Effat University

SPRING 2025

COLLEGE OF ENGINEERING – DEPARTMENT OF COMPUTER SCIENCE

Maze Solver

Course : Data Structures
CS2011

By:
Qusai Rahaim
&
Moataz Al Ahmadi

Instructor: Dr. Naila

Introduction :

The Project Uses two algorithms in C++ to make a maze solver .
Algorithms:
-Depth-First Search (DFS) using Stack.
- Breadth-First Search (BFS) using Queue

The maze is in 2D , the solver has to find a path from start to end.

Architecture:

1. Maze in 2D
2. printMaze() Displays layout
3. Runs DFS & BFS
4. Runs Stack & Queue
5. Prints each move
6. Output path

Data Structure :

- 2D Array to represent Maze
- Stack - DFS to track cells to explore
- Queue - BFS to find the shortest path
- Struct hold the coordinates
- Boolean

Snippet for maze:

```
char maze[ROWS][COLS] = {
   {'S', '0', '1', '1', '1'},
   {'1', '0', '0', '0', '1'},
   {'1', '1', '1', '0', '0'},
   {'1', '1', '1', '1', 'E'}
};
```

```
void printMaze() {
   cout << "Maze : " << endl;
   for (int i = 0; i < ROWS; i++) {
      for (int j = 0; j < COLS; j++) {
         cout << maze[i][j] << " ";
      }
      cout << endl;
   }
   cout << endl;
}
```

Snippet for DFS:

```
bool dfsMazeSolver(int startX, int startY) {
   stack<Position> s;
   s.push({startX, startY});
   visitedDFS[startX][startY] = true;

   cout << "Using Stack:\n" << endl;

   while (!s.empty()) {
      Position current = s.top();
      s.pop();

      cout << "destination (" << current.x << ", " << current.y << ")" << endl;

      // path
```

```
        if (maze[current.x][current.y] == '0') {
            maze[current.x][current.y] = '*';
        }

        if (maze[current.x][current.y] == 'E') {
            cout << "DFS completed" << endl << endl;
            return true;
        }

        for (int i = 0; i < 4; i++) {
            int newX = current.x + dx[i];
            int newY = current.y + dy[i];

            if (isValidDFS(newX, newY)) {
                s.push({newX, newY});
                visitedDFS[newX][newY] = true;
            }
        }
    }

    cout << "DFS no path" << endl << endl;
    return false;
}
```

Screenshots of output:

```
Maze :
S 0 1 1 1
1 0 0 0 1
1 1 1 0 0
1 1 1 1 E

Using Stack:

destination (0, 0)
destination (0, 1)
destination (1, 1)
destination (1, 2)
destination (1, 3)
```

```
   destination (2, 3)
   destination (2, 4)
   destination (3, 4)
   DFS completed

   Using Queue:

   destination (0, 0)
   BFS no path
Path Marked '*' :

Maze :
S * 1 1 1
1 * * * 1
1 1 1 * *
1 1 1 1 E

PS C:\Users\qusai\OneDrive\Desktop\C++ Project>
```

Conclusion:

This project demonstrates the application of two key data structures — Stack and Queue — in solving mazes through DFS and BFS.