Tristan Bowler
CS4300
Project 1

3.1 Depth First Search (12 pts)

1. (10 pts) Code Implementation

2. (1 pt) Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?

No he does not. When I initially started this problem, I expected that pacman should illustrate all the tried and failed paths as well. I was (not so quickly) proven wrong. The path that needs to be returned is ONLY the direct path to the solution.

3. (1 pt) Is this a least cost solution? If not, think about what depth-first search is doing wrong.

This is not. In my case, tinyMaze returns with a cost of 10, whereas bfs returns with a cost of 8 on tinyMaze (which is the ideal cost). The issue is the order in which nodes are explored. As we saw with tree searches, dfs goes down the whole left side first, so it finds the leftmost solution, not the shortest path solution.

3.2 Breadth First Search (11 pts)

1. (10 pts) Code Implementation

2. (1 pt) Does BFS find a least cost solution? If so explain why ?

As noted above, it does for my tinyMaze. And I would say that it also does for medium maze, based on how drastically different my dfs and bfs results are. Bfs finds the shortest path because it methodically goes across the whole graph, expanding nodes at the same depth all at once, so it finds the shallowest solution, which is least cost.

3.3 Uniform Cost Search (11 pts)

1. (10 pts) Code Implementation

2. (1 pt) Specify the data structure used from the util.py for the uniform cost search

I used a PriorityQueue() with the priority being a function of the cost provided as state[2].

3.4 A* search (12 pts)

1. (10 pts) Code Implementation

2. (2 pts) What happens on openMaze for the various search strategies? Describe your answer in terms of the solution you get for A* and Uniform cost search.

In my case, A* and UCS have the same result. This is because A* is running with a null heuristic, so it is simply the cost of the path effecting it, so it is exactly the same as UCS. BFS also has the same behavior as A* and UCS, because the latter two are BFS based searches.

DFS however zig zags down the entire screen before reaching the goal, in its typical non-optimal way.

3.5 Finding All the Corners (12 pts)

1. (10 pts) Code Implementation

2. (2 pts) Describe in few words/ lines the state representation you chose or how you solved the problem of finding all corners?

I keep track of the unvisited corners with the positions on the frontier in the form of a list. The goal state is reached when all of the corners have been eaten, or in other words when an unvisited list of a state has a length of zero. Finally, the successors function is simple, if a direction does not result in hitting a wall, then it is added to successors.

3.6 Corners Problem: Heuristic (11 pts)

1. (10pts) Code Implementation

2. (1 pt) Describe the heuristic you had used for the implementation.

I first guessed that it would be some function of the distances for choosing a particular path, however all of my attempts at this ended up inconsistent.
I ended up with some crazy amalgamation of the closest node and the second closest node with the logic here being that if you always go to the closest node, you may end up shooting yourself in the foot for getting to the second one in a reasonable cost.

3.7 Eating All Dots (15 pts)

1. (13 pts) Code Implementation

2. (2 pts) Describe the heuristic you had used for the FoodSearchProblem.

This is simply finding the farthest food source in maze distance. Manhattan distance was unideal and minimums were un ideal.

3.8 Suboptimal Search (11 pts)

1. (10 pts) Code Implementation

2. (1 pt) Explain why the ClosestDotSearchAgent won't always find the shortest possible path through the maze.

Because it is a greedy algorithm that takes the shortest path to the nearest food. This stops it from having the foresight to come up with the shortest overall path in the process.

4 Self Analysis (5 pts)

1. What was the hardest part of the assignment for you?

Figuring out what it was that I was supposed to be doing and how the code based worked.

2. What was the easiest part of the assignment for you?

Part 7, which was incidentally supposed to be the hardest. I totally lucked out with the heuristic since I tried finding the shortest and farthest Manhattan and then maze distances first.

3. What problem(s) helped further your understanding of the course material?

It had been a while since I looked at search algorithms, so that was helpful, but I hated that it took me so long to figure out how to represent path lists on the frontier. I struggled with dfs for two days before figuring something out.

4. Did you feel any problems were tedious and not helpful to your understanding of the material?

The heuristic problems, I honestly felt like they were more guess work for me than actual problem solving. It would have been helpful to the assignment before this be matching so we could figure out the good heuristics on paper first.

5. What other feedback do you have about this homework?

The documentation being so sparse led to me spending a lot of time beating my head against the code. Honestly, I would have appreciated clearer problem descriptions and documentation because I spend all day at work pouring through messy or undocumented code. When completing an assignment, I want to be able to understand exactly what it is that I am expected to do quickly, so I can learn the objective of the assignment before I end up frustrated and desperate.

Also, it would have been nice if we had been told EXACTLY which python version to use. I wasted my first day on this assignment using python 3.8.1, which gave errors in the grading packages and did not give me the useful failure messages I'm supposed to get. If I hadn't had an internship with python where we had gone through re-versioning, I probably would not have thought to downgrade to 3.7.6.