Group Members:

Abhishek Ayachit : 862188073

Shashank Dahiya : 862170706

**Week 4:**

List of Files:

search.py : Implementation of DFS, BFS, UCS and A\* search

We implemented the DFS and BFS using stack and queue respectively. For the uniform cost search algorithm, we have implemented the Priority Queue which stores the nodes.For aStarSearch we follow the same process as of uniform cost search except that we add the heuristics value to the cost.

Q1. The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?

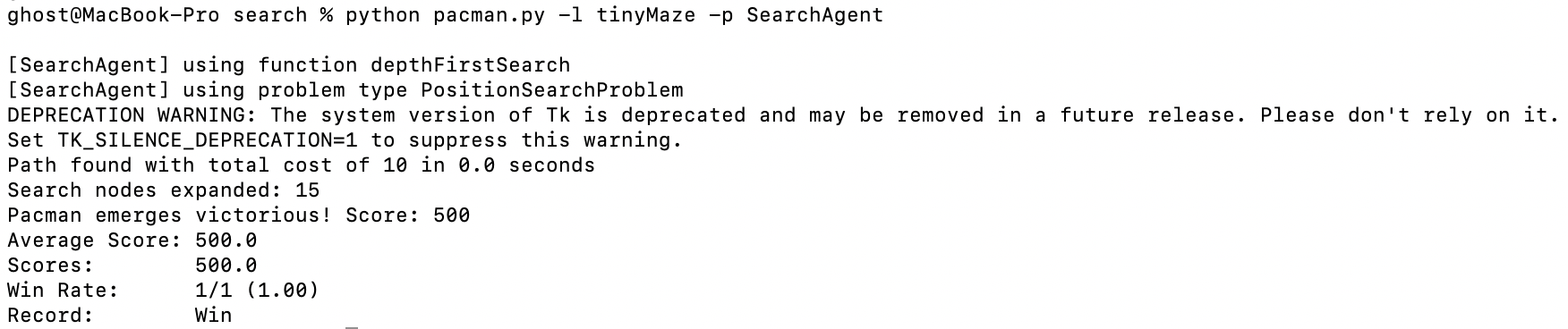
Answer: The exploration order is what we had expected. The Pacman doesn’t actually go to all the explored squares but only the squares which are on the path to the goal.

Found the goal with the cost of 130.

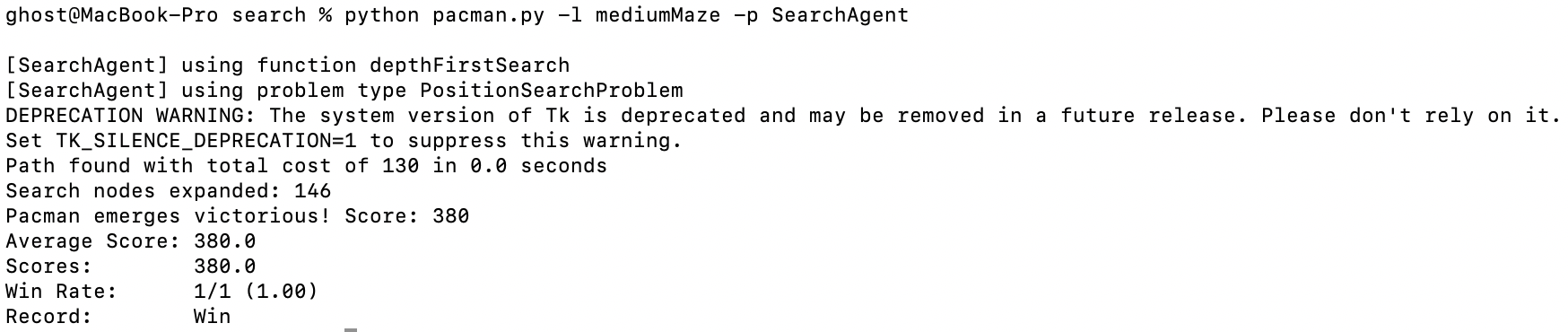
Q1.

Output for Commands:

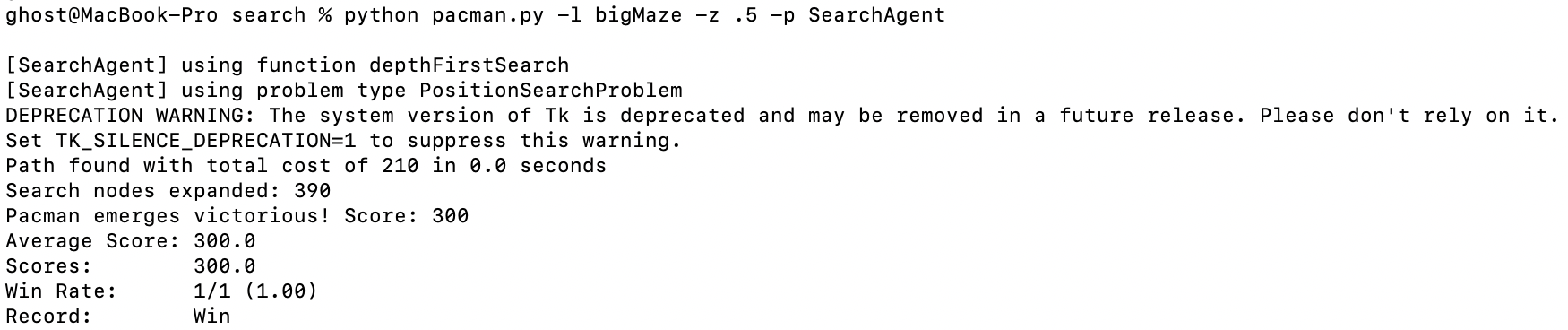
* python pacman.py -l tinyMaze -p SearchAgent



* python pacman.py -l mediumMaze -p SearchAgent



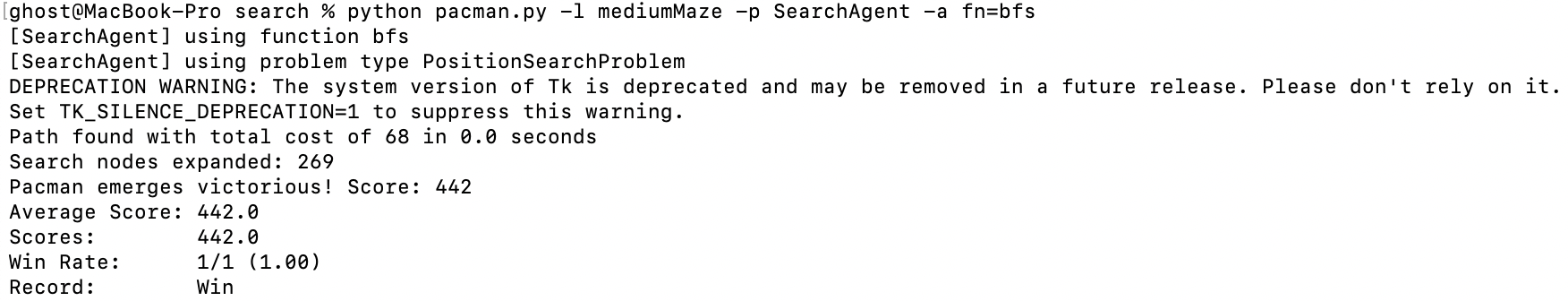
* python pacman.py -l bigMaze -z .5 -p SearchAgent



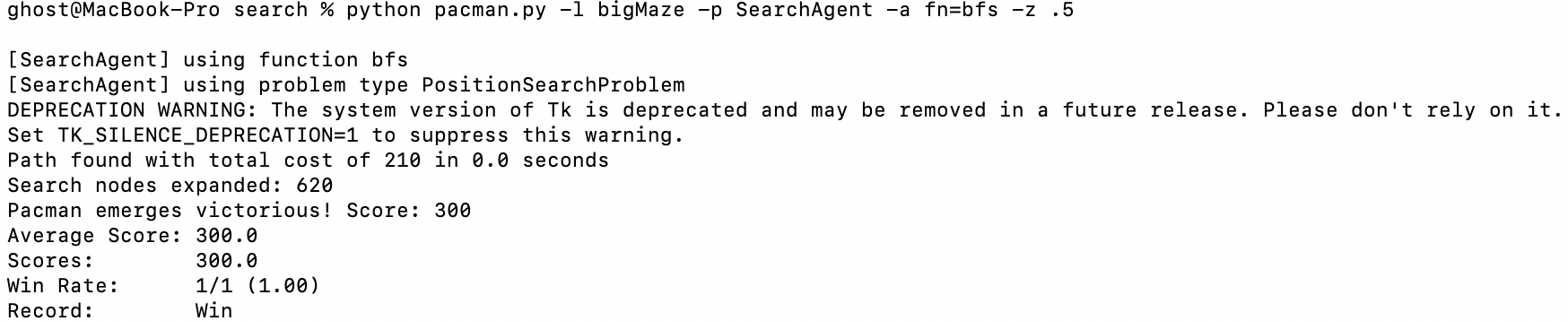
Q2.

Output for Commands:

* python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs



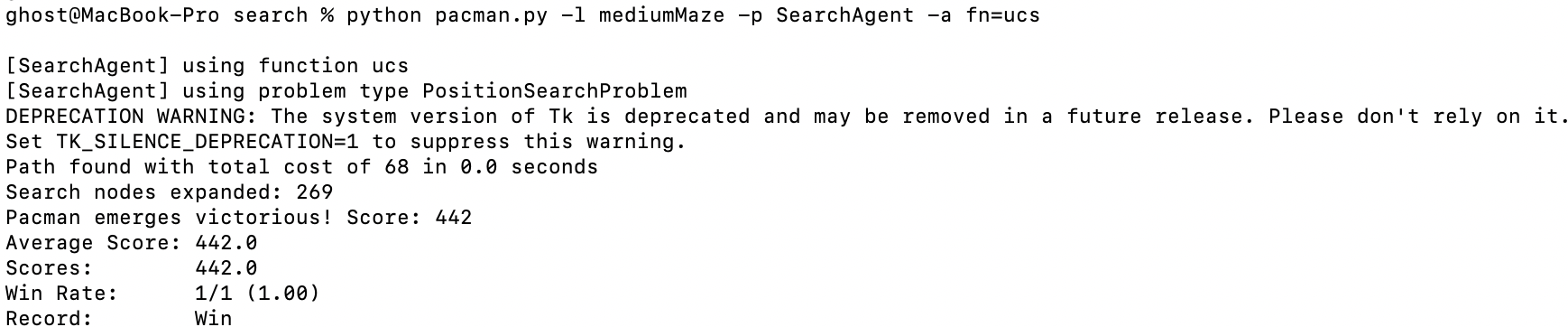
* python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

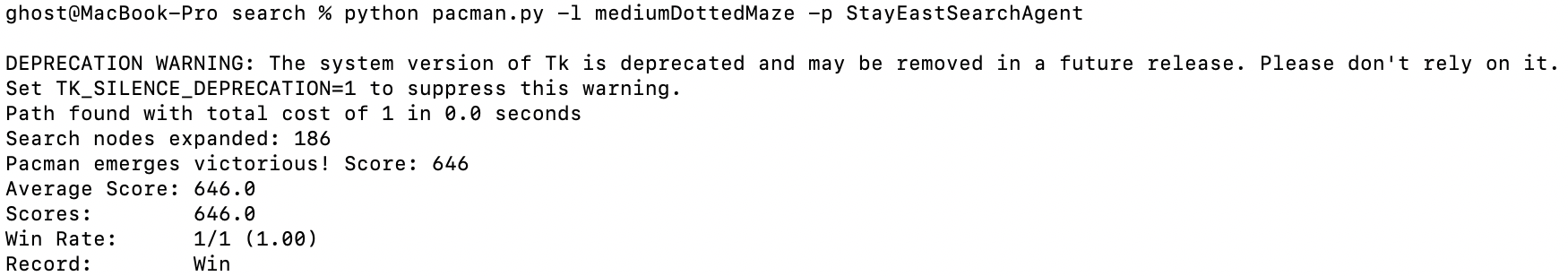


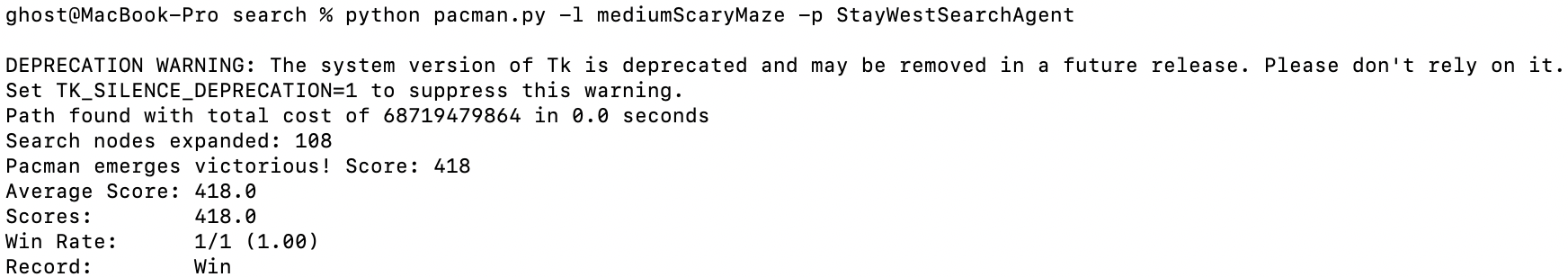
Q3.

Output for Commands:

* python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

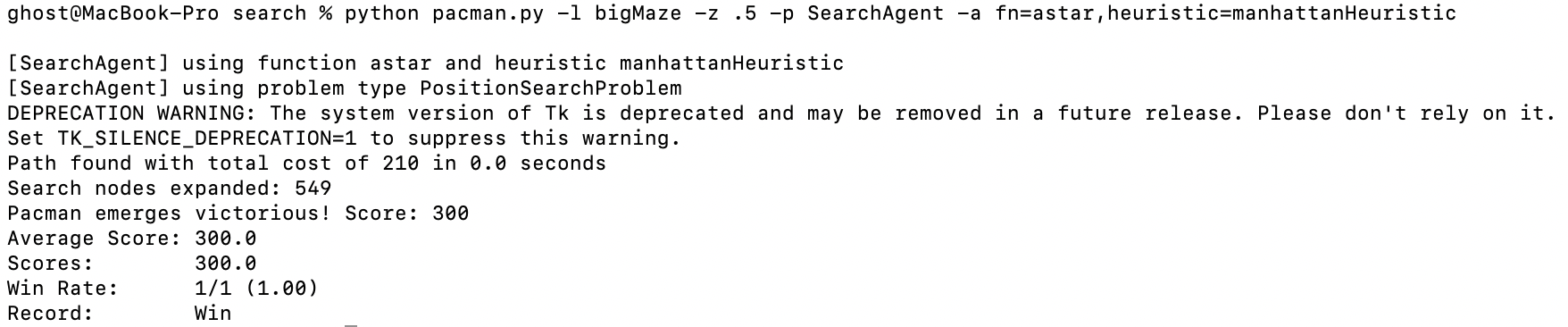


* python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
* python pacman.py -l mediumScaryMaze -p StayWestSearchAgent



Q4.

Output for Commands:

* python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

**Week 5:**

List of Files:

searchAgent.py : Code for the corner problem and cornerHeuristic

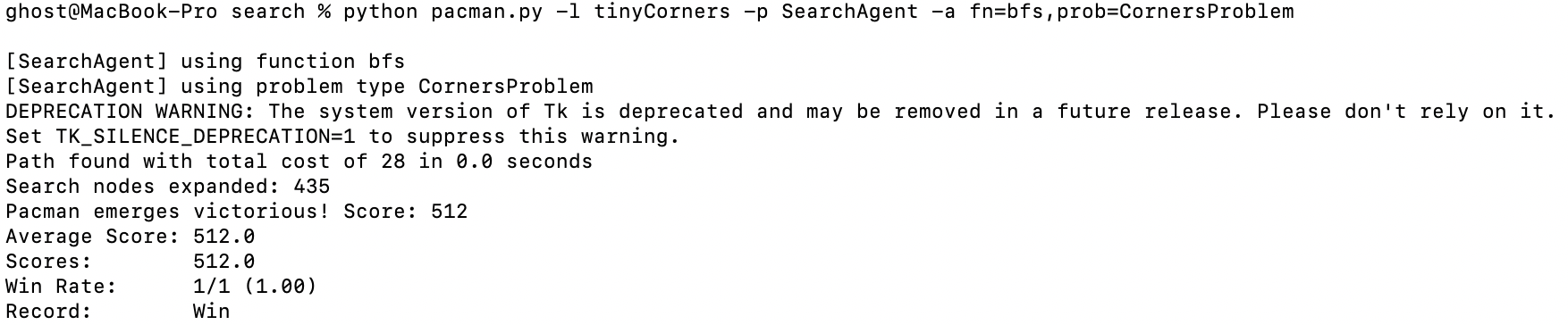
Q5: Finding All the Corners

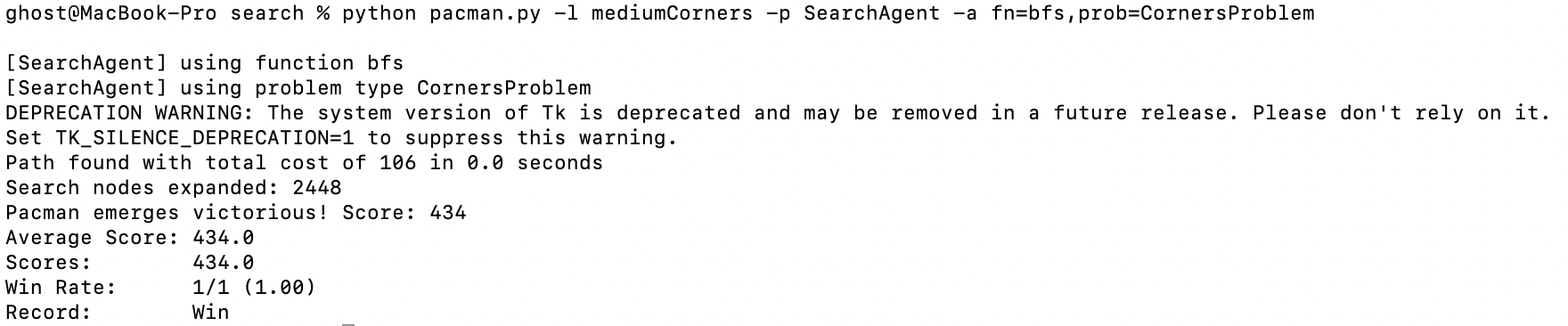
We get only the start position from GameState, i.e self.startingPosition. We are maintaining a visitedList[] for all corners. At the beginning all the visited corners are empty. We have taken two variables where head[0] refers to current position and head[1] refers to visited corner.

When we run our game we will check the successor nodes and if the node is a corner and it has not been visited then it is added to the list of visited corners. The goal of the corners problem is to visit every corner of the maze and hence it is checked if the number of visited corners is equal to the overall sum of the numbers of corners in the maze.

Output for Commands:

* python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem



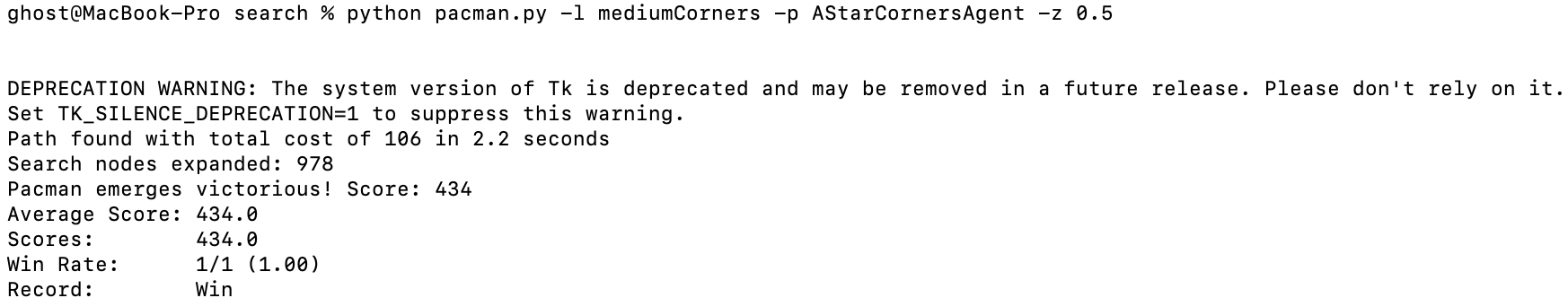
* python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

Q6.

This is a Corners Problem: Heuristic. According to the question, we have to implement a non- trivial heuristic to search for our A\* search to find the corners of the map. We have to check if he UCS and A\* ever return paths of different lengths, then our heuristic will be inconsistent. So the goal state for our problem is that we have find the most distant unvisited corners. For the heuristic we have calculated the pac-man maze distance between the most distant unvisited corner and the current position. We are taking the help of breadth first search for the calculation of the distance to the most distant unlisted corner(as it is mentioned q6 uses q4). Then for every corner in the set of unvisited corners, the maze distances is calculated towards the current position and the largest value among them would be the most distant corner.

Output for Commands:

* python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5



**Week 6:**

List of Files:

search.py: Implementation of DFS, BFS, UCS and A\* search

searchAgent.py : Code for theFoodSearchProblem and findPathToClosestDot

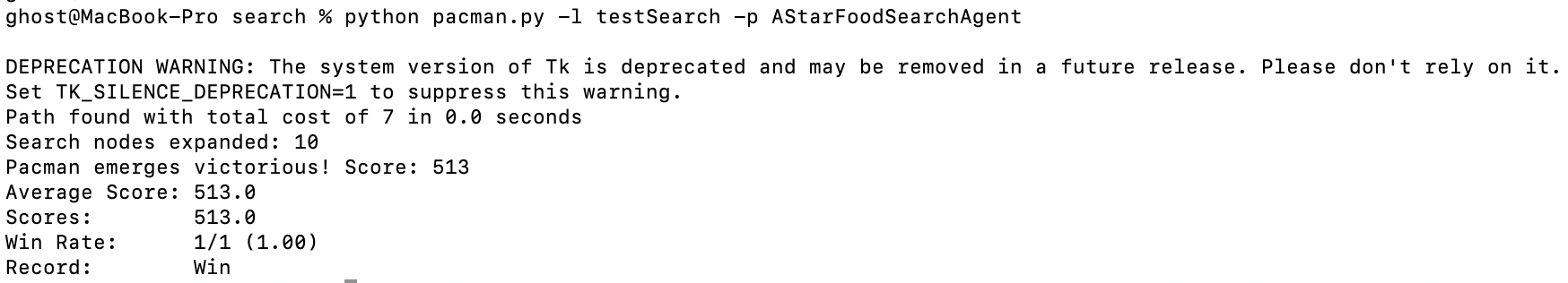
For problem 7, here we calculate the heuristic as the node distance between the current position and the food position at the greatest distance. We are taking the help of breadth first search for the calculation of the distance to the most distant node for the food position. We are calculating the distance for each food position to the current position. The further the distance, then this would be the farthest food point.

For problem 8, here we are greedily finding the closest path such that the agent always eats the closest dot. We have added the missing function that finds a path to the closest dot. Below is the result for both q7 and q8.

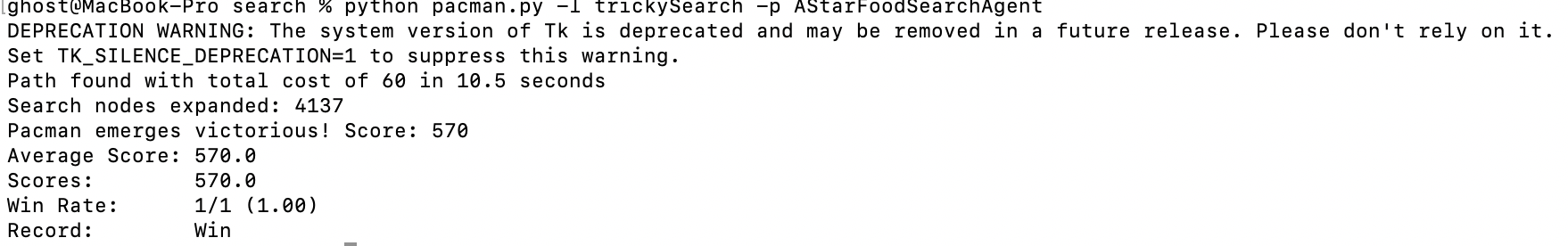
Q7.

Output for commands:

* python pacman.py -l testSearch -p AStarFoodSearchAgent



* python pacman.py -l trickySearch -p AStarFoodSearchAgent



Q8.

Outputs for commands:

* python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5