Maze Solver Using Backtracking

Project Submitted in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Technology in the field of Computer Science and Engineering

BY

SHADAAN ALI (23CS2021068)
RAZA KHAN (23CS2021051)
RAHUL KUMAR THAKUR (23CS2021048)
SAKIL AHAMED (23CS2021057)
SAMIR KUMAR (23CS2021059)

School of Engineering

JIS University

81, Nilgunj Rd, Jagrata Pally, Deshpriya nagar, Agarpara

Kolkata, West Bengal 700109.

CERTIFICATE

This is to clarify that Shadaan Ali (23CS2021068),

Raza Khan (23CS2021051),

Rahul Kumar Thakur (23CS2021048),

Sakil Ahamed (23CS2021057), and

Samir Kumar (23CS2021059) have completed their project entitled

'Quantrive: A Website for Exploring Quantum Computing', under the guidance of Debmitra Ghosh, in partial fulfillment of the requirements for the award of the Bachelor of Technology in the department of School of Engineering of JIS University, Agarpara.

This work is an authentic record of their own work carried out during the academic year 2023–24 and to the best of our knowledge, this work has not been submitted elsewhere as part of the process of obtaining a degree, diploma, fellowship, or any other similar title.

Date: 21/05/24 Place: JIS University, Kolkata

_____

_____

Signature of HOD

Signature of Supervisor

Table of Contents

Abstract

The "Maze Solver with Backtracking using Java Swing" project is a graphical application that demonstrates how recursive backtracking algorithms can be applied to solve maze problems. Developed using Java, this application combines object-oriented programming principles with a visually interactive interface built using the Swing framework. The maze is represented as a two-dimensional array, where walls and paths are defined numerically. The solver starts from the top-left corner and attempts to find a path to the bottom-right corner, dynamically visualizing the search process.

The core algorithm explores all possible directions—right, down, left, and up—while avoiding walls and previously visited cells. If a path leads to a dead end, the algorithm backtracks and tries a different route, continuing this process until the destination is reached or all options are exhausted. The graphical interface enhances understanding by color-coding the maze grid: walls, paths, the start and end points, and the correct path once found.

This project serves as an educational tool to understand recursion, backtracking, and GUI development in Java. It lays the foundation for more advanced pathfinding strategies and demonstrates how logic and visual elements can work together to create meaningful simulations.

Introduction

Solving mazes is one of the most fundamental problems in computer science. It introduces students to the concept of recursion, search algorithms, and backtracking strategies. This project simulates a real-time maze solver using the recursive backtracking method and visualizes the algorithm's behavior using Java Swing components. The combination of logic processing and GUI development ensures a complete application that can be used to illustrate concepts in algorithm design and graphical programming. Users can interact with the GUI to trigger the maze-solving algorithm and receive immediate visual feedback on the path discovered by the program.

Objective

The primary objectives of this project are:

- To implement a maze-solving algorithm using recursion and backtracking.
- To design a graphical user interface using Java Swing.
- To visually demonstrate the working of the algorithm.
- To provide a modular and easily understandable codebase.
- To reinforce understanding of arrays, recursion, and event-driven programming.
- To prepare students for more complex pathfinding problems such as those involving A*, DFS, or BFS algorithms.

Technologies Used

This section lists the technologies and tools utilized in the development of this project:

- **Programming Language**: Java (Object-Oriented Programming)
- **GUI Toolkit**: Java Swing (for building interactive graphical user interfaces)
- **Integrated Development Environment (IDE)**: IntelliJ IDEA / Eclipse / NetBeans (based on preference)
- **Operating System**: Platform-independent (Tested on Windows 10 and Linux Ubuntu)

Java Swing provides an efficient and flexible toolkit for creating interactive GUIs that integrate seamlessly with Java logic. The choice of Java ensures platform independence and robustness.

System Design

The overall design of the system is modular, consisting of multiple classes and methods that handle different responsibilities:

- **MazeSolverGUI Class**: This is the main class extending `JFrame` which serves as the application window.
- **MazePanel Inner Class**: Extends `JPanel`, overrides `paintComponent()` to draw the maze and solution path.
- **solveMaze Method**: Core recursive backtracking function to solve the maze.
- **Action Listener**: Triggered when the user clicks the "Solve Maze" button.

This structure ensures separation of concerns and easier debugging and extension of the code.

Maze Representation

The maze is represented using a 2D integer array where:

- `0` indicates an open cell that can be traversed.
- `1` indicates a wall that blocks movement.

```
int[][] maze = {
    {0, 1, 0, 0, ...},
    ...
};
```

Additionally, a 2D boolean array `solution` tracks the path currently being explored and eventually displays the final solution path if one exists. This abstraction allows easy manipulation and visualization of maze logic.

Backtracking Algorithm

The backtracking algorithm is a classic technique in which we recursively explore all paths in a maze. The steps are:

1. Check if the current cell is within bounds.
2. Check if the cell is a wall or already visited.
3. Mark the cell as part of the path.
4. If the destination is reached, return true.
5. Otherwise, recursively try moving right, down, left, and up.
6. If all directions fail, backtrack by unmarking the current cell.

The beauty of backtracking lies in its simplicity and power, allowing us to systematically explore potential solutions and discard incorrect paths efficiently.

Java Swing GUI

The GUI is designed using Java Swing components and provides an interactive visualization of the maze and its solution. Main components include:

- **JFrame**: Acts as the main window.
- **JPanel (MazePanel)**: Renders the maze grid using colored rectangles.
- **JButton**: Triggers the solving process.
- **JOptionPane**: Displays dialogs for success or failure.

Color scheme:

- **Black**: Wall
- **White**: Open path
- **Cyan**: Part of the solution path
- **Green**: Start point (0,0)
- **Red**: Destination (bottom-right corner)

This color-based system makes it easy for users to understand the maze structure and how the algorithm proceeds.

Code Explanation - Main Class

```
public class MazeSolverGUI extends JFrame {
    // Constants for maze size and cell dimensions
    // Maze and solution arrays declared here
    // GUI components like MazePanel and JButton initialized
    // JFrame setup in constructor
}
```

The `MazeSolverGUI` class is the entry point of the program. It initializes the GUI, prepares the layout, and attaches listeners to buttons. It encapsulates the logic for managing the overall application flow.

Code Explanation - Maze Logic

The recursive `solveMaze` method is the brain of the maze solver:

```
private boolean solveMaze(int row, int col) {
    if (invalid) return false;
    mark cell as visited;
    if (goal reached) return true;
    recursively explore all directions;
    if all fail, backtrack and unmark;
    return false;
}
```

This logic ensures we visit only valid paths and backtrack when a dead-end is encountered, which is essential in ensuring completeness of search.

Code Explanation - GUI Design

The `MazePanel` class extends `JPanel` and overrides the `paintComponent(Graphics g)` method. It draws the maze and the solution path as per the data in the arrays

```
protected void paintComponent(Graphics g) {
    for each cell in maze {
        draw color-coded cell;
    }
    highlight start and end cells;
}
```

This method is automatically called whenever the GUI needs to be redrawn, ensuring the visual display is always current.

Testing & Output

Test Procedure:

- Launch application
- View initial maze
- Click "Solve Maze" button
- View result: solution path or error dialog

Expected Output:

- If a solution exists: cyan path is displayed
- If no solution: message dialog shown

Screenshots (to be added):

- Initial State
- Solved Maze

Edge Cases:

- Start or End blocked
- Entire maze blocked

These cases have been handled to ensure robustness and graceful handling of errors.

Conclusion & References

Conclusion: This project illustrates how algorithmic thinking and GUI programming can come together to create educational tools. The maze solver showcases recursion, data structure handling, and real-time visualization. Through the graphical output, users can intuitively grasp how backtracking works. The modular design allows for future improvements and serves as a foundation for more complex algorithms like BFS, DFS, or A*.

Future Scope:

- Maze generator tool
- User-drawn maze with mouse
- Animation of each move
- Speed controls
- 3D maze support

References:

1. Oracle Java Documentation - https://docs.oracle.com/javase/8/docs/api/
2. GeeksforGeeks - Maze Solving using Backtracking
3. Java2s - Swing and Graphics Tutorials
4. Stack Overflow - Discussions on Java Swing Best Practices