**COMSATS University Islamabad, Lahore Campus**

**Block–B, Department of Electrical and Computer Engineering** **COMSATS Institute of Information Technology, 1.5KM Defence Road, Off Raiwind Road, Lahore**

**Lab Project**

|  |  |  |
| --- | --- | --- |
| **1** | **Course Code and Title** | CSC 211 |
| **2** | **Credit Hours** | 3+1 |
| **3** | **Assessment Type** | Lab Project |
| **5** | **Semester** | 3rd |
| **6** | **Resource Person** | Dr Hamid |

Program Specifications

|  |  |
| --- | --- |
| **Application/ Program name:** | Maze Solver using BFS Search |
| **Written by:** | Muhammad Zawahir Amin 075  Uzair Sahi 098  Abdullah Razzaq 006 |

|  |
| --- |
| **Purpose or problem definition:** |
| To find the **shortest path** between two points in a maze using the **Breadth-First Search (BFS)** algorithm. It solves the problem of navigating from a starting position to a destination while avoiding walls and marking the path taken. |

|  |
| --- |
| **Program Procedures:** |
| **Maze Setup:**   * Define a 2D grid (maze) using vector<vector<char>> to represent walls (#) and open spaces (' ').   **Pathfinding with BFS:**   * Use BFS to explore the maze starting from the given start position. * Enqueue valid neighboring cells and track visited cells to avoid exploration.   **Mark the Shortest Path:**   * If the end position is reached, mark the shortest path in the maze with 'X'. **Display Output:** * Print the updated maze showing the path or indicate that no path exists.   A two-dimensional grid (maze) is defined using a vector<vector<char>>. The grid represents walls using # and open paths using ' '. This abstraction ensures modularity, allowing for flexible maze dimensions and configurations.  **Steps to Setup:**   * Initialize a grid with user-defined dimensions. * Populate the grid with walls and open spaces. * Specify the start and destination points within the grid. |

|  |
| --- |
| **Algorithm/Processing/Conditions:** |
| **Inputs:** |
| Maze grid (vector<vector<char>>) with walls (#) and open spaces (' '). Starting position (start) and destination (end). |
| **Processes:** |
| Initialize a queue with the starting position and an empty path.  Use a set to track visited positions.  While the queue is not empty:   * Dequeue the current position and path. * If the current position equals the destination: |

Page **1** of **2**

|  |
| --- |
| o Store the path.  • Otherwise, explore all four possible directions (up, down, left, right): o Check if the new position is within bounds, unvisited, and not a wall (' '). o If valid, mark it as visited and enqueue with the updated path. |
| **Outputs:** |
| Print the maze with the shortest path marked ('X') or indicate failure to find a path. |

Page **2** of **2**