RWA4 Report: MicroMouse Robot Maze Navigation System

Team Members:
Shanthosh Raaj Mohanram Mageswari
Chris Collins
Jonathan Leonard Crespo
Lucas Janniche

1 Introduction

This report describes the implementation of Depth-First Search (DFS) for maze exploration in the Micromouse simulator. The robot starts at cell $(0, 0)$ facing north and must reach one of the four center goal cells. The assignment requires the robot to:

- Build a **discovered map** of the maze using live sensor readings.
- Plan a path using DFS based only on the discovered map.
- Physically move one cell at a time.
- If a step is blocked by a newly discovered wall, **stop, update the map, and replan using DFS**.
- Continue until the goal is reached.

All interactions with the simulator occur only through the `MazeControlAPI`. The robot never reads directly any map; it must discover the environment incrementally.

This document also includes an optional Breadth-First Search (BFS) implementation, as allowed by the bonus section.

2 Robot Motion and Sensing Model

The robot has access to three wall sensors:

$$\text{frontSensor}, \quad \text{leftSensor}, \quad \text{rightSensor}.$$

These are queried through:

- `has_wall_front()`
- `has_wall_left()`
- `has_wall_right()`

The robot moves using the following:

- `move_forward(1)`
- `turn_left()`
- `turn_right()`

In every cell, after orienting toward the next step in the planned path, the robot senses the walls and updates the internal map using:

`set_wall(x, y, direction).`

No planning algorithm is allowed to use unknown cells; DFS and BFS may only use walls that have been discovered so far.

3 DFS Path Planning on the Discovered Map

DFS explores cells in priority order:

$$\text{North} \rightarrow \text{East} \rightarrow \text{South} \rightarrow \text{West}.$$

A node in the maze is defined as:

$$N = (x, y, d, x_0, y_0)$$

where $d \in \{n, e, s, w\}$ is the direction of motion used to arrive at the node and x_0, y_0 are the parent node coordinates.

A move from (x, y) to (x', y') in the direction d is legal only if:

$$\neg\text{has\_known\_wall}(x, y, d).$$

DFS uses a stack and marks visited states to prevent revisiting discovered cells.

When the goal is reached, the path is reconstructed by storing parent pointers.

4 Robot Execution and Re-planning

After DFS returns an initial planned path, the robot executes it one step at a time.

At each step:

1. Determine the direction of movement required.
2. Rotate in that direction.
3. Read wall sensors at front, left and right.
4. Update the internal map with any newly detected walls.
5. If any new newly detected wall blocks any part of path to goal ⇒ re plan using DFS.
6. If no part of current path or re-planned path is blocked ⇒ continue along path.

This ensures that the robot never blindly follows an outdated plan.

5 BFS Implementation (Bonus)

A BFS implementation was added as a subclass of the `Algorithm` base class.

BFS differs from DFS in that it explores levels by level using a queue. This guaranties the **shortest path** on a grid if all edges have equal weight.

The BFS algorithm used here:

- Follows the same movement constraints as DFS.
- Uses only walls discovered by live sensing.
- Reconstructs the path using parent pointers.

Neither of the algorithms are used to actually move the robot (per instructions) but can be run for comparison and visualization.

6 DFS vs. BFS Comparison

Completeness

- **DFS**: Complete on finite grids if cycles are avoided.
- **BFS**: Always complete.

Optimality

- **DFS**: Not optimal; may find long detours.
- **BFS**: Optimal on unweighted grids; guaranties shortest path.

Memory Usage

- **DFS**: Requires storing only a stack and visited set.
- **BFS**: Requires storing a queue containing the full frontier; memory grows much faster.

Suitability for This Assignment

DFS is appropriate because:

- Re-planning is often expected.
- Guaranteed to find a path under the constraints of the assignment.
- The map is only partially discovered.
- Optimality is not required.

BFS is still useful for analysis.

7 Conclusion

This project successfully implements:

- DFS planner based solely on discovered walls.
- Live sensing and internal map updates.
- Step-by-step execution with blocking detection.
- Automatic re-planning using DFS when necessary.
- Optional BFS planner for comparison.

The robot navigates the maze incrementally and robustly, satisfying all requirements of the assignment.