# Maze Puzzle Game – Report

## Introduction

The Maze Puzzle Game is a simple yet engaging game that challenges players to navigate through a labyrinth from a start point to an exit using keyboard controls. Players can move the blue player icon 🔵 using either the arrow keys or WASD keys. The goal is to reach the green exit icon 🟢 in the fewest moves possible.

The game is built with a clear maze structure and provides an automated solution using Prolog, demonstrating the use of algorithmic problem-solving techniques in artificial intelligence.

## Game Rules

1. Objective: Navigate from the starting point to the exit.
2. Player Representation: Blue circle 🔵.
3. Exit Representation: Green circle 🟢.
4. Movement: Use arrow keys or WASD keys.
5. Counting Moves: Each movement increments the move counter.
6. Maze Completion: The game announces a win once the player reaches the exit, showing the number of moves taken.

## Maze Structure

- The maze is represented as a 10×10 grid.
- Walls block movement; paths allow movement.
- Start Position: (1,1)
- Exit Position: (8,8)
- The maze includes internal walls that create challenges and force players to find the correct path.

## Prolog Maze Solver Algorithm

The game includes a Prolog-based maze solver that demonstrates algorithmic solutions for finding paths. The solver uses Depth-First Search (DFS) and Breadth-First Search (BFS).

### 1. Maze Representation

- wall(X, Y) – Defines a wall at position (X, Y).
- valid_position(X, Y) – Checks whether the position is within the maze bounds and not a wall.
- start_position(X, Y) – Player starting point.
- exit_position(X, Y) – Exit point.

### 2. Movement

Possible moves are defined as:
- Up: (X-1, Y)
- Down: (X+1, Y)
- Left: (X, Y-1)
- Right: (X, Y+1)

Each move is validated to ensure it is within bounds and not blocked by a wall.

### 3. Depth-First Search (DFS)

DFS explores paths by moving forward until it reaches a dead end, then backtracks:
dfs((X, Y), Exit, Visited, Path)

- Visited keeps track of already visited positions.
- The algorithm recursively searches for a path from the start to the exit.

### 4. Breadth-First Search (BFS)

BFS finds the shortest path by exploring all possible moves level by level:
bfs([[Position|Path]|_], Position, [Position|Path])

- Uses a queue (CurrentPath) to explore all possibilities.
- Stops when the exit is reached, ensuring the shortest path is found.

### 5. Additional Features

- Path Printing: The solver prints the path in (X, Y) coordinates.
- Path Length: Calculates the total number of steps to the exit.
- Dead End Detection: Finds all positions with only one possible move.
- Move Count: Counts possible moves from a given position.

## Conclusion

The Maze Puzzle Game combines interactive gameplay with algorithmic problem-solving. It allows players to:
- Experience maze navigation challenges.
- Understand how DFS and BFS algorithms work.
- Analyze maze complexity and dead ends.

By integrating Prolog for the automated solver, this game provides a practical example of how artificial intelligence can be used to solve real-world navigation problems efficiently.