Report of CSE246 Project

Maze Path Finding

Submitted By:

Aklhak Hossain
2022-3-60-057
https://ahjim.com

Saba Tasnim Khan
2022-3-60-049

MD. Muntasir Ahmed Rifat
2022-1-60-333

Submitted To:

Tanni Mittra
Senior Lecturer
Department of CSE
East West University

EAST WEST UNIVERSITY
excellence in education

Jim

Table of Contents

Introduction

A Maze Path Finder is an algorithm used to navigate through a maze by finding a valid path from a starting point to a destination point. The maze is represented as a grid of cells, where each cell is either a free space (typically represented by 1) or a wall (typically represented by 0). The objective of the pathfinder is to determine whether there exists a path from the start (0, 0) to the destination (n-1, n-1) without crossing through walls, and if such a path exists, to find it.

Maze pathfinding algorithms are useful in several applications, including:

1. Robotics and Autonomous Vehicles: Autonomous robots use pathfinding algorithms to navigate around obstacles and find optimal paths to reach a target destination.
2. Artificial Intelligence (AI): AI agents (NPCs in video games) use pathfinding algorithms to navigate complex environments and avoid obstacles.
3. Puzzle Solving: Algorithms are applied in puzzles like Sudoku, labyrinths, and other maze-based games.
4. Network Routing: Pathfinding is used in networking to find the shortest or optimal route for data transmission between nodes in a network.

The backtracking algorithm, which is used in the provided project to solve the maze, was developed in the 20th century by Donald Knuth. Backtracking, a form of depth-first search (DFS), was formalized in Knuth's works in the 1960s and 1970s. In this context, backtracking refers to trying different paths to find a solution and undoing or "backtracking" when a dead-end is reached.

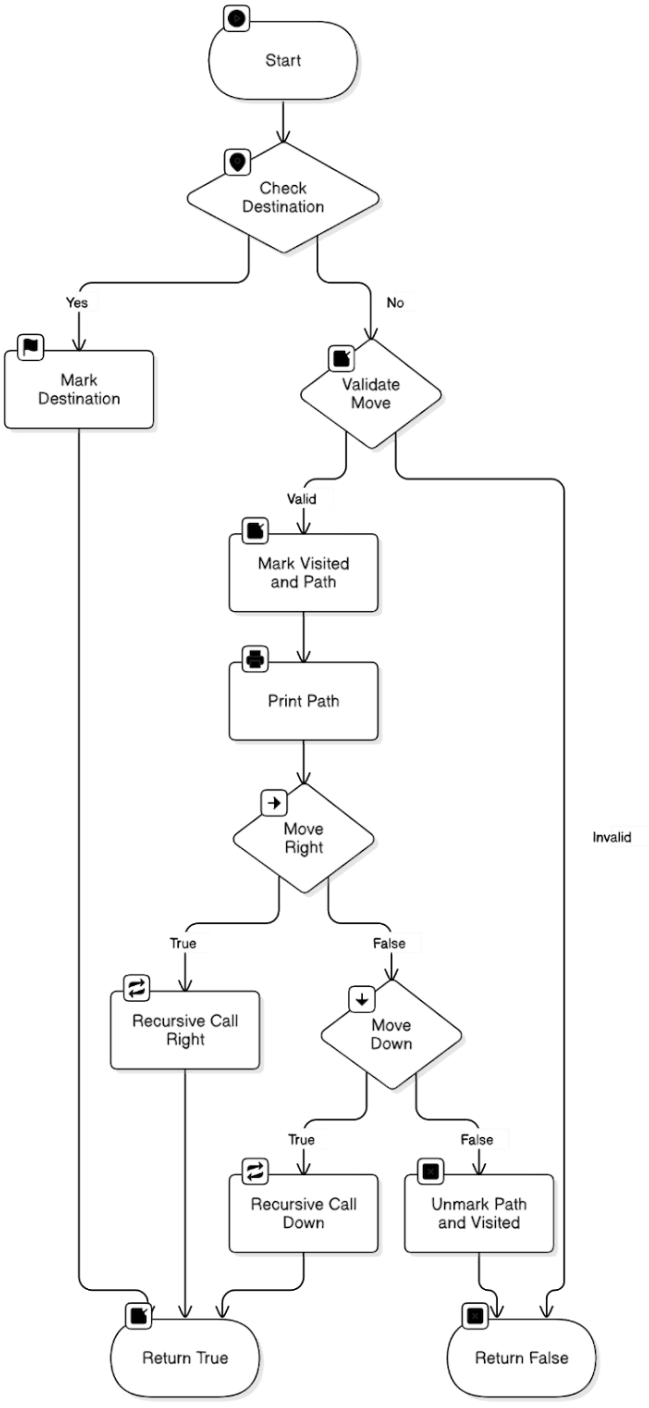

Figure: Find Path in Maze Flowchart

Types of Data Patterns for Input

The input to the maze solver program is provided as a matrix or grid with (n x n) size. Each element in the matrix represents a cell in the maze, and each cell can either be a 1 (representing a valid path) or 0 (representing a wall or obstacle).

Input Types:

1. Square Grid Maze (n x n): The maze is represented as a square matrix with n rows and n columns. The user provides the grid, where 1 is an open path and 0 is a wall.
2. Valid Input Format: The input for the maze can either be:
    - A fully open maze with some blocked paths.
    - A completely blocked maze (no solution).
    - A maze with a single path from start to end.
    - A complex maze with multiple possible paths.

Example Input (4x4 Maze):

1 0 0 0
1 1 0 1
0 1 0 0
1 1 1 1

Where,

- 0 represents a blocked path (wall).
- 1 represents a valid path (open space).

Program Steps

1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
----
1 0 0 0
1 0 0 0
0 0 0 0
0 0 0 0
----
1 0 0 0
1 1 0 0
0 0 0 0
0 0 0 0
----
1 0 0 0
1 1 0 0
0 1 0 0
0 0 0 0
----
1 0 0 0
1 1 0 0
0 1 0 0
0 1 0 0
----
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 0

Solution Path:
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1

Code

```
bool findPath(int x, int y, int ** maze, int ** visited, int ** path,
int n) {
  if (x == n - 1 && y == n - 1) {
    path[x][y] = 1; // Mark the destination in the path
    return true;
  }

  if (isValidMove(x, y, maze, visited, n)) {
    visited[x][y] = 1;
    path[x][y] = 1;

    printMatrix(path, n);

    if (findPath(x, y + 1, maze, visited, path, n)) return true;

    if (findPath(x + 1, y, maze, visited, path, n)) return true;

    path[x][y] = 0;
    visited[x][y] = 0;
  }

  return false;
}
```

Real-Life Applications

- Autonomous Vehicles: Autonomous cars use pathfinding algorithms to navigate through roads, avoiding obstacles, and finding the optimal route from start to destination.
- Robotics: Robots use maze-solving algorithms to explore new environments, identify obstacles, and plan movement accordingly.
- Video Games: Many games, especially puzzle or adventure games, use pathfinding algorithms for AI-controlled characters to move through maze-like environments.
- Network Routing: Algorithms like A* and Dijkstra's algorithm, which are based on the same principles, are used in networking to determine the shortest or most efficient path for data packets.

Time Complexity

The time complexity of the recursive backtracking algorithm used here is:

- Time Complexity: $O(n^2)$, where n is the size of the maze. In the worst case, the algorithm may explore every cell once before reaching the destination or determining that no solution exists.
- Space Complexity: $O(n^2)$, primarily due to the storage of the maze, visited, and path matrices, each of which requires space for (n x n) elements.

Relevancy of Maze Path Finding Algorithm

Maze Path Finding algorithm is one kind of Backtracking algorithm. Backtracking is an algorithm where we make choices, but if a dead-end is reached, we backtrack to the previous decision point and try another path.

1. Simplicity: One of the main reasons we have chosen backtracking is because it is easy to understand and implement. It is a recursive algorithm that explores all possible paths, marking valid ones and backtracking when necessary. This is ideal for a maze problem where the solution may require testing different paths.
2. Guaranteed Solution: Backtracking guarantees that if a path exists, we will find it. If there is no path, the algorithm will exhaust all possibilities and return that there is no solution. This makes backtracking reliable for solving mazes of any complexity.
3. Memory Efficiency for Small to Medium Mazes: Backtracking is memory efficient because we do not need to store all possible paths. The recursion stack keeps track of the current path, and when we backtrack, we discard that path and explore new ones.
4. Clear Termination: Backtracking works by reaching the destination or realizing the current path leads nowhere, at which point it will backtrack and try other paths. This results in clear and predictable termination of the algorithm, either finding the path or proving that no path exists.

There are other well-known algorithms for solving maze pathfinding problems, such as BFS (Breadth-First Search), DFS (Depth-First Search), but each has its own strengths and weaknesses, and for this specific problem, backtracking is a solid choice.

1. Breadth-First Search (BFS):

    - BFS guarantees finding the shortest path, but it requires storing all the nodes at the current level in a queue. This can lead to higher memory usage for large mazes. In contrast, backtracking only stores the current path, making it more memory efficient.
    - BFS can be more optimal when we need the shortest path, but it is also more resource intensive in terms of memory.

2. Depth-First Search (DFS):

    - DFS is similar to backtracking in that it explores paths deeply before backtracking, but DFS does not guarantee that the shortest path is found. Additionally, DFS can get stuck in infinite loops if cycles exist, and handling that requires additional complexity.

- In contrast, backtracking naturally avoids cycles without needing additional logic since it explicitly marks visited cells.

In this project, we used the backtracking algorithm to solve the maze pathfinding problem. This approach was chosen due to its simplicity, reliability, and memory efficiency, particularly for small to medium-sized mazes. While other algorithms like BFS, DFS could also solve the problem, backtracking proved to be the best fit for the problem due to its minimal memory requirements and clear structure. Although backtracking may not be the most optimal for large mazes or for finding the shortest path, it serves as a great educational tool and an effective solution for many pathfinding problems. This project demonstrates how simple approach like backtracking can be highly effective in solving real world problems such as maze pathfinding.

Future Scope

- Optimized Algorithms: We can improve the solution using other algorithms like Breadth-First Search (BFS) for unweighted mazes or A algorithm* for weighted mazes, which will help find the shortest path in a more efficient manner.
- Multi-Path Solutions: Extend the program to find all possible paths between the start and the destination.
- 3D Maze Navigation: The algorithm can be adapted for more complex 3D mazes, where navigation occurs in three dimensions.

Conclusion

The maze pathfinder algorithm demonstrates the utility of backtracking in solving pathfinding problems. It is an essential algorithm for applications like robotics, AI, and puzzle-solving. While the current implementation works well for small-to-medium-sized mazes, further optimizations and extensions can be explored to enhance performance, scalability, and the ability to handle more complex scenarios.