



SOFTWARE **ENGINEERING AND** **PROJECT MANAGEMENT** **LAB**

LAB FILE SUBMITTED BY:

NAME: SAKSHAM GOEL
SAP ID: 500107184
BATCH: 3 (AI-ML (NH))

SUBMITTED TO:

Dr. Arjun Arora

Index

S. No.	EXPERIMENT	DATE	Remarks
1.	Perform requirement analysis and to find the requirement specification of a given Problem.	25/01/2024	
2.	To perform the function-oriented diagram using a structure chart: Flowchart.	08/02/2024	
3.	To perform the user's view analysis for the given system – Use case diagram	29/02/2024	
4.	To develop ER diagram to showcase structural view analysis	14/03/2024	
5.	To develop Sequence diagram to showcase behavioral view analysis	14/03/2024	
6.	To develop Software Requirement Specification (SRS) for taken up project	21/03/2024	
7.	To perform various testing using the test cases for unit testing, integration testing and system testing for a given project.	18/04/2024	
8.	To implement the given project as per the set of objectives.	22/04/2024	
9.	To prepare PERT Chart for given project.	22/04/2024	

Experiment 1: Perform requirement analysis and to find the requirement specification of a given Problem.

Project Title:	Maze Solving Program
Team Size:	4
Team Member Names:	Saksham Goel Sharad Verma Rishita Kumar Ria Singhal

Requirements established after multiple brainstorming sessions:

Representation of maze:

The maze will be created in the form of a 2D array. The array will only contain 0s and 1s. 0 will represent wall (or closed cell) and 1 will represent an open cell.

Static input:

The size of the maze (or array) will be fixed, i.e., 6*6. Also, the arrangement of 1s and 0s in the array (i.e., structure of maze) will be static.

Purpose:

The purpose of the program is to determine the correct path(s) of the maze from the starting point (0,0) to ending point (5,5) and display it in the output.

Single-player:

The program will be single-player and would require the user to enter (or edit) the static inputs of 1s and 0s for the maze.

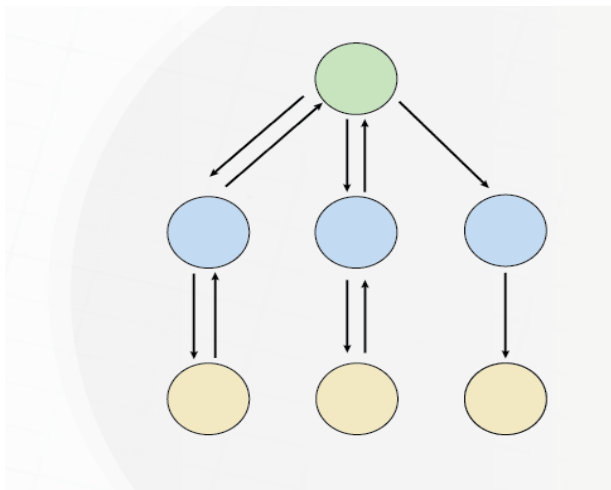
Requirements mentioned above will be achieved by using the following:

The above requirements will be achieved by creating the program in Java.

Version: 21.0.1

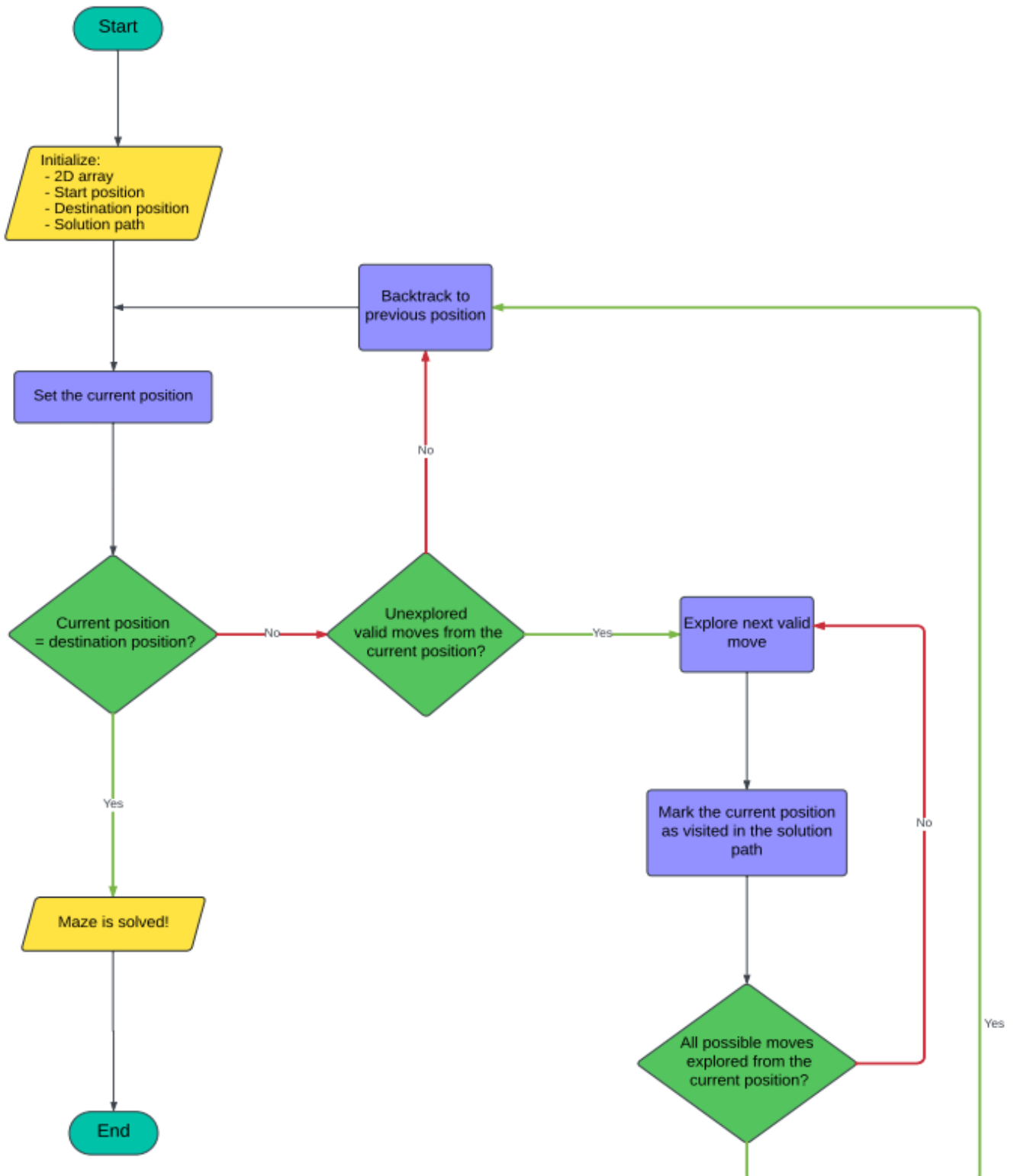
Backtracking Algorithm is the algorithm which will be implemented in our Java program to solve the problem.

A backtracking algorithm is a problem-solving algorithm that uses a **brute force approach** for finding the desired output.



The term backtracking suggests that if the current solution is not suitable, then backtrack and try other solutions. Thus, recursion is used in this approach.

Experiment 2: To perform the function-oriented diagram using a structure chart:



Explanation:

Initialization:

A 2D array representing the maze is created.

The starting and ending positions within the maze are defined.

A variable to store the solution path is initialized.

Set the current position:

The current position is set to the starting position.

Check if the current position is the destination:

If the current position is the same as the destination position, then the maze has been solved, and the solution path is returned.

Explore next valid moves:

If the current position is not the destination, then it will check if all of the valid moves from the current position are explored. A valid move is one that is not blocked by a wall and has not been visited before.

Backtrack to a previous position:

If there are no more valid moves from the current position, then the algorithm backtracks to a previous position by removing the most recent move from the solution path and setting the current position to the previous position.

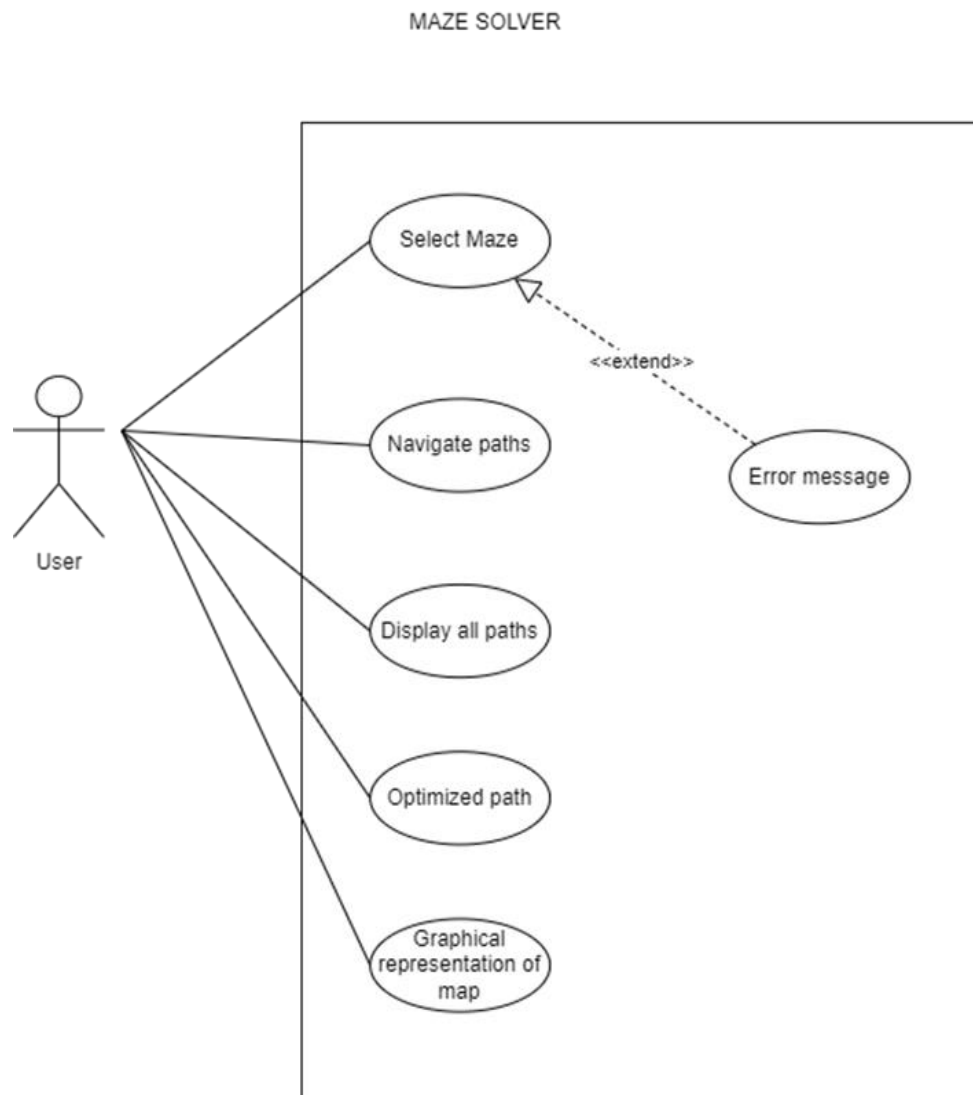
Mark the current position as visited:

If a valid move is found, then the current position is marked as visited in the solution path to prevent the algorithm from exploring the same position again.

Repeat steps 2-6:

Steps 2-6 are repeated until the maze is solved or all possible moves have been explored from the starting position, in which case there is no solution to the maze.

Experiment 3: To perform the user's view analysis for the given system - Use case diagram.



Explanation:

Actors

User: This represents the person interacting with the maze solver system.

Use Cases (Actions the user can take)

Select Maze: This use case allows the user to choose a maze to be solved.

Navigate Paths: This use case represents the core functionality of the system. It allows the user to navigate through the chosen maze.

Display all paths: This use case enables the user to view all possible solutions to the maze.

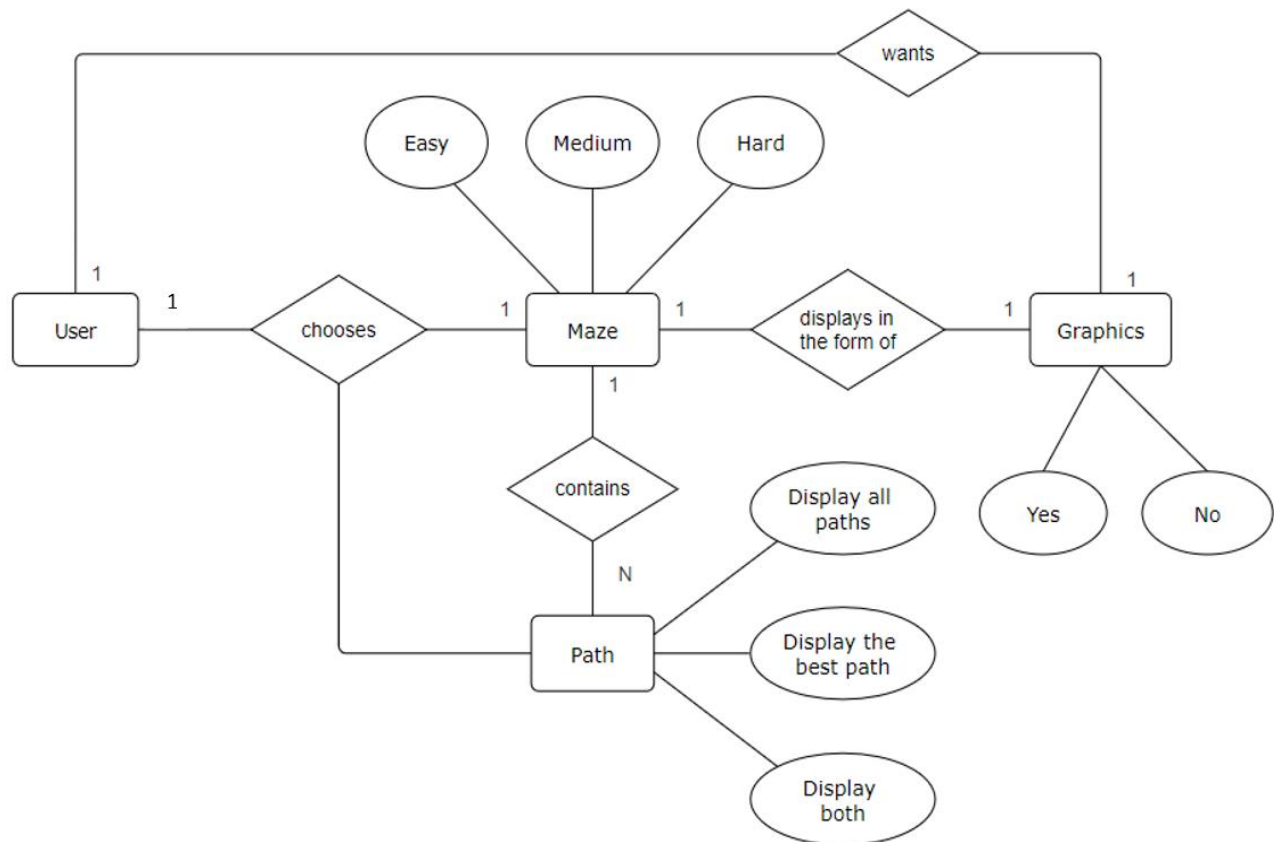
Optimized Path: This use case provides the user with the most efficient path to navigate through the maze.

Graphical representation of map: This use case shows the user a visual representation of the maze.

Relationships

<<extend>>: The diagram shows an extend relationship between "Select maze" and "Error message". This indicates that the "Error message" use case is an optional extension of "Select maze". This means that the system might display an error message during the selection of maze.

Experiment 4: To develop ER diagram to showcase structural view analysis



Explanation:

Entities

User: Represents the user interacting with the maze-solving system.

Maze: Represents the mazes available for solving. Each maze has a difficulty level (easy, medium, or hard).

Path: Represents the different paths within a maze.

Graphics: Represents the visual representation of the maze.

Relationships

User chooses Maze: Indicates that a user selects a specific maze.

Maze contains Path: Shows that a maze can contain multiple paths.

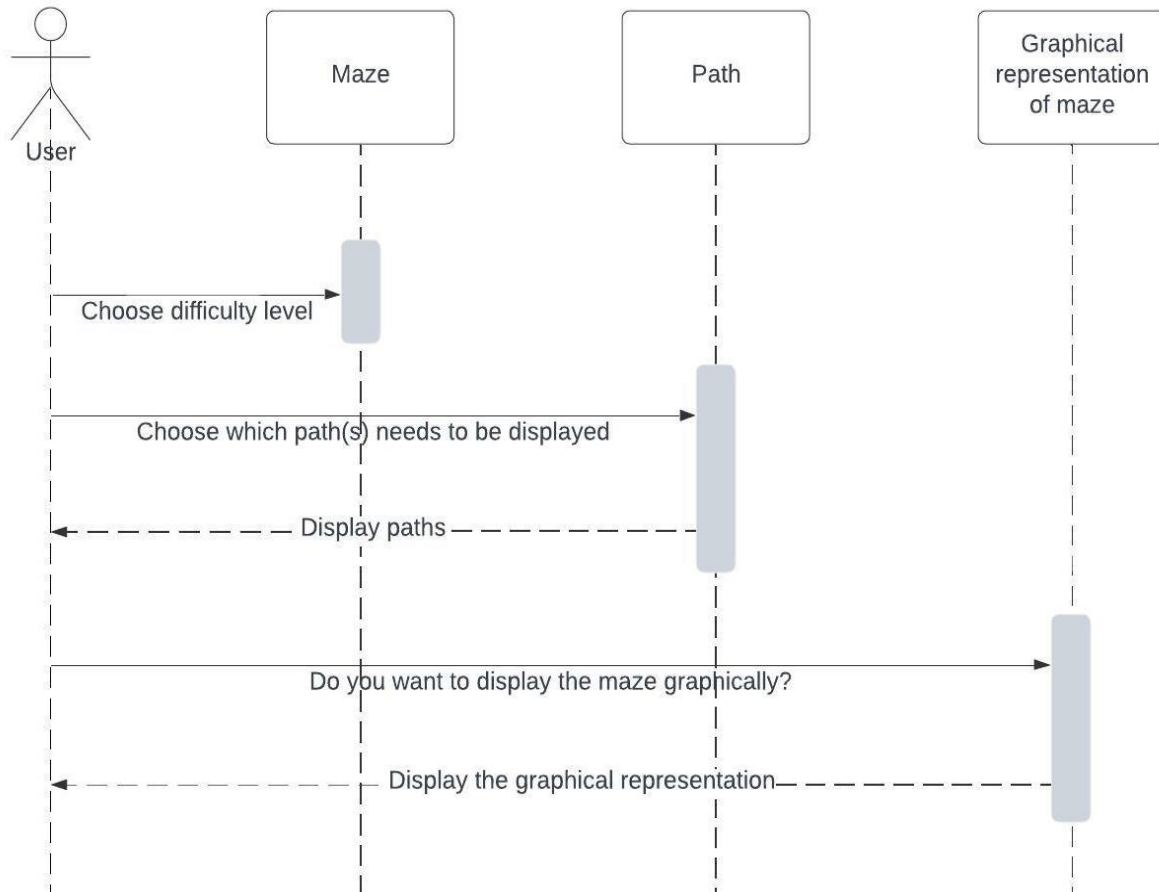
Maze displays in the form of Graphics: Indicates that mazes can be visualized graphically.

User chooses Path:

- Display all paths: Allows the user to view all available paths.
- Display the best path: Shows the optimal path.
- Display both: Provides both the complete set of paths and the best path.

User wants Graphics: Signifies that whether the user wants a visual representation of the maze to be displayed or not.

Experiment 5: To develop Sequence diagram to showcase behavioral view analysis



Explanation:

Entities and Actions

User: Represents the individual interacting with the maze-solving application.

Maze: Refers to the maze itself, which the user aims to solve.

Path: Represents the possible paths within the maze.

Graphical representation of maze: Indicates how the maze can be visually displayed.

Processes

Choose Difficulty Level: The user initiates interaction by selecting the difficulty level of the maze (e.g., easy, medium, or hard).

Select Paths to Display: Next, the user decides which path(s) they want to see. This step involves choosing specific paths within the maze.

Display Paths: The application displays the selected paths to the user based on their choice.

Graphical Representation (Optional): The user has the option to view the maze graphically. If they choose to do so, a graphical representation of the maze is shown.

Experiment 7: To perform various testing using the test cases for unit testing, integration testing and system testing for a given project.

(a) Unit Testing:

Test Case 1: When user is prompted to enter 'start'		
Inputs	Expected Output	Actual Output
1) strt, str, or any other string (except 'start')	Program not executed. You didn't enter 'start'.	Program not executed. You didn't enter 'start'.
2) start	(The program shall run normally and ask the user to enter next input) Choose the difficulty level: 1. Easy 2. Medium 3. Advanced Enter your choice:	Choose the difficulty level: 1. Easy 2. Medium 3. Advanced Enter your choice:

Test Case 2: When the user is prompted to enter the difficulty level of maze		
Inputs	Expected Output	Actual Output
1) 1, 2 or 3	(The program shall run normally and ask the user to enter next input) Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:	Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:
2) 4, 5, or any other input (except 1, 2, 3)	Invalid option. Exiting.	Invalid choice. Using easy difficulty. Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:

***Explanation:** In the 2nd part of test case 2, we can see that when an invalid difficulty level is being entered by the user, the program will select the easy level maze by itself, instead of getting terminated. And, it is happening due to the following piece of code in line no. 190:

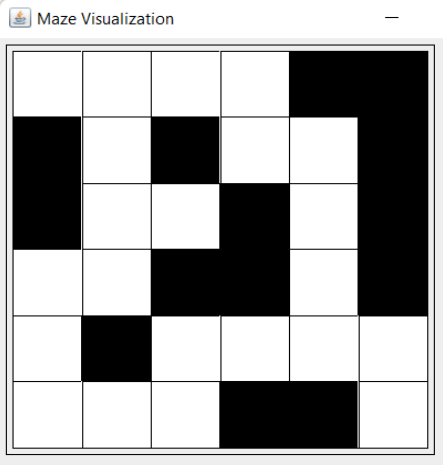
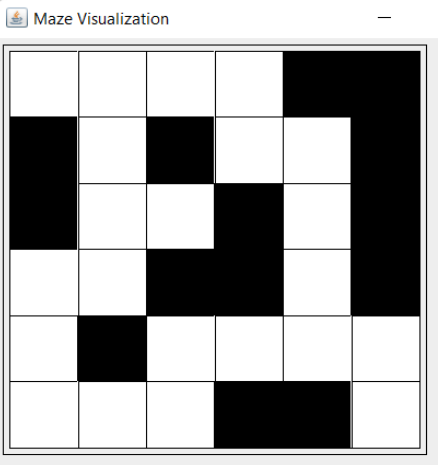
default:

```
System.out.println1("Invalid choice. Using easy difficulty.");
difficulty = "easy";
break;
```


Test Case 3: When user is prompted to enter which path(s) are wanted to be displayed (for medium level maze)		
Inputs	Expected Output	Actual Output
1) 1	All paths: RDDLDDRRURRRD RRRDRDDDDR	All paths: RDDLDDRRURRRD RRRDRDDDDR
2) 2	Best Path: RRRDRDDDDR	Best Path: RRRDRDDDDR
3) 3	All paths: RDDLDDRRURRRD RRRDRDDDDR Best Path: RRRDRDDDDR	All paths: RDDLDDRRURRRD RRRDRDDDDR Best Path: RRRDRDDDDR
4) 4, 5, or any other input (except 1, 2, 3)	Invalid option. Exiting.	Invalid option. Exiting. Do you want to display the maze graphically? (yes/no)

***Explanation:** In the 4th part of test case 3, we see that the program should have terminated on receiving an invalid input. But, even after displaying the message “Invalid option. Exiting.”, the program does not actually end and continue to ask the user if they want the graphical representation of the maze, as if a valid input is received. This is happening because just after line no. 232 in the program, a “return” statement should have been present, to immediately terminate the program on receiving an invalid input.

Test Case 4: When user is prompted to choose whether to have graphical representation of the maze displayed or not (for medium level maze)

Inputs	Expected Output	Actual Output
1) yes		
2) no	Thank you, hope you enjoyed your game!	Thank you, hope you enjoyed your game!
3) Any other string (except 'yes' and 'no')	Invalid option. Exiting.	Thank you, hope you enjoyed your game!

***Explanation:** In the 3rd part of test case 4, we can see that upon entering a wrong input, the output should have displayed a message telling the user than an invalid input is encountered. But instead of that, the program is thanking the user as if they entered 'no' in the input.

This is happening because in line no. 238 of the program, the condition for 'yes' as input has been mentioned properly. However, the condition for 'no' has not been tackled in the similar manner. After the 'yes' condition, a simple else statement has been used, due to which the program is displaying the message "Thank you, hope you enjoyed your game!" in both the situations (i.e., receiving 'no' as the input as well as receiving an invalid input).

(b) Integration Testing:

*In the integration testing, we will keep only one maze (of a random difficulty level) in our program and edit out the other two mazes. This means that we will link only one module (i.e., maze) of our program with the main method at a time. And thus, here we will be able to test that how the program works if only one module is kept and the other two are excluded.

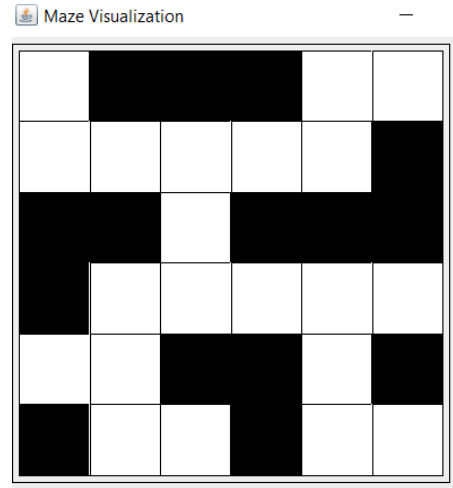
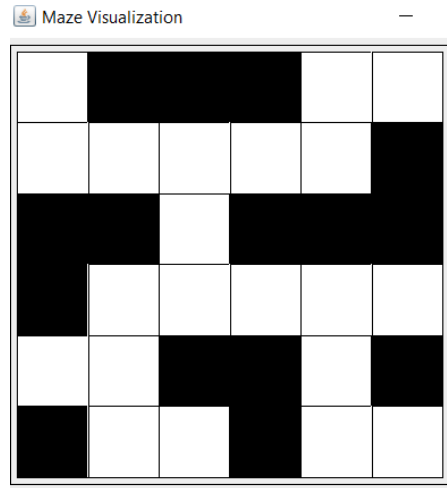
Test Case 1: Selecting the maze difficulty when linking the easy level maze with the main method		
Inputs	Expected Output	Actual Output
1) 1	Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:	Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:
2) 2 or 3	java.lang.ArrayIndexOut OfBoundsException	java.lang.ArrayIndexOut OfBoundsException
3) 4, 5, or any other input (except 1, 2, 3)	Invalid choice. Using easy difficulty. Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:	Invalid choice. Using easy difficulty. Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:

*The medium and advanced level of mazes would also behave similarly as the easy one, when integration testing would be implemented on them respectively.

(c) System Testing:

Test Case 1: Entering valid inputs and testing the whole program (using advanced level maze)		
Inputs	Expected Output	Actual Output
start	Choose the difficulty level: 1. Easy 2. Medium 3. Advanced Enter your choice:	Choose the difficulty level: 1. Easy 2. Medium 3. Advanced Enter your choice:
3	Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:	Options: 1. Display all paths 2. Display the best path 3. Display both Enter your choice:
3	All paths: DRRDDRRDDR Best Path: DRRDDRRDDR Do you want to display the maze graphically? (yes/no)	All paths: DRRDDRRDDR Best Path: DRRDDRRDDR Do you want to display the maze graphically? (yes/no)

yes



Experiment 8: To implement the given project as per the set of objectives.

```
class Maze_Solver_Improvised {  
    public static void solve(int i, int j, int a[][], int n, ArrayList<String> ans, String  
move, int vis[][]) {  
  
        if (i == n - 1 && j == n - 1) {  
            ans.add(move);  
            return;  
        }  
        // downward  
        if (i + 1 < n && vis[i + 1][j] == 0 && a[i + 1][j] == 1) {  
            vis[i][j] = 1;  
            solve(i + 1, j, a, n, ans, move + 'D', vis);  
            vis[i][j] = 0;  
        }  
        // left  
        if (j - 1 >= 0 && vis[i][j - 1] == 0 && a[i][j - 1] == 1) {  
            vis[i][j] = 1;  
            solve(i, j - 1, a, n, ans, move + 'L', vis);  
            vis[i][j] = 0;  
        }  
        // right  
        if (j + 1 < n && vis[i][j + 1] == 0 && a[i][j + 1] == 1) {  
            vis[i][j] = 1;  
            solve(i, j + 1, a, n, ans, move + 'R', vis);  
            vis[i][j] = 0;  
        }  
    }  
}
```

```

// upward
if (i - 1 >= 0 && vis[i - 1][j] == 0 && a[i - 1][j] == 1) {
    vis[i][j] = 1;
    solve(i - 1, j, a, n, ans, move + 'U', vis);
    vis[i][j] = 0;
}
}

```

```

public static ArrayList<String> findPath(int[][] m, int n) {
    int vis[][] = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            vis[i][j] = 0;
        }
    }
}

```

```

ArrayList<String> ans = new ArrayList<>();
if (m[0][0] == 1) {
    solve(0, 0, m, n, ans, "", vis);
}
return ans;
}

```

```

private static String findBestPath(ArrayList<String> paths) {
    if (paths.isEmpty()) {
        return null;
    }
}

```

```

String bestPath = paths.get(0);
for (String path : paths) {
    if (path.length() < bestPath.length()) {
        bestPath = path;
    }
}
return bestPath;
}

private static int[][] generateMatrix(String difficulty) {
    switch (difficulty.toLowerCase()) {
        case "easy":
            return new int[][]{
                {1, 1, 0, 1, 0, 1},
                {0, 1, 1, 1, 1, 1},
                {1, 0, 0, 1, 1, 1},
                {1, 1, 1, 0, 0, 1},
                {0, 1, 0, 1, 1, 1},
                {1, 1, 1, 1, 0, 1}
            };
        case "medium":
            return new int[][]{
                {1, 1, 1, 1, 0, 0},
                {0, 1, 0, 1, 1, 0},
                {0, 1, 1, 0, 1, 0},
                {1, 1, 0, 0, 1, 0},
                {1, 0, 1, 1, 1, 1},
                {1, 1, 1, 0, 0, 1}
            };
    }
}

```



```

case "advanced":
    return new int[][]{
        {1, 0, 0, 0, 1, 1},
        {1, 1, 1, 1, 1, 0},
        {0, 0, 1, 0, 0, 0},
        {0, 1, 1, 1, 1, 1},
        {1, 1, 0, 0, 1, 0},
        {0, 1, 1, 0, 1, 1}
    };
default:
    System.out.println("Invalid difficulty. Select a valid level for maze.");
    return new int[][]{};
}
}
}

```

These lines include the methods for solving the maze, finding paths, and generating the maze matrix based on difficulty.

Experiment 9: To prepare PERT Chart for given project.

