Robot Motion Planning

Domain Background

In this project, a robot mouse is tasked with plotting a path from a corner of the maze to its center. The robot mouse may make two 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, based on the provided simplified model of the world, I have created functions to control a virtual robot to navigate a virtual maze. My goal was to obtain the best score in a series of test mazes.

Problem Statement

On each maze, the robot must complete two runs. In the first run, the robot is allowed to freely roam the maze to build a map of the maze. It must enter the goal room at some point during its exploration, but is free to continue exploring the maze after finding the goal. After entering the goal room, the robot may choose to end its exploration at any time. The robot is then moved back to the starting position and reset for its second run. Its objective now is to go from the start position to the goal room in the fastest time possible.

Datasets and Inputs

Most mazes have an entrance and an exit. You enter the maze and try to escape. In contrast to most mazes, this instead is a closed square maze with no entrance or exit since we do not want the mouse to escape. The maze exists on an n x n grid of squares. The value of “n” can be 12, 14 or 16. Along the outside perimeter of the grid, and on the edges connecting some of the internal squares, are blocked by thin pieces of wood so the mouse cannot climb the obstacles.

We put the mouse in the square in the bottom left corner of the grid facing upwards. A goal room consisting of a 2 x 2 square in the center of the grid is set up with some food inside. The mouse then explores the maze to obtain the food.

Solution Statement

The robot mouse lives in a discrete world moving from one square to another with only the ability to strictly face either north, east, south or west. The mouse can sense the distances to the left, front and right obstacles. In this discrete world, the time is also discrete. On each time step, the robot may choose to rotate clockwise or counterclockwise in ninety-degree pivots, then move forwards or backwards with up to three distance units. The robot’s turning and movement is perfect. If the robot tries to move into a wall, the robot stays where it is and its location is not progressed.

Benchmark Model

Different controllers (such as Random, Dead-End, Counter, Heuristic) cause different performances. I will compare the path length, optimal moves, and score each controller obtains. The random controller will set the worst score benchmark. Once the robot has visited every area within the maze, I will start the second run using the dynamic programming method. We can find the optimal path which will be set as the optimal path benchmark.

Evaluation Metrics

The robot's score for the maze is equal to the number of time steps required to execute the second run, plus one thirtieth the number of time steps required to execute the first run. A maximum of one-thousand time steps is allotted to complete both runs for a single maze. That should be reasonable due to the score and in addition, taking into account both the second and first run as well, it balances the time and the accuracy. Apparently, the robot can find the optimal path if it explores the whole maze. So, if the score is equal to the number of time steps required to execute the second run, the robot will choose to explore the whole maze in the first run. Although the robot mouse gets the optimal path, it wastes a lot of time.

Project Design

In the first run, I will use four algorithms to exploring the maze.

- Random
- Dead-End
- Counter
- Heuristic

With the Random Controller, the robot gets valid potential rotations and movements, and based on the sensors it just randomly chooses one. The robot is moving randomly with its eyes open, so it will never hit the wall. It also knows how to move backwards to escape a dead end, but it has poor memory so it may return to the same dead end or get trapped in a loop as well.

Before we start the second run, I will map the maze based on the collected data, then use dynamic programming to get the value for how long the path of each location to the goal. Using these values, I can get the optimal path from any place of the maze to the goal. In the Methodology section, I will explain the details of dynamic programming.

In the second run, on each step the robot will check all the available places it can move to, find the place that has the smallest value and move there. Finally, the robot mouse will hit the goal with the optimal path/moves.

References

- Plot and Navigate a Virtual Maze - Project Description
- Plot and Navigate a Virtual Maze - Naoki Shibuya (report example 3)