Quiz 2 - DAA (D)
Azhar Abiyu Rasendriya Harun
5025211177

https://github.com/pusendut/Quiz2-Design-Analysis-of-Algorithms-D

Maze Game with Depth-First Search (DFS) Implementation

This project is a simple maze game implemented in Java with an integrated Depth-First Search (DFS) algorithm to find a path through the maze.

Design

The design of the project revolves around the following components:

- Maze : Represents the maze structure. It encapsulates information about the maze layout, the starting point, and the exit point. The maze layout is represented as a 2D array of integers, where `0` represents an open path and `1` represents a wall.
- Player : Represents the player within the maze. It keeps track of the player's current position.
- Game : Manages the game logic, including the DFS algorithm to find a path from the starting point to the exit. It also handles user input and displays the maze with the player's position.
- Main : Acts as the entry point of the application. It initializes the maze, player, and game objects and starts the game loop.

Implementation

The implementation of the project follows these main steps:

1. Maze Generation : The maze layout is predefined as a 2D array in the `Main` class. Alternatively, you can implement maze generation algorithms such as recursive backtracking or Prim's algorithm to generate random mazes.

2. Depth-First Search (DFS) : The DFS algorithm is implemented in the `Game` class to find a path from the starting point to the exit in the maze. It explores the maze recursively, moving through each available path until it reaches the exit or explores all possible paths.

3. Game Logic : The `Game` class manages the game logic, including handling user input for player movement and updating the display to show the maze with the player's position.

Analysis/Evaluation

Performance:

- The DFS algorithm provides a simple and efficient way to find a path through the maze. However, its performance may degrade on larger mazes or mazes with complex layouts.
- For larger mazes or more complex scenarios, alternative pathfinding algorithms such as A* (A-star) or Dijkstra's algorithm may offer better performance.

Scalability:

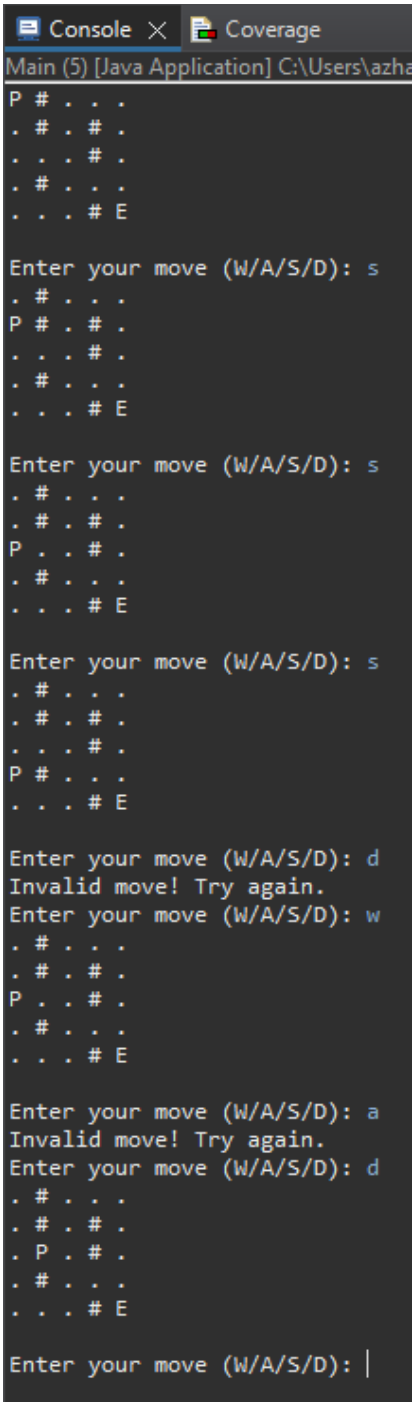
- The project is designed to handle small to medium-sized mazes.
- To support larger mazes or additional features, such as multiple players or dynamic maze generation, the codebase may require refactoring and optimization.

Conclusion

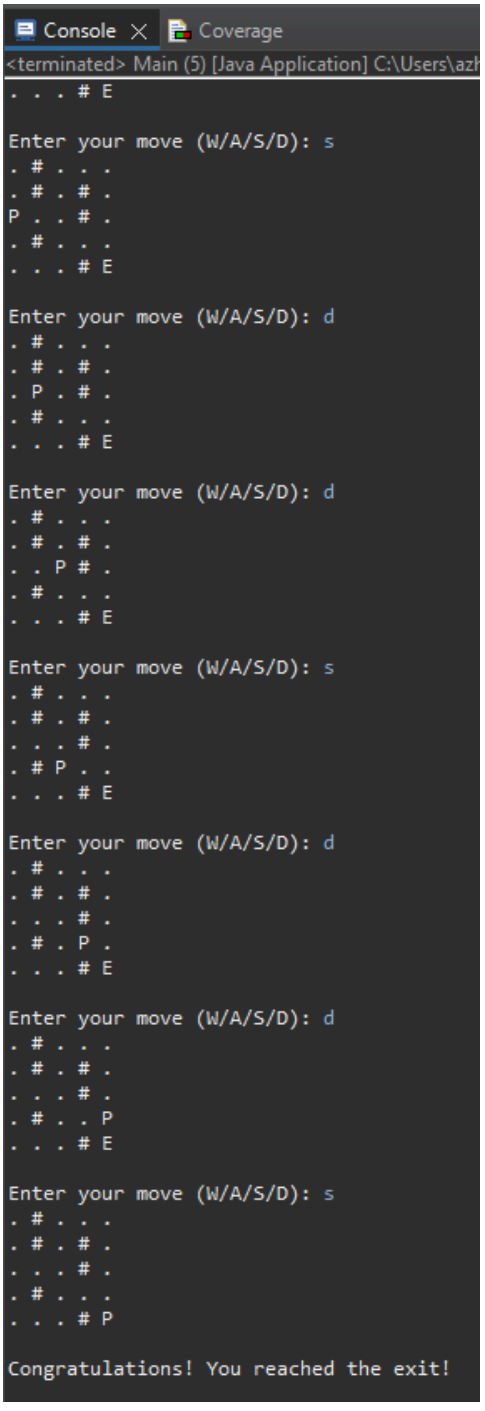
The maze game project demonstrates the implementation of a simple game with integrated pathfinding using the DFS algorithm. While the project provides a basic foundation for maze navigation, there is room for further enhancements and optimizations to improve performance, scalability, and user experience. Future iterations could include features such as interactive gameplay, support for larger mazes, additional pathfinding algorithms, and customizable maze generation.

Output

Fail / Player hitting wall



Success / Player found the exit



Source Code

Maze.java Class

```
package mazegame;
public class Maze {
   private final int[][] maze;
   private final int rows;
   private final int cols;
   private final int startRow;
   private final int startCol;
   private final int exitRow;
   private final int exitCol;
   public Maze(int[][] maze, int startRow, int startCol, int exitRow, int exitCol)
{
       this.maze = maze;
       this.rows = maze.length;
       this.cols = maze[0].length;
       this.startRow = startRow;
       this.startCol = startCol;
       this.exitRow = exitRow;
       this.exitCol = exitCol;
   }
   public int getRows() {
       return rows;
   }
   public int getCols() {
       return cols;
   }
   public int getStartRow() {
       return startRow;
   }
   public int getStartCol() {
       return startCol;
   }
   public int getExitRow() {
       return exitRow;
   }
   public int getExitCol() {
       return exitCol;
   }
   public boolean isValidMove(int row, int col) {
        return row >= 0 && row < rows && col >= 0 && col < cols && maze[row][col]
== 0;
   }
}
```

Player.java Class

```java
package mazegame;
public class Player {
    private int row;
    private int col;
    public Player(int startRow, int startCol) {
        this.row = startRow;
        this.col = startCol;
    }
    public int getRow() {
        return row;
    }
    public void setRow(int row) {
        this.row = row;
    }
    public int getCol() {
        return col;
    }
    public void setCol(int col) {
        this.col = col;
    }
    public void move(int newRow, int newCol) {
        this.row = newRow;
        this.col = newCol;
    }
}
```

Main.java Class

```java
package mazegame;
public class Main {
    public static void main(String[] args) {
        int[][] mazeLayout = {
            {0, 1, 0, 0, 0},
            {0, 1, 0, 1, 0},
            {0, 0, 0, 1, 0},
            {0, 1, 0, 0, 0},
            {0, 0, 0, 1, 0}
        };
        Maze maze = new Maze(mazeLayout, 0, 0, 4, 4);
        Game game = new Game(maze);
        game.start();
    }
}
```

Game.java Class

```
package mazegame;
import java.util.Scanner;
public class Game {
   private Maze maze;
   private Player player;
   private enum Direction {
       UP, DOWN, LEFT, RIGHT
   }
   public Game(Maze maze) {
       this.maze = maze;
       this.player = new Player(maze.getStartRow(), maze.getStartCol());
   }
   public void start() {
       Scanner scanner = new Scanner(System.in);
       printMaze();
       while (true) {
           System.out.print("Enter your move (W/A/S/D): ");
           String move = scanner.nextLine().toUpperCase();
           if (move.equals("W")) {
               movePlayer(Direction.UP);
           } else if (move.equals("S")) {
               movePlayer(Direction.DOWN);
           } else if (move.equals("A")) {
               movePlayer(Direction.LEFT);
           } else if (move.equals("D")) {
               movePlayer(Direction.RIGHT);
           } else {
               System.out.println("Invalid move! Use W/A/S/D.");
           }
                 if (player.getRow() == maze.getExitRow() && player.getCol() ==
maze.getExitCol()) {
               System.out.println("Congratulations! You reached the exit!");
               break;
           }
       }
       scanner.close();
   }
   private void movePlayer(Direction direction) {
       int newRow = player.getRow();
       int newCol = player.getCol();
       switch (direction) {
           case UP:
               newRow--;
               break;
           case DOWN:
               newRow++;
               break;
           case LEFT:
               newCol--;
               break;
```

```
            case RIGHT:
                newCol++;
                break;
        }
        if (maze.isValidMove(newRow, newCol)) {
            player.move(newRow, newCol);
            printMaze();
        } else {
            System.out.println("Invalid move! Try again.");
        }
    }
    private void printMaze() {
        for (int i = 0; i < maze.getRows(); i++) {
            for (int j = 0; j < maze.getCols(); j++) {
                if (i == player.getRow() && j == player.getCol()) {
                    System.out.print("P ");
                } else if (i == maze.getExitRow() && j == maze.getExitCol()) {
                    System.out.print("E ");
                } else if (maze.isValidMove(i, j)) {
                    System.out.print(". ");
                } else {
                    System.out.print("# ");
                }
            }
            System.out.println();
        }
        System.out.println();
    }
}
```