CAPSTONE PROPOSAL

BY: IBRAHIM MUHAMMED

PROJECT DESCRIPTION

*This project takes inspiration from **Micromouse** competitions, wherein a robot mouse is tasked with plotting a path from a corner of the maze to its center. The robot mouse may make multiple runs in a given maze. In the first run, the robot mouse tries to map out the maze to not only find the center, but also figure out the best paths to the center. In subsequent runs, the robot mouse attempts to reach the center in the fastest time possible, using what it has previously learned.*

In this project, we will create functions to control a virtual robot to navigate a virtual maze. A simplified model of the world is provided along with specifications for the maze and robot;

our goal is to obtain the fastest times possible in a series of test mazes.

PROBLEM STATEMENT

The goal is simple - GET TO THE CENTER OF THE MAZE AS FAST AS POSSIBLE. The robot will start in the bottom-left corner of the maze and must navigate from the origin to the goal in the shortest possible time, and in as few moves as possible. The maze is a square with equal rows and columns where the number of squares along each side (n) must be 12, 14, or 16. The entire perimeter of the maze is enclosed by walls preventing the micromouse from traveling outside the maze. The starting position will have walls on the left, right, and back sides, so the first move will always be forward. The goal area will be made up of a 2x2 square in the center of the maze.

The micromouse will run a first trial that is exploratory in nature where it will aim to build knowledge of the structure and shape of the maze, including all possible routes to the goal area. The micromouse must travel to the goal area in order to complete a successful exploration trial, but is allowed to continue its exploration after reaching the goal. The second trial is where the action happens. In this trial, the micromouse will recall the information from the first trial and will attempt to reach the goal area in an optimal route.

DATASETS AND INPUTS

The information of the maze layout is provided in a text file. The first line of the text file describes the number of squares for each side of the maze. The following lines define the actual structure of the maze via comma-separated binary values that define where all of the walls and openings are located in each square.

For each move, the micromouse must have knowledge of its current environment - its exact location in the maze, the number of walls surrounding it, what its intended action will be, and any prior information and routes pertaining to the maze from previous trials.

The location of the micromouse will be defined by a 2-digit pair, such as [5,13]. Each action contains movement which will be a number from -3 to 3, and rotation which will be -90, 0, or 90.

SOLUTION STATEMENT

Each trial attempts to solve a different problem. During the first trial, the main problem to solve is - what is the layout of the maze? A viable solution during the first trial will be a detailed knowledge of the maze layout and all possible paths leading to

the goal area. During the second trial, we are trying to reach the goal in an optimized route, so the target solution would be reaching the goal in the fewest possible moves.

This environment is easily quantifiable, because we are constantly assessing the number of total moves taken along with the total time the micromouse is navigating the maze. After the exploration trial, the micromouse should have recorded knowledge of multiple possible routes to the goal, and should choose the shortest route (or the route that requires the fewest steps) during its optimization round. The solution can be replicated by repeating the optimization trial for additional rounds. Assuming the micromouse initially identified and traveled in an optimal route, its path to the goal should remain the same if it were subjected to additional trials.

BENCHMARK MODEL

The micromouse's performance will be scored by adding the following:

- *Number of total steps in the exploration trial divided by 50*
- *Total number of steps to reach the goal in the optimization (second) trial*

Each trial is capped at 1000 steps. The micromouse can only make 90 degree turns (clockwise or counterclockwise) and can move up to 3 spaces forward or backwards in a single movement.

$$Score = No.\ steps\ in\ trial2 + \left(\frac{No.\ steps\ in\ trial1}{50}\right)$$

Because we have capped the number of possible steps at 1000, this will be the most basic form of a benchmark performance. The benchmark model will consist of a micromouse that makes a random movement decision at each step. If the mouse is able to navigate to the goal in fewer than 1000 steps during its exploration trial, a new benchmark should be set to the total number of steps in Trial 1. Because the micromouse has gained knowledge of the maze layout in its exploration trial, it should then optimize its route to the goal in Trial 2. Regardless of the number of steps taken during Trial 1, by nature of design, each maze has an optimal route - which is the minimum amount of steps required to get from the starting location to the goal area.

EVALUATION METRICS

As discussed in previous sections, the main evaluation metric to quantify the performance of the benchmark and solution models is:

$$Score = No.\ steps\ in\ trial2 + \left(\frac{No.\ steps\ in\ trial1}{50}\right)$$

This evaluation metric is impacted by both the exploration trial and optimization trial, however the optimization trial will impact the score significantly more than the exploration trial. Our goal is to minimize this score, which would be the result of an optimal trial.

PROJECT DESIGN

To approach a solution, my workflow will begin with defining and understanding all essential project parameters. I will then analyze all of the starter code that is given: robot.py, maze.py, tester.py, showmaze.py, and test_maze_[1:3].txt. Next, I will preprocess all of the data which is entirely handled in the Maze class, where the contents of the maze are given in a text file and decoded into a virtual maze. After getting the mazes set up, I will begin to modify robot.py by adding appropriate algorithms and functionality suited for exploration and optimization trials. I will then run the algorithms with my benchmark model and record the performance, then run the optimized robot.py model and record the performance. I will test my results using tester.py, and will graphically visualize the micromouse performance in each maze. If the micromouse did not find an optimal solution, then I will review robot.py and repeat the process until it finds the optimal solutions. Lastly, I will document my process and write up a thorough report.

There are several algorithms that I am considering using for this project. Both the exploration trial and the optimization trial will need appropriate algorithms because each trial serves a different purpose.

***Dead Reckoning**: Algorithm that makes a random movement decision at each step*

***Flood Fill**: Algorithm that starts in the center area and will fill each surrounding cell with a number of its relative distance from the goal area.*

***Wall Follower**: A Wall Follower algorithm works by simply directing the micromouse to take every left or right turn possible, which should eventually lead to the goal area (assuming the micromouse does not get caught in a circular loop). An example of a Left Hand Rule (LHR) is shown below.*

A: A* is a best-first search algorithm that searches for all possible solutions to the goal area and will choose the shortest path from the starting node to the ending node. Starting from a specific node, it creates a tree path of each possible direction until it reaches the goal area.*

***Dijkstra's**: Dijkstra's algorithm is an algorithm that finds the shortest path from one node to all other nodes in its network. By finding the distance to all nodes in its path, it will inevitably create a connection with the nodes represented by the goal area. Unlike A*, which focuses on finding the path between two single nodes, Dijkstra's algorithm links a path between all nodes on a network.*

***Breadth-First Search:** Is an algorithm that will start at a tree root and consider its closest neighbors before looking on to its next level neighbors.*

***Depth-First Search:** This is similar to Breadth-First Search, but will start at a root and follow a branch as far as possible before considering a different path.*