MazeWorld

cs 76, Artificial Intelligence
Fall 2024
Assignment 1
Ahmad Wahab

Description:

Multi-robot Problem:-

For multi robot problem, we represented the current state and start state as a tuple containing the coordinates of each of the robot and the the robot whose turn it was to move (the sequence was (turn number,x1,y1,x2,y2,….xi,yi). For the Heuristic, we used the Manhattan heuristic from the goal which is the sum of the distances of each robot from the goal in the x axis and in the y axis. An interesting decision was to keep the goal to be of length n-1 where n is the length of our state representation. This was because we do not care about the robot turn number once we do reach the goal. Meanwhile, the get_successors function considers which robots turn it is to move and it gives n successors back for a state where n <6. This is because the robot can potentially move up,down,right and left but it can also choose to stay at its position and pass the turn to the next robot. The get_successors function uses a helper function called check_safe that makes sure that any future coordinate is calculated. For example if I say move up robot 1 then the new coordinate of robot1 would be the y coordinate it had plus 1. The check safe function makes sure that x,y+1 is a moveable place i.e it is a floor and not outside the grid itself.

Blind-robot Problem:-

For Blind robot problem, we represented the current state and start state as a tuple containing the x and y coordinates of all possible places the robot can be situated at plus the move it made to get to the current state from its parent state. This number ranges from -1 to 3 where -1 means it's the start_state and each of the numbers from 0 to 3 indicate a movement ranging from up, down, right, left. (the sequence in the state was (movement made,x1,y1,x2,y2,….xi,yi). An interesting decision was to use the Manhattan heuristic again with a slight variation i.e from the goal to each of the current possible belief states. This allowed me to run my A* algorithm only once to get from any possible location to my goal. This heuristic localized me AND got me closer to the goal with each move. The goal is of length n-1 where n is the length of our state representation. This was because we do not care about the last move made to get to the goal. Meanwhile, the get_successors function gives n successors back for a state where n <5. This is because the robot can potentially move up,down,right and left.

Astar_search:-

The AstarNode class represents a node in the search tree, containing the state of the problem, its heuristic value, a reference to the parent node, and the cost of the transition from the parent to the current node. The priority method computes the priority of a node based on the sum of its transition cost and heuristic value, which helps the algorithm decide which nodes to explore first. The backchain function traces the solution path from the current node to the root by following parent references and building the path in reverse order. The astar_search function initializes the search with the start state and pushes nodes into a priority queue (heap) while exploring successor states. It tracks the cost of visiting nodes in the visited_cost dictionary to avoid redundant or inefficient exploration. The function continues exploring until it reaches a goal state, at which point it returns the solution with the optimal path and associated cost.

Testing:-

For testing I have mazes ranging from 0 to 8. maze 0 test is to show what happens when a solution is not found. The mazes 1 and 2 show that my algorithm works with a single robot and then for the future mazes, I set it to 3 to show that we can have any number of robots that are computationally feasible. maze 4 was an interesting one because it starts with all 3 robots cooped up together in a tight and the way to escape is through a bottleneck where only 1 robot can pass through at a time and then fit into another cooped up space as the goal. For maze5 the goal was near the start state but separated by a long wall, hence you have to technically move away from the goal, climb up the wall and then get to the goal. This is a good test of my heuristic to show that it can decide to go away from the goal if it encounters a long wall. In Maze 6 and 7, the robots had to orient themselves differently in the same place but it was a tight space with only 1 direction to move into. For maze 7, robots had to go to a wider space to reorient themselves before going back to goal. For maze8, it was a random ascii art to show that robots can find their way through complicated and large mazes efficiently.

Naturally, for similar reasons these mazes became good testing grounds for the blind robot as well which performed well on them too and found a path to goal.

EXTRA CREDIT:-

To visualize that my blind robot can give directions such that if it starts anywhere on the map which is a floor, it can to go to a goal that is reachable and is also a floor, I devised a graphics file that will let you see that PATH(colored green) a blind robot will take from START (coloured blue) to GOAL (colored orange). Please note that floors are white and obstacles are red.

- MY BLIND PROBLEM only needs to run Aster one to to localize and get to goal. The prof told me it would count as extra credit because it goes beyond the class demands and its more efficient.

LOOK AT README FOR MORE DETIALS

Discussion Questions:

Q. If there are k robots, how would you represent the state of the system? Hint -- how many numbers are needed to exactly reconstruct the locations of all the robots, if we somehow forgot where all of the robots were? Further hint. Do you need to know anything else to determine exactly what actions are available from this state?

Answer:- You can represent it by a tuple of length 2k+1 where the x and the y coordinate of each robot is represented hence 2 values per robot for k robots is equal to 2k and you also need to represent the turn number hence 1 more value which in total leads to a tuple of 2k+1 length.