Plot and Navigate a Virtual Maze

Project Overview

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. In this project the goal is to code a robot mouse which is capable of plotting a path from a corner of the maze to its center. The robot mouse is given multiple tempts to complete the maze. In the first run the robot must explore the map and try to find the target location. In the second run it must reach the target location using the best path it has found.

Problem Statement

Code a robot mouse which can navigate through a maze to reach the goal area. This task must be performed in two runs. The robot is given one thousand steps in each run. In the first run the robot must explore the maze to build a map of the maze and find the path to the goal area. In the second run the robot must use the map it built in the first run to reach the goal area using the shortest path possible. The robot must accomplish these task while not crossing the thousand step limit. The robot cannot move through walls. If the robot tries to move through walls it stays where it is. The maze through which the robots navigates through is of dimension n*n. The minimum value for n is twelve and maximum value is sixteen. One way to solve this would be to explore the map till the robot finds the goal area. When the goal area is found the robot can store the path to it and use that information to get to the goal area in the second run.

Metrics

The robot is given two turns to solve the maze. In the first turn, the robot explores the map looking for the goal area and building a map of the maze. If the robot enters the goal area it may choose the end the run or continue to explore the map. When the robot selects to end the run it is moved back to the start position. In the second run the robot has to reach the goal area using the fastest path it has found. The final score of the robot is the number of 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 are allotted to complete both runs for a single maze.

Data Exploration

The input received from the three sensors by the robot is the number of steps till a obstacle is encountered. The sensors takes readings from the front, right and the left. There is no sensor on the back of the robot. For example if the sensor data is (1,2,3) then it means that the robot can move one, two and three steps to the left front and right respectively before it encounters an obstacle. The data received from the sensor is perfect which means there is no noise in it. Since the data is perfect there is no requirement for data processing.

The maze is provided to the program in form of a text file. In the file the first number gives the dimension of the maze. Then the rest of the lines describe which edges of the square are open and which are closed. Each number is a representation of a four bit number that has bit value of 0 if an edge is walled and 1 if an edge is not walled. Here 1, 2, 4, 8 registers represent up, right, down and left respectively.

The maze is represented as a graph, where every space in the maze is a vertex, and we have edges connecting adjacent spaces. Since the robot can move on each turn up to 3 spaces (forward or backwards), we have edges connecting vertices with a distance up to 3 spaces. For example:

Vertex (2,0) would be connected to vertices (1,0), (0,0), (2,1), (2,2) and (2,3). It would not be connected to vertex (2,4) since it's Manhattan distance is bigger than 3. In order to represent the spaces as integer numbers, given the row i and column j of a space, we transform it into an integer as i*100+j. This way, every space in the maze is represented by a unique number. For example, position (0,0) will be vertex 0. Position (3,2) is vertex 3*100+2 = 302, and so on. We also provide a operation to obtain the coordinates from the vertex number (decode function). Moving from a space to the north (up) can be done by adding 100 to a vertex number. Moving south (down) by subtracting 100 from the vertex number. Moving to the west (left) is achieved by subtracting 1, and adding 1 we obtain a movement to the east (right).

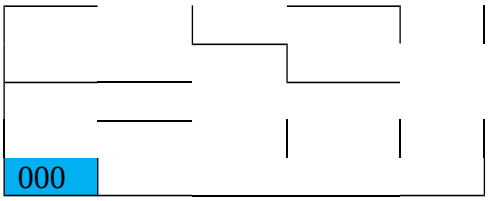
Algorithms and Techniques

Phase 1: exploring
In the exploring phase we have two lists. The explored list and the notexplored list. The explored list contains all the vertices that have been found, and have been explored. The notexplored list contains all the vertices which we have found (we know how to get there), but we still have not explored. Initially the explored list is empty, and the notexplored list contains only vertex 0 (initial position of the maze). In each iteration during the exploration phase we are either on an explored or on a notexplored vertex.
Whenever we are on a notexplored vertex we mark it as explored, meaning we get the information from the sensors, so we can now get connectivity information for the vertex (which other vertices is the vertex connected to). Then, we have to plan the next movement. A BFS algorithm is run to get the distance from the current vertex to all the notexplored vertex. Then our next target will be the notexplored vertex closer to the current vertex. In case of a tie we choose as next vertex the one closer to one of the goals. The BFS algorithm computes the distances to each vertex, and the paths to them are stored in a list which contains the previous vertex to get to each vertex. This way we can build the path to get to the next target.

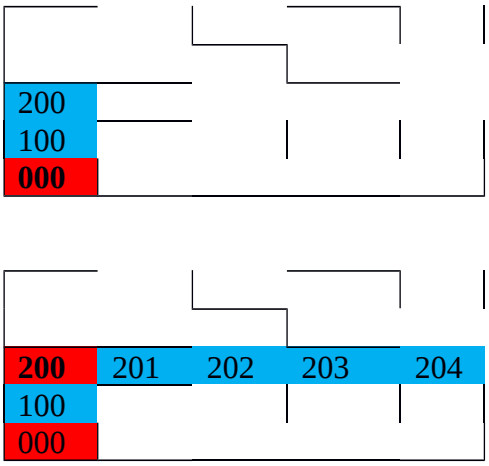
If we are on an explored vertex, there must be a path we are following (because explored vertices can be part of a path, but they are never the target of a path), so we simply get the next vertex on the path to decide our following movement. The following movement is decided by taking the current position

(i1,j1) and the following positon (i2,j2). Since the only possible movements are always on the same column or on the same row, either i1 = i2 (same row) or j1 = j2 (same column). Depending on which case we are on, and which direction we are heading, we decide in which direction the robot should move next (-90, 0, 90), and how many spaces should we move (1,2,3 or -1, -2, -3), negative numbers can be used which means we need to move backwards.

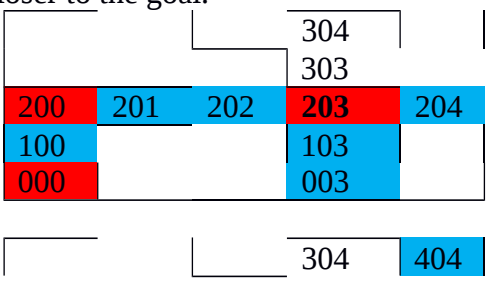
Following the previous example, on the first iteration we only have the node 0 as notexplored, and the explored list is empty. Blue squares are notexplored, and red squares and explored.



The first iterations of the algorithm are shown next



In this point vertices 201, 202, 203 and 100 are all at distance 1 from vertex 200 (distance 1, because it's just one instruction to the robot, the instruction could be either (90, 1), (90, 2), (90, 3) or (0,-1)). Vertex 203 is chosen since it is closer to the goal.



			303	304
200	201	202	203	**204**
100			103	104
000			003	004

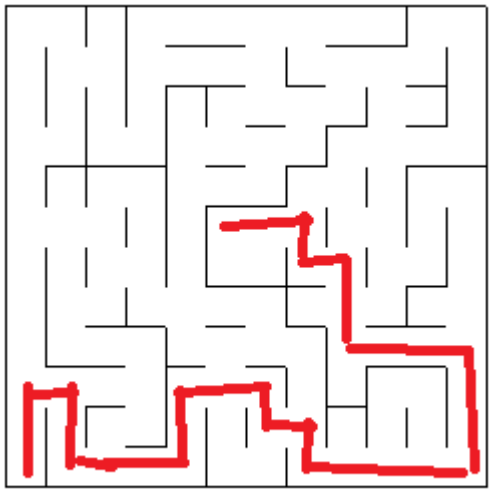
This process keeps going until a goal is reached. At this moment phase 1 stops, and we send a (Reset, Reset) command in order to go to phase 2. At this point we know at least one path from position (0,0) to the goal, so we simply run the BFS algorithm, this time from position (0,0) to get the path to the goals. Then we build the path, using the list of previous vertex, as we did previously, and obtain the path list.

If we have explored the nodes which happen to be closer to the goal but we have not reached the goal then we explore the nodes which have not been explored to see if they take us to the goal area.

While exploring our goal is to always make moves which take us closer to the goal. Since the goal area lies in the center of the maze it can be calculated if the dimensions of the maze is given. If the robot has explored the nodes which takes it closer to the goal and has not been able to get to the goal, it will explore the nodes which have not been explored even if those nodes take the robot away from the goal area. These nodes which are not explored are stored in the list notexplored as mentioned above.

Phase 2: maze solving
The second phase is quite simple since we already computed the path from the starting position to the goal. On each iteration we simply get the next node from the path list (which was computed at the end of phase 1), and we compute the next movement as previously explained.

Exploratory Visualization



This is the maze one which was provided for testing. The red line in the maze shown is the path taken by the robot in the second run.

Benchmark

The score given to the robot is the sum of one-thirteenth the number of steps taken during the first run and the number of steps taken during the second run. If robot explores roughly half of the maze during before reaching the goal area then for the first maze, which is of 12x12 in dimension, there are 144 cells and let us assume that the shortest path to the goal area is around thirty steps then we get a score of 35.53 for the first maze. Similarly, for the second and third maze, which are of 14x14 and 16x16 in dimension respectively, there are 196 and 256 steps and 40 and 50 steps to reach the goal area respectively. Therefore, we get a score of 47.53 and 59.8 for third and fourth maze respectively.

Data Preprocessing

There is no need for data processing as the the sensor data is always correct and the environment design provided does not contain any flaws.

Implementation

This section describes functions of the program

next_move_phase1: This function implements the first phase (exploration). First it checks if we are already in a goal state. If so, it changes to the second phase and computes the path for the second phase. If we are not on a goal state yet, the function checks if we are in a explored or notexplored vertex. When we are in an explored vertex we simply move to the next vertex on the path. When we are in a notexplored vertex we first explore the current vertex, which means going on every direction read from the sensors and connecting the vertices on that direction. For example, if the sensors tell us that on the east direction there is a 5, and we are on vertex (2,3), this means that we can go from (2,3) to (2,4), (2,5) and (2,6). And then from (2,4) to (2,5), (2,6) and (2,7). From (2,5) to (2,6), (2,7) and (2,8). From (2,6) to (2,7) and (2,8). And from (2,7) to (2,8). All the respective edges have to be added to the graph. Once the graph has been updated (the current vertex was explored), we update the explored and notexplored lists, and run the BFS to get the distances to all notexplored vertices. When the BFS ends we use the distances to chose the next vertex to be explored, and we finally build the path from the current vertex to the chosen vertex.

sensor_data: This function transforms the sensors reading which is relative to the robot's heading, to an absolute direction. The same sensors reading can mean something different depending of where the robot is heading. For example, if the sensors input is (3,0,2) and the robot is heading north, this means 3 open spaces to the east, 0 open spaces to the north, and two open spaces to the east. The same input, when the robot is heading west means 3 open spaces to the south and 2 open spaces north. The input to the function is always a list with three elements (input from the sensors) and the output is always a four elements list (an absolute value on each direction, north, east, south and west).

new_movement: This function computes the new movement based on the current path and position. The path is already computed, and the next node is in path[0]. So we want to move from current to path[0]. Since these two nodes are consecutive, they must be either in the same column or row.

bfs: This function does breath first search to compute all distances from the starting vertex. Given the vertex "start", it will compute the shortest path to every other node in the graph.

decode: This function takes the value of an given node and decodes the i,j coordinates.

add_edge: This function is used to add an edge between two nodes.

dist_to_goal: This function is used to calculate the distance to the goal area. It returns the distance to the closest goal area out of the four.

dist: This function is used the calculate the distance between two nodes.

Refinement

At first the idea was to build an entire map of the maze and then apply different algorithm to reach the goal area but mapping the entire maze appeared to be difficult as there were no way of finding out if the entire maze was mapped. However, instead of mapping the entire maze if we prioritize just getting to the goal and storing the path to the goal the problem becomes simpler.

Model Evaluation and Validation

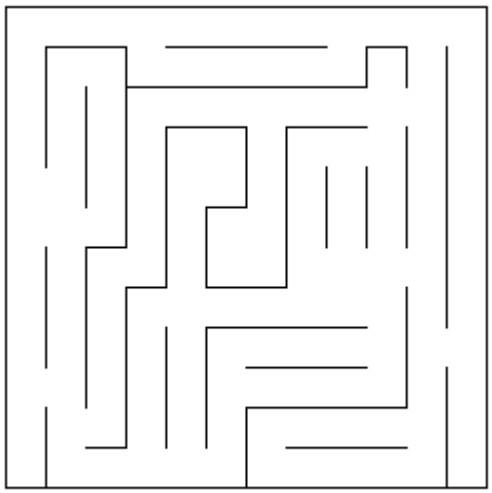
If we put one or two edges in the path to the goal area which the robot has found then the robot will find an alternate path to the goal area if there is one. The method used by the robot to find the path is explained in the above sections.

The robot will be able to complete any maze as long as there is at least one path leading to the goal area. We can verify this in the fourth maze by putting up some edges and opening others there is difference in the score as the robot takes a different path to the goal area.

Justification

The benchmark we were aiming for was a score of 35.53, 47.53 and 59.8 for maze of size 12x12, 13x13 and 14x14 respectively. The robot got a score of 22, 39.5, 35.33 and 11.733 for the first, second, third and fourth respectively which is better than the expected result. The first, second, third and fourth maze are of dimension 12x12, 14x14, 16x16 and 12x12 respectively.

Free-Form Visualization



This is a 12x12 maze designed for the further testing of the robot. The robot is able to complete the maze within the requirements with a score of 22.

Reflection

The difficult part in the project was coming up with the way the maze was to be stored and the way the node coordinates were to be stored as a single number.

The interpretation of the sensor value was also difficult. The input from the sensor is a list with three value which gives the distance the robot can move in the direction left, right and forward before encountering a wall. As these directions are relative it was better to convert them to absolute values north, east, south and west.

Then there was the problem of exploring the maze in the first run. It was difficult to tell when the robot had explored the entire maze. So, instead of exploring the entire maze if we explore the maze till the robot finds the goal area then we have a path from the start location to the goal area and the problem becomes much simpler.

Therefore, in the final version the robot uses the first run to explore the map till it has found the goal area. Since, the robot has reached the goal area it has the path from the starting area to the goal area. The robot uses breath first search algorithm to find the shortest distance from one node to the other nodes. The robot uses this to find the shortest distance to the goal area from the start area. In the second run the robot uses this path to reach the goal area.

Improvement

If the scenario took place in a continuous domain. Where each square has a unit length, walls were 0.1 units thick, and the robot was a circle of diameter 0.4 units then we would have to take these measurements into consideration. When moving from one square to the next we have to add the thickness of the wall to the distance moved while taking turns. It would also not be necessary to take 90 degree turns and this would have to be taken into account while designing a robot which navigates maze in continuous domain.