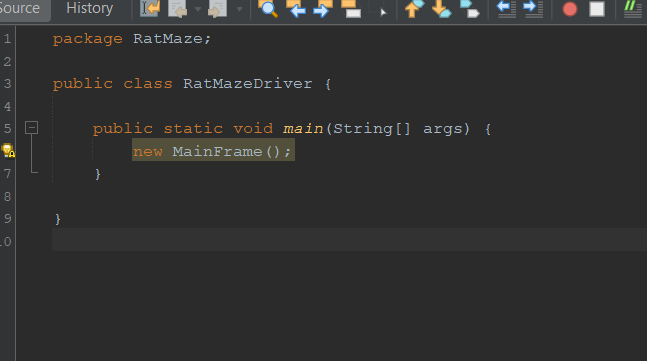
5-GridView

6-SolutionFrame

7-SolutionView

8-DFS

1-RatMazeDriver: Run the Program and call main frame class



2-MainFrame:

The `Mainframe` class serves as the main frame for the Rat Maze Game application. It is responsible for configuring the frame settings, setting the application icon, and adding the main user interface (UI) represented by the `Main View` class to the frame.

Key Features:

- The frame title is set to "Rat Maze Game."

- The application icon is configured using an image named "RatMazeIcon.png."

- The frame is set to initially maximize to the full screen.

- The frame is resizable.

- The main view, an instance of the `Main View` class, is added to the frame's content pane.

- The frame is set to be visible.

- The layout is defined as a Border Layout.

- The frame's default close operation is set to exit the application when closed.

A screen shot of a computer

Description automatically generated

3-MainView:

The `MainView` class represents the main user interface panel for the Rat Maze Game application. Here is a brief overview of its functionality:

1-User Interface Components:

- Several UI components are defined, including labels (`welcome`, `dimension Label`, `note Label`), a text field (`dimensions Field`), and a button (`buildItButton`).

- A custom `Buttonlike` listener is instantiated to handle button clicks.

2-Background Image:

- The class loads a background image ("moazz2.jfif") and paints it on the panel using the `paint Component` method.

3-UI Styling:

- Font styles, colors, and layout bounds are set for the UI components to customize their appearance.

4- Button Click Handling:

- The `OnButtonClick` class implements `ActionListener` to respond to the "Build it!" button clicks.

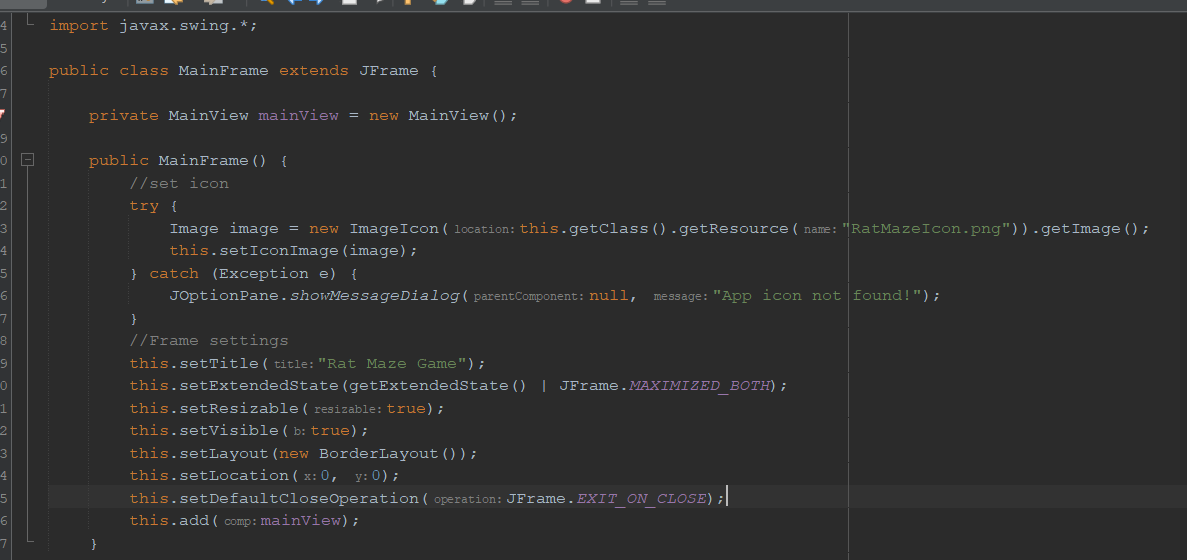
- On button click, the program attempts to parse the entered dimensions from the text field.

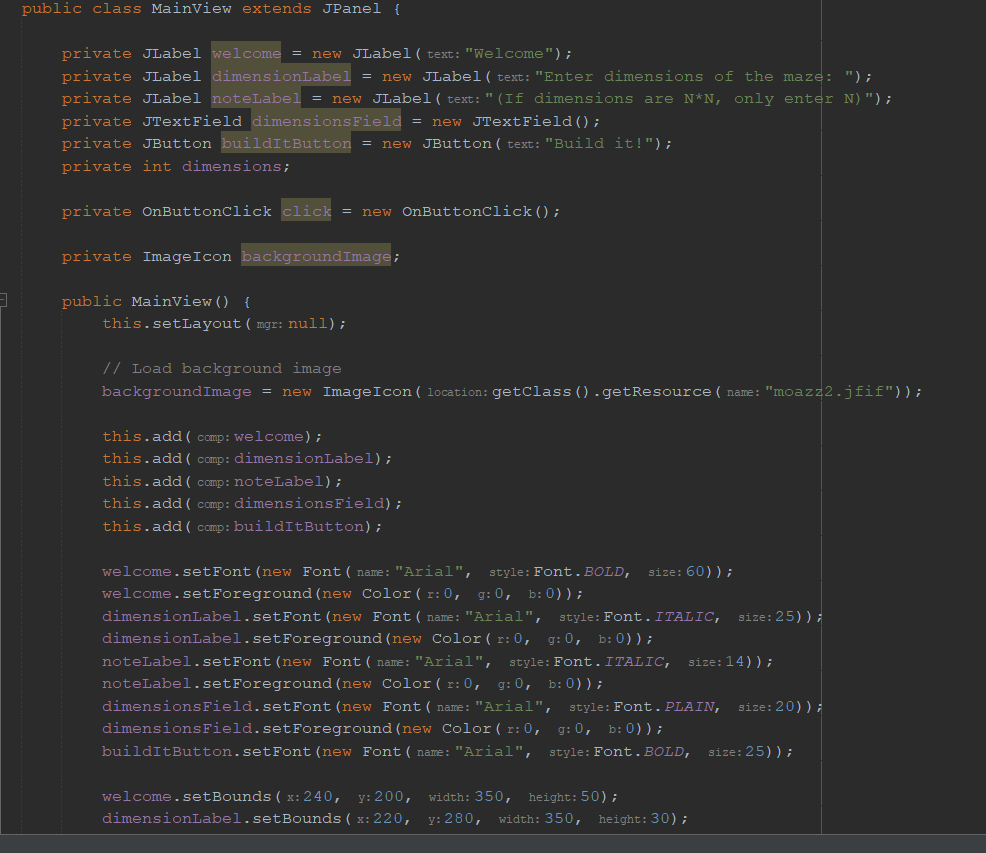
- If valid dimensions (greater than or equal to 3) are provided, the current panel is removed, and a new `GridFrame` with the specified dimensions is created.

- The current frame is disposed of (closed).

5- Exception Handling:

- Exception handling is in place to catch and display any unexpected errors during the button click event.





4-Grid Frame:

The GridFrame class represents a JFrame that serves as the container for the maze grid in the Rat Maze Game application. Here's a brief summary of its functionality:

1- Frame Initialization:

- The class extends `JFrame` and is constructed with a specified dimension for the maze (`dimensions`).

- Attempts are made to set the application icon using an image file ("RatMazeicon.png").

2- Frame Settings:

- The frame title is set to "Maze Grid."

- The frame is configured to be maximized on startup and is set to exit on close.

3- Background Color:

- The content pane's background color is set to white. You can customize this color by changing the `Color.WHITE` argument to a color of your choice.

4- Grid View Integration:

- An instance of the `Grid View` class, responsible for rendering the maze grid, is created (`grid View`).

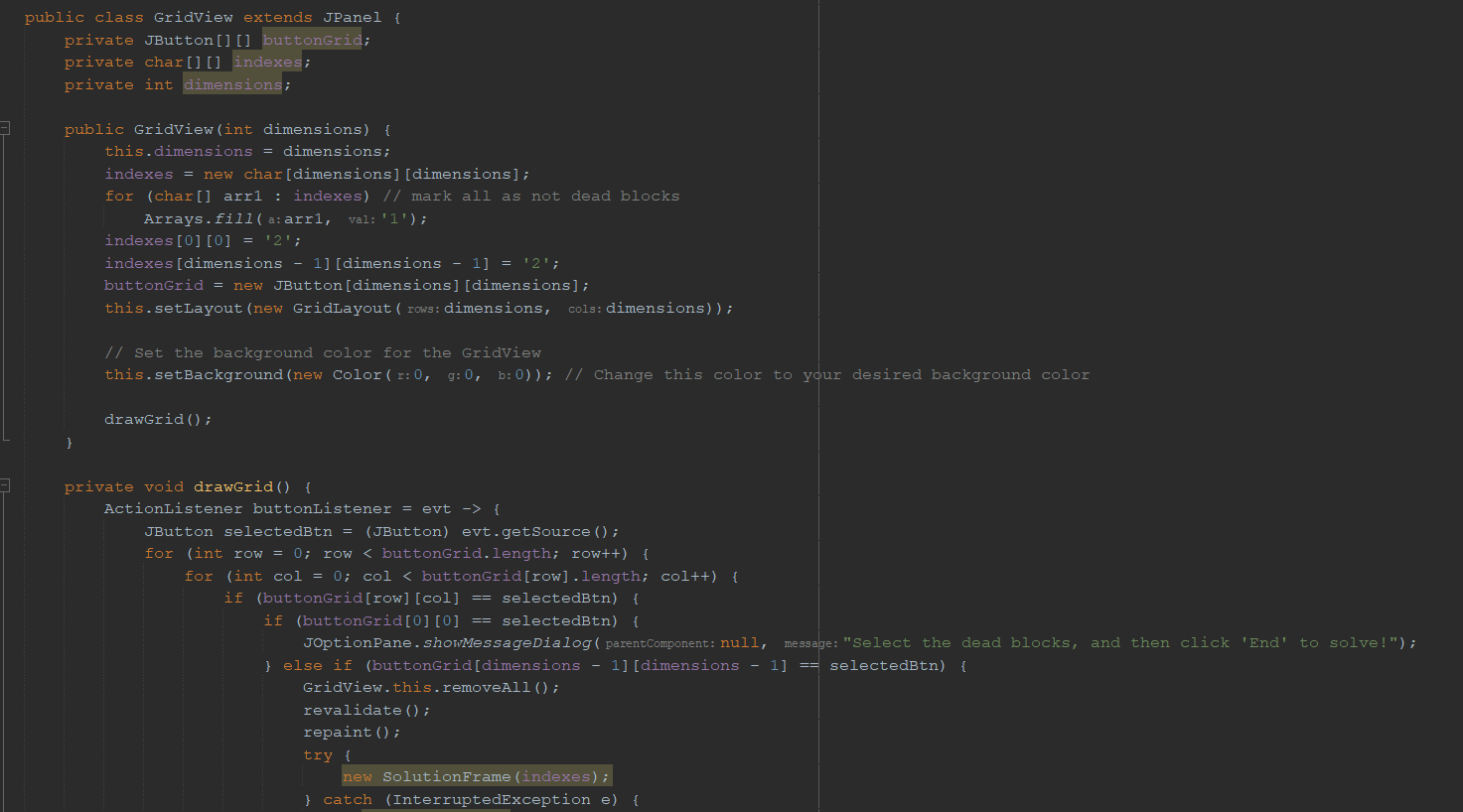
- The `Grid View` is added to the frame using the `add` method.

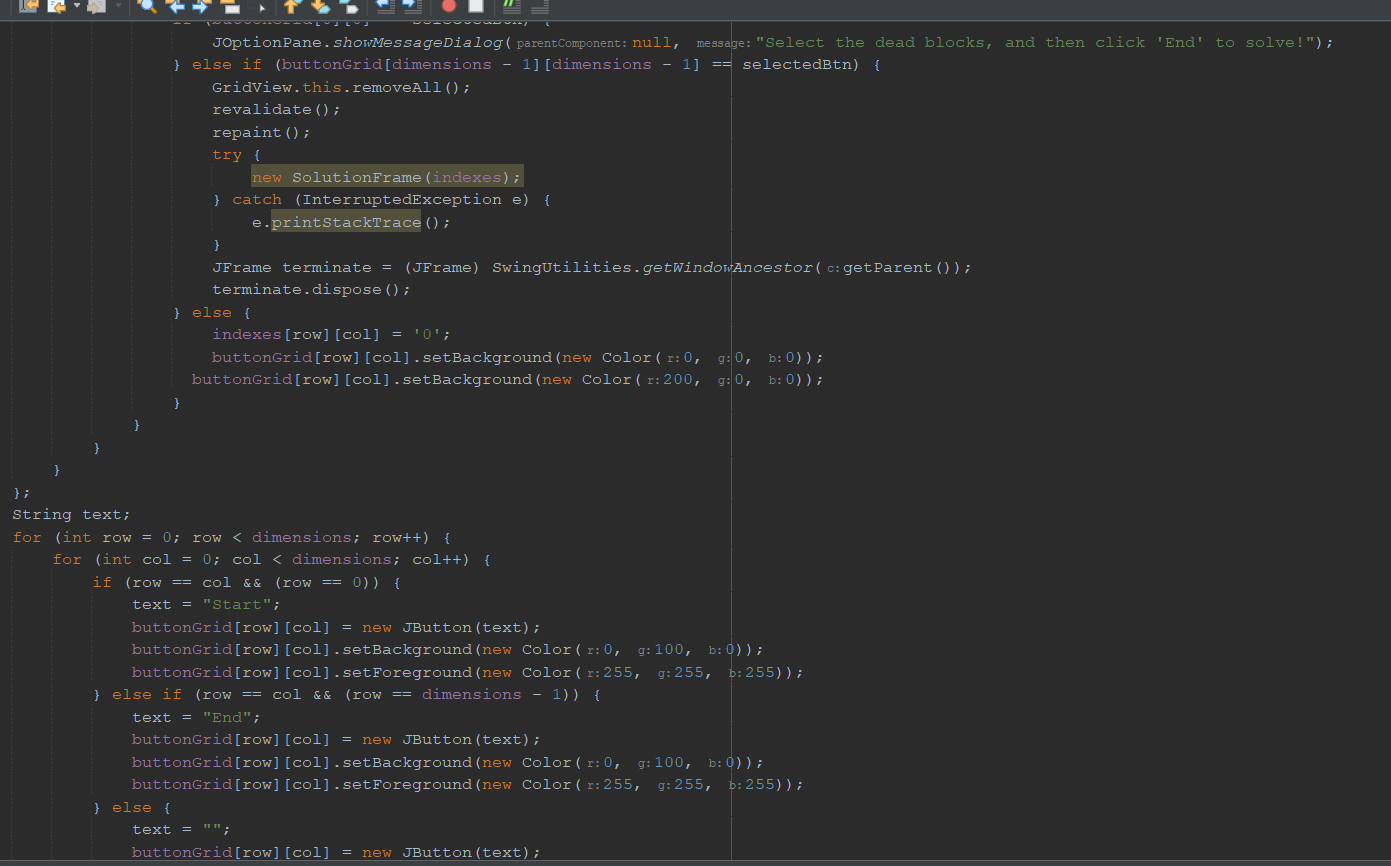
5- Frame Visibility:

- The frame is set to be visible.

6- Main Method:

- The class includes a `main` method, which, when invoked, creates an instance of `GridFrame` with a default dimension of 10. You can adjust the dimension as needed.





5-Solution Frame

The `SolutionFrame` class represents a JFrame that displays the solution to the Rat Maze Game. Below is a brief overview of its functionality:

1-Frame Initialization:

- The class extends `JFrame` and is constructed with a 2D character array (`indexes`) representing the solution path.

2- Frame Settings:

- Attempts are made to set the application icon using an image file ("RatMazeIcon.png").

- The frame title is set to "Maze Solution."

- The frame is configured to be maximized on startup, resizable, and set to exit on close.

3- Layout Configuration:

- The layout of the frame is set to use `BorderLayout`.

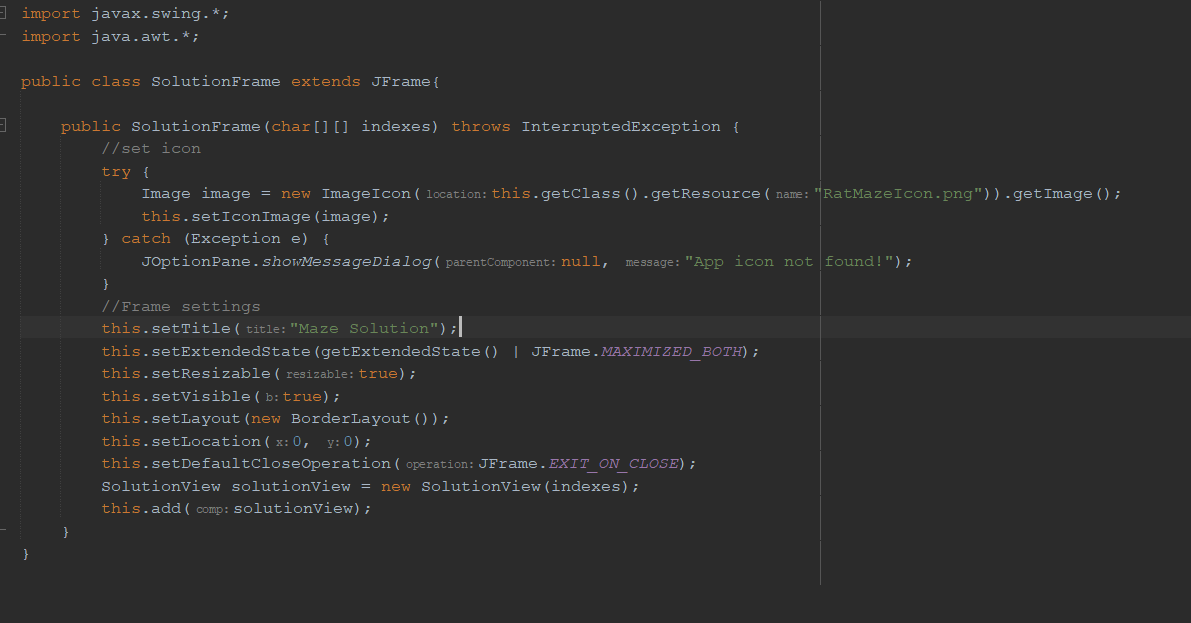
4- SolutionView Integration:

- An instance of the `SolutionView` class, responsible for rendering the solution path based on the provided indexes, is created (`solutionView`).

- The `SolutionView` is added to the frame using the `add` method.

5- Frame Visibility:

- The frame is set to be visible.



6-SolutionView:

The `SolutionView` class represents a JPanel that displays the solution to the Rat Maze Game. Here is a brief overview of its functionality:

1-Initialization:

- Takes a 2D character array `indexes` as a parameter, representing the solution path.

- Initializes button grid, dimensions, and a boolean flag (`found`) indicating whether a solution was found.

2- Background Computation:

- Sets up a grid layout for the panel and initializes colors based on the values in the `indexes` array.

- Applies a depth-first search (DFS) algorithm (`DFS\_Class`) to find the solution path, marking the visited cells.

- Modifies the `indexes` array to reflect the solution path.

3- Draw Grid:

- Iterates through each cell in the grid, creating buttons with appropriate labels ("Start," "End," or empty string).

- Sets button colors based on the values in the `indexes` array, highlighting the start, end, and solution path.

- Attaches an ActionListener to each button to handle button clicks.

4- Button Listener:

- Responds to button clicks by displaying appropriate messages.

- If "Start" is clicked, it recreates the main frame and disposes of the current frame.

- If "End" is clicked, it disposes of the current frame.

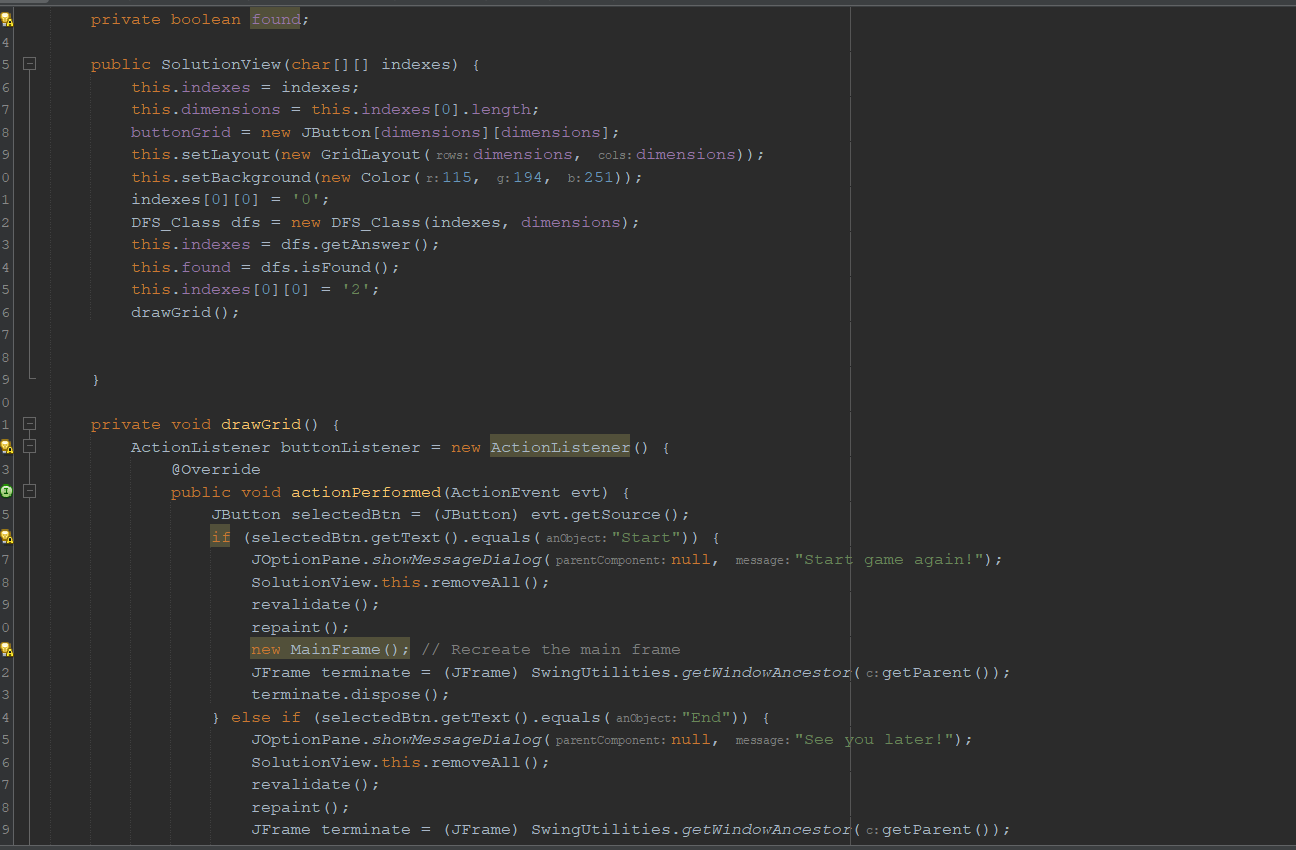
- If other buttons are clicked, it displays a message indicating whether a solution path was found.

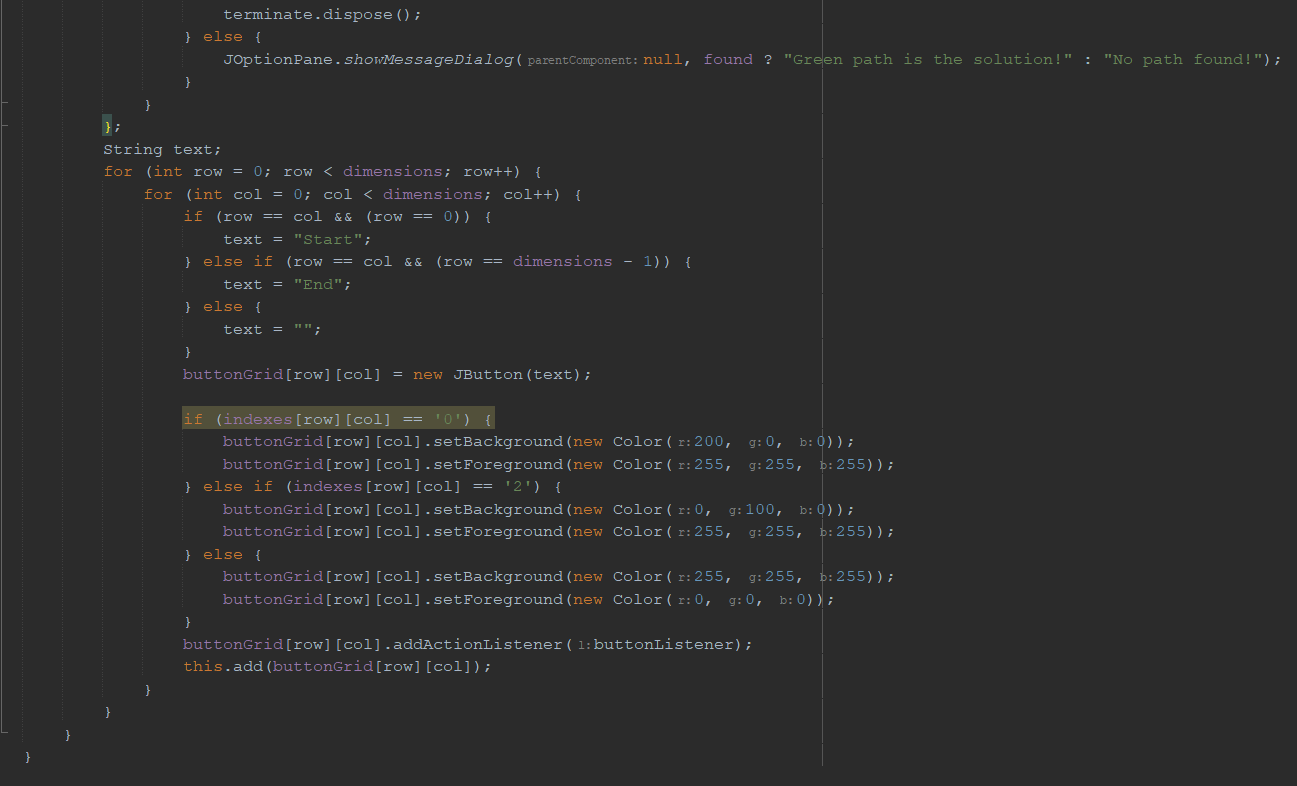
5- Color Codes:

- Red color: Represents walls (value '0' in `indexes`).

- Green color: Represents the solution path (value '2' in `indexes`).

- White color: Represents empty cells.





7-DFS Class:

The `DFS\_Class` represents a depth-first search (DFS) algorithm implemented in a threaded manner to find a solution path in a maze represented by a 2D character array. Here's a brief overview of its functionality:

1. Initialization:

- Takes a 2D character array `indexes` and its dimensions as parameters.

- Initializes the graph, dimensions, a 2D boolean array for visited nodes (`traversed`), and other necessary variables.

- Initializes an ArrayList of `Pair` objects (`Answer`) to store the solution path.

2. DFS Implementation:

- Utilizes a threaded DFS approach, where each thread explores potential moves concurrently.

- The `RealDFS` inner class represents the actual threaded DFS implementation.

- Threads are created to explore neighboring cells concurrently, and each thread maintains its own path (`answerList`).

- The `run()` method is responsible for invoking the DFS algorithm in a threaded manner.

- Each thread explores potential moves until the solution is found or the destination is reached.

3. DFS Algorithm:

- Recursive DFS algorithm explores neighboring cells in the graph.

- If a solution is found, the `Answer` is updated, and the `found` flag is set to true.

- Threads are created based on the available active thread count, with a maximum limit (`MAX\_THREADS`).

- Threads are started using the `start()` method to enable concurrent execution.

4. Init DFS Method:

- Invoked only once when `MAX\_THREADS` value is more than 2.

- Initiates the DFS process from the starting point.

5. getAnswer() Method:

- Waits until all threads finish before proceeding.

- Returns the graph representing the maze, modified to include the solution path.

6. isFound() Method:

- Returns a boolean indicating whether a solution path was found.



