CS76, Fall 2021, Assignment 2, Tracey Mills

Description

I implemented A* search as discussed in class and the textbook. I used a heap to implement my priority queue so that pushing and popping would be efficient. Instead of removing nodes from the queue when a shorter path to that node was found, I kept a dictionary recording the minimum encountered cost of getting to a certain state. Then, after popping the next node from the queue, I only expanded it if its cost matched the stored minimum cost for the node's state. Otherwise the node was ignored, since the same state with a lower visiting cost was waiting in the queue. This saved time by not having to remove nodes from the queue, but required me to keep a dictionary of the costs for each state.
As for my problem models, their get_successor functions and goal_test functions function similarly so that they are both compatible with the A* file. Both return successor states as lists of tuples, and recieve states as tuples to check if the state is a goal state.

Evaluation

Multi-robot Problem

When testing on the 5x6 maze with 3 bots, using the manhattan heuristic greatly reduced the nodes visited and path cost over the null heuristic (uniform cost search) as shown below:

```
Mazeworld problem:
attempted with search method Astar with heuristic null_heuristic
number of nodes visited: 2000
solution length: 64
cost: 32
path: [(0, 1, 0, 1, 1, 2, 1), (1, 1, 0, 1, 1, 2, 1), (2, 1, 0, 1, 1, 2, 1), (0, 1, 0, 1, 1, 2, 2), (1, 1, 0, 1, 1, 2, 2), (2, 1, 0, 2, 1, 2, 2), (0, 1, 0, 2, 1, 2, 2), (1, 1, 1, 2, 1, 2, 2), (2, 1, 1, 2, 1, 2, 2), (0, 1, 1, 2, 1, 3, 2), (1, 1, 1, 2, 1, 3, 2), (2, 1, 1, 2, 1, 3, 2), (0, 1, 1, 2, 1, 3, 1), (1, 1, 2, 2, 1, 3, 1), (2, 1, 2, 2, 1, 3, 1), (0, 1, 2, 2, 1, 3, 1), (1, 2, 2, 2, 1, 3, 1), (2, 2, 2, 2, 1, 3, 1), (0, 2, 2, 2, 1, 3, 1), (1, 3, 2, 2, 1, 3, 1), (2, 3, 2, 1, 1, 3, 1), (0, 3, 2, 1, 1, 2, 1), (1, 2, 2, 1, 1, 2, 1), (2, 2, 2, 1, 1, 2, 1), (0, 2, 2, 1, 1, 2, 1), (1, 1, 2, 1, 1, 2, 1), (2, 1, 2, 1, 1, 2, 1), (0, 1, 2, 1, 1, 2, 1), (1, 1, 3, 1, 1, 2, 1), (2, 1, 3, 1, 1, 2, 1), (0, 1, 3, 1, 1, 3, 1), (1, 1, 3, 1, 1, 3, 1), (2, 1, 3, 1, 0, 3, 1), (0, 1, 3, 1, 0, 3, 1), (1, 1, 4, 1, 0, 3, 1), (2, 1, 4, 1, 0, 3, 1), (0, 1, 4, 1, 0, 3, 1), (1, 2, 4, 1, 0, 3, 1), (2, 2, 4, 1, 1, 3, 1), (0, 2, 4, 1, 1, 3, 1), (1, 3, 4, 1, 1, 3, 1), (2, 3, 4, 1, 2, 3, 1), (0, 3, 4, 1, 2, 3, 2), (1, 3, 4, 1, 2, 3, 2), (2, 3, 4, 1, 3, 3, 2), (0, 3, 4, 1, 3, 3, 2), (1, 3, 4, 1, 3, 3, 2), (2, 3, 4, 1, 4, 3, 2), (0, 3, 4, 1, 4, 3, 2), (1, 3, 4, 1, 4, 3, 2), (2, 3, 4, 2, 4, 3, 2), (0, 3, 4, 2, 4, 2, 2), (1, 3, 4, 2, 4, 2, 2), (2, 3, 4, 1, 4, 2, 2), (0, 3, 4, 1, 4, 1, 2), (1, 2, 4, 1, 4, 1, 2), (2, 2, 4, 1, 4, 1, 2), (0, 2, 4, 1, 4, 1, 1), (1, 2, 4, 1, 4, 1, 1), (2, 2, 4, 1, 3, 1, 1), (0, 2, 4, 1, 3, 1, 1), (1, 1, 4, 1, 3, 1, 1), (2, 1, 4, 1, 3, 1, 1), (0, 1, 4, 1, 3, 1, 2)]
```

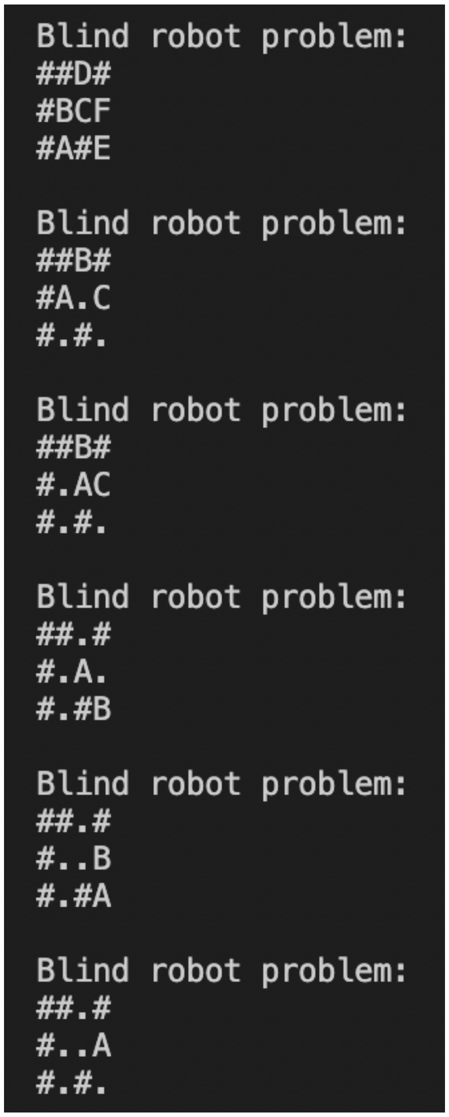
Mazeworld problem: attempted with search method Astar with heuristic manhattan_heuristic
number of nodes visited: 101
solution length: 55
cost: 26
path: [(0, 1, 0, 1, 1, 2, 1), (1, 1, 0, 1, 1, 2, 1), (2, 1, 0, 1, 2, 2, 1), (0, 1, 0, 1, 2, 1, 1), (1, 1, 0, 1, 2, 1, 1), (2, 1, 0, 1, 3, 1, 1), (0, 1, 0, 1, 3, 1, 2), (1, 1, 1, 1, 3, 1, 2), (2, 1, 1, 1, 3, 1, 2), (0, 1, 1, 1, 3, 2, 2), (1, 1, 2, 1, 3, 2, 2), (2, 1, 2, 1, 3, 2, 2), (0, 1, 2, 1, 3, 2, 1), (1, 1, 2, 1, 3, 2, 1), (2, 1, 2, 1, 3, 2, 1), (0, 1, 2, 1, 3, 1, 1), (1, 1, 2, 1, 3, 1, 1), (2, 1, 2, 1, 3, 1, 1), (0, 1, 2, 1, 3, 1, 0), (1, 1, 2, 1, 3, 1, 0), (2, 1, 2, 1, 4, 1, 0), (0, 1, 2, 1, 4, 1, 0), (1, 1, 3, 1, 4, 1, 0), (2, 1, 3, 1, 4, 1, 0), (0, 1, 3, 1, 4, 1, 1), (1, 1, 3, 1, 4, 1, 1), (2, 1, 3, 1, 4, 1, 1), (0, 1, 3, 1, 4, 1, 2), (1, 1, 3, 1, 4, 1, 2), (2, 1, 3, 2, 4, 1, 2), (0, 1, 3, 2, 4, 1, 2), (1, 1, 4, 2, 4, 1, 2), (2, 1, 4, 2, 4, 1, 2), (0, 1, 4, 2, 4, 1, 1), (1, 1, 4, 2, 4, 1, 1), (2, 1, 4, 3, 4, 1, 1), (0, 1, 4, 3, 4, 1, 1), (1, 1, 4, 3, 4, 1, 1), (2, 1, 4, 3, 3, 1, 1), (0, 1, 4, 3, 3, 1, 0), (1, 1, 4, 3, 3, 1, 0), (2, 1, 4, 3, 2, 1, 0), (0, 1, 4, 3, 2, 1, 0), (1, 1, 4, 3, 2, 1, 0), (2, 1, 4, 2, 2, 1, 0), (0, 1, 4, 2, 2, 1, 0), (1, 1, 4, 2, 2, 1, 0), (2, 1, 4, 1, 2, 1, 0), (0, 1, 4, 1, 2, 1, 0), (1, 1, 4, 1, 2, 1, 0), (2, 1, 4, 1, 3, 1, 0), (0, 1, 4, 1, 3, 1, 1), (1, 1, 4, 1, 3, 1, 1), (2, 1, 4, 1, 3, 1, 1), (0, 1, 4, 1, 3, 1, 2)]

Continuing to use the manhattan distance heuristic, I was able to solve puzzles of size 10x10, 20x20, 30x30, and 40x40, with 1-3 robots. These tests can be run in the test_sensorless.py file. Mazes with tight corridors which forced robots to wait turns or navigate around each other had more nodes be explored, with the 30x30 maze requiring robots to squeeze through a small passage in the middle of the maze in turn taking the longest out of all those I tested. Stats for this maze are shown below, with the path abbreviated.
Mazeworld problem:
attempted with search method Astar with heuristic manhattan_heuristic
number of nodes visited: 1796630
solution length: 625
cost: 228
path: [(0, 0, 0, 19, 0, 10, 6), (1, 0, 1, 19, 0, 10, 6), ... (2, 19, 19, 0, 0, 0, 18), (0, 19, 19, 0, 0, 0, 19)]

Sensorless Robot Problem

The sensorless robot was able to deduce its belief state from a large range of mazes, from sizea 4x3 to 30x30. It did so for a 5x6 maze in just 6 moves, with the progression of its belief state shown below.

4x3 Maze with 3 robots:



```
Blind robot problem:
##D#
#BCF
#A#E

Blind robot problem:
##B#
#A.C
#.#.

Blind robot problem:
##B#
#.AC
#.#.

Blind robot problem:
##.#
#.A.
#.#B

Blind robot problem:
##.#
#..B
#.#A

Blind robot problem:
##.#
#..A
#.#.
```

Even in a 30x30 maze, it ran quickly, and just 10488 states were visited before finding a solution path of length 1118.

Discussion

Multi-robot Problem

1. With k robots, the system can be represented as a tuple of length 2k+1. The first number in the tuple indicates which robot is to move next. The only valid moves from the current state or those in which this robot moves N,S,E or W. The next 2k numbers indicate the 2d positions of each of the

k robots, which are necessary for reconstructing each robot's position on the board.

2. An upper bound for the number of states is k(nˆ(2k)), since there are k options for the first number (k possible robots), and then n options for the next 2k numbers (n possible x coordinates, n possible y coordinates).
3. Considering collisions to be when a robot occupies a space where there is a wall, there are w locations in the nxn rectangle in which a collision might occur. The probability of any one robot causing a collision is therefore w/(nˆ2), and so the probability of any of the k robots causing a collision is about kw/(nˆ2), given that n is much larger than k. This gives us roughly (kw/(n$^{2))(k(n}$(2k))) = (k$^{2)(w)(n}$(2k-2)) collision states.
4. This gives us 10(100ˆ(20)) possible states. By the estimate above, about (w)(100ˆ19) of these will be collision states. So, we have 10(100ˆ20) - w(100ˆ19) = (100ˆ19)(1000-w) truly possible states, which is an extremely large number. Since breadth first search must visit and store every single state, this is not computationally feasible.
5. A monotonic heuristic for this search space is manhattan distance. Manhattan distance is the number of vertical or horixontal movements required before reaching a goal, ignoring obstacles. For a monotonic heauristic h, h(n) <= c(n,a,n') + h(n'), where n and n' are states, and c(n,a,n') gives the cost of moving from n to n'. Let m(n1,n2) give the manhattan distance between states n1 and n2. We know that m(n,n') <= c(n,a,n'), since c is the number of steps from one state to the other, and manhattan distance, taking the most straightforward path between states, cannot overestimate this number. Also, we know that m(n1,n2) + m(n2,n3) = m(n1,n3). So, with goal node g, we know that
   m(n,g) = m(n,n') + m(n',g) <= c(n,a,n') + m(n'g).
   We then see that this is the definition of a monotonic heuristic.
6. The 8 puzzle problem is an instance of this problem with a 3x3 rectangle, 8 robots, and no walls. The start and goal locations are those given in the book. Manhattan distance is a good heuristic for this puzzle because it is admissable and monotonic, as described above.
7. To prove that the state space of the 8 puzzle is two disjoint sets, we could generate the entire state space by taking every ordering of the 9 locations in the grid ((0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)). For each ordering we get 8 states by assigning the first location to the first location to the first robot, the second to the second robot, and on, leaving out the last location for the blank square. We have 8 different states with these given locations, one for each robot that could be up to move. Once we have this complete state space, A, we can use our algorithm to show that it is disjoint.
   First, we make our goal test always return false. Then, starting from any state in A, run the algorithm and record which states are visited. Call this set of states B. Since the algorithm will visit nodes until a solution is found or no new nodes are reachable, B will contain all states reachable from the initial state (which is all states reachable from any state reachable from

the initial state). We thus know that no other state in A-B is reachable from any state in B.
Next, take any state in A-B and repeat the above process. Call the resulting set of states C. Analogously, we know that no state in A-C is reachable from any state in C. We now know that B and C are disjoint. Finally, check that A-C-B is the empty set. This shows that A is made of two disjoint sets B and C.

Sensorless Robot Problem

1. I tried out two different heuristics for this problem. First, I tried just using the size of the state space, so that states with fewer possible locations would be explored first. This heuristic is not optimistic - it might overestimate the cost of the path to the goal. For example if there are two adjacent 5 square columns of possible locations, the heuristic value is 10. However, the possible locations might be reduced to one after moving once to the right and up 4 times. I then tried summing the number of possible x locations -1 and the number of possible y locations -1 . This worked better in practice, giving shorter paths and visiting less nodes than the first heuristic for a variety of mazes. This heuristic is optimistic, because each movement can change either the x or y value by at most one, so that at most one x or y value will be removed from the possible locations with each movement. This heuristic gives the number of x and y values that must be removed from the state. Thus, it gives a lower bound on the number of actions required to reach the goal, so that the heuristic value is less than or equal to the true cost. Because it is optimistic, A* will not overlook the optimal solution when using this heuristic, as it might when using the first heuristic I tried out.