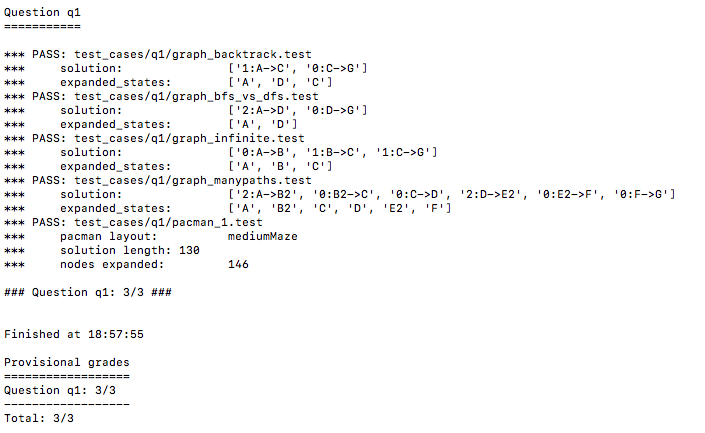
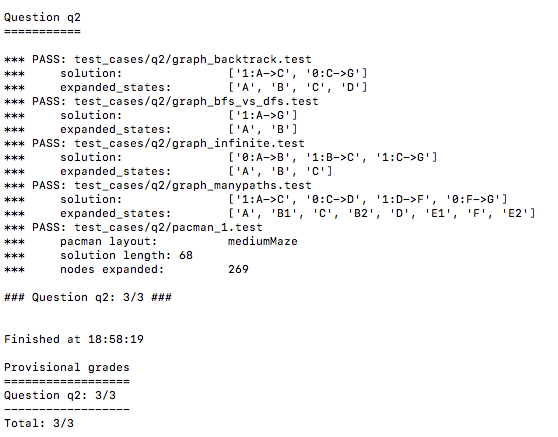
Week 3:

Playing with the interface of PacMan and working out the implementation of Breadth-First Search and Depth-First Search.

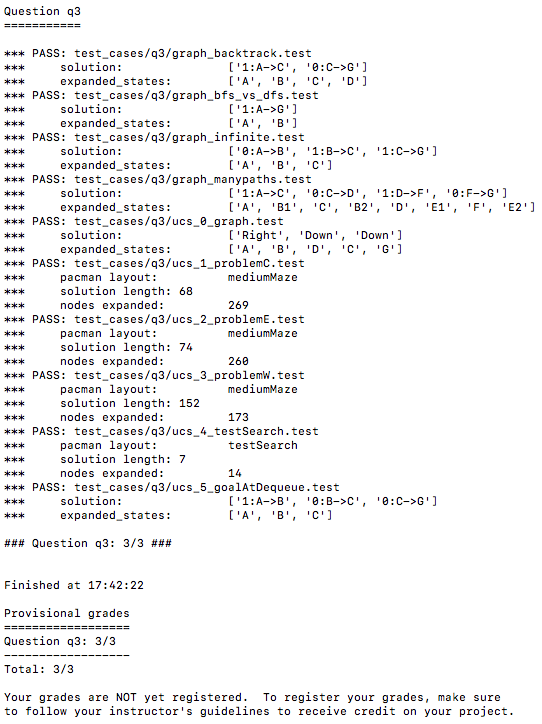




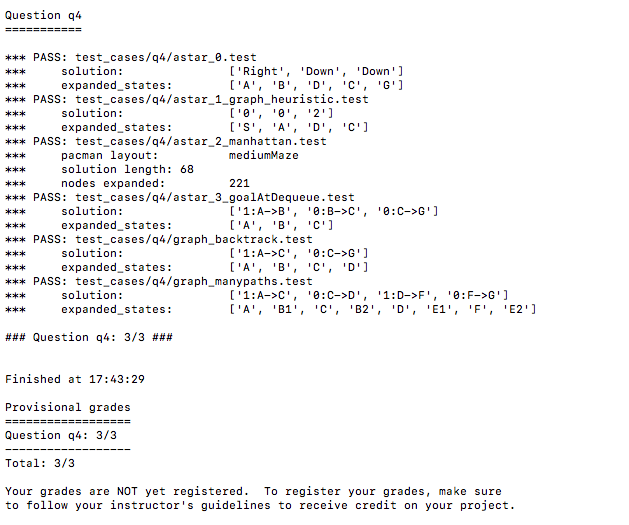
-------------------------------------------------------------------------------------------------------------------------------

Week 4:

For the uniform cost search algorithm, we have implemented the Priority Queue which stores the nodes. We then keep track of the nodes while visiting the grid. “startState” is the current PacMan position and the variable cost stores all the costs till the current position. The variable “action” contains the list of all actions which are required to reach from initial position to the startState position. We will continue till the priority queue is empty. If the goalState is reached then we should return all the list of actions or else we will have to continue pushing all the remaining nodes which are not yet visited in priority queue. Here is the auto grader result for q3.



For aStarSearch we follow the same process as of uniform cost search except that we add the heuristics value to the cost. Here is the auto-grader result for q4.



Week 5:

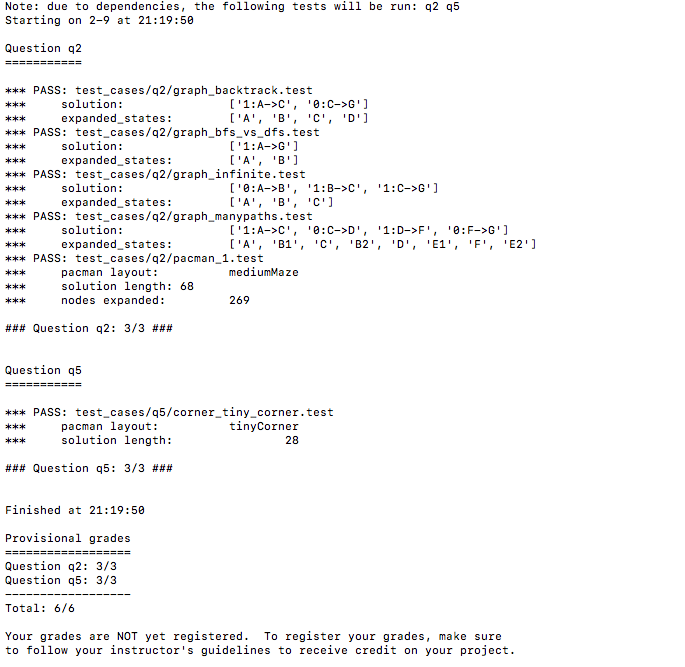
### Q5: A\* : Finding All the Corners

\*\*Problem:\*\* According to the question our new search problem is to find the shortest path through the maze that touches all four corners irrespective of whether the maze actually has food or not.

\*\*State Representation:\*\* 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.

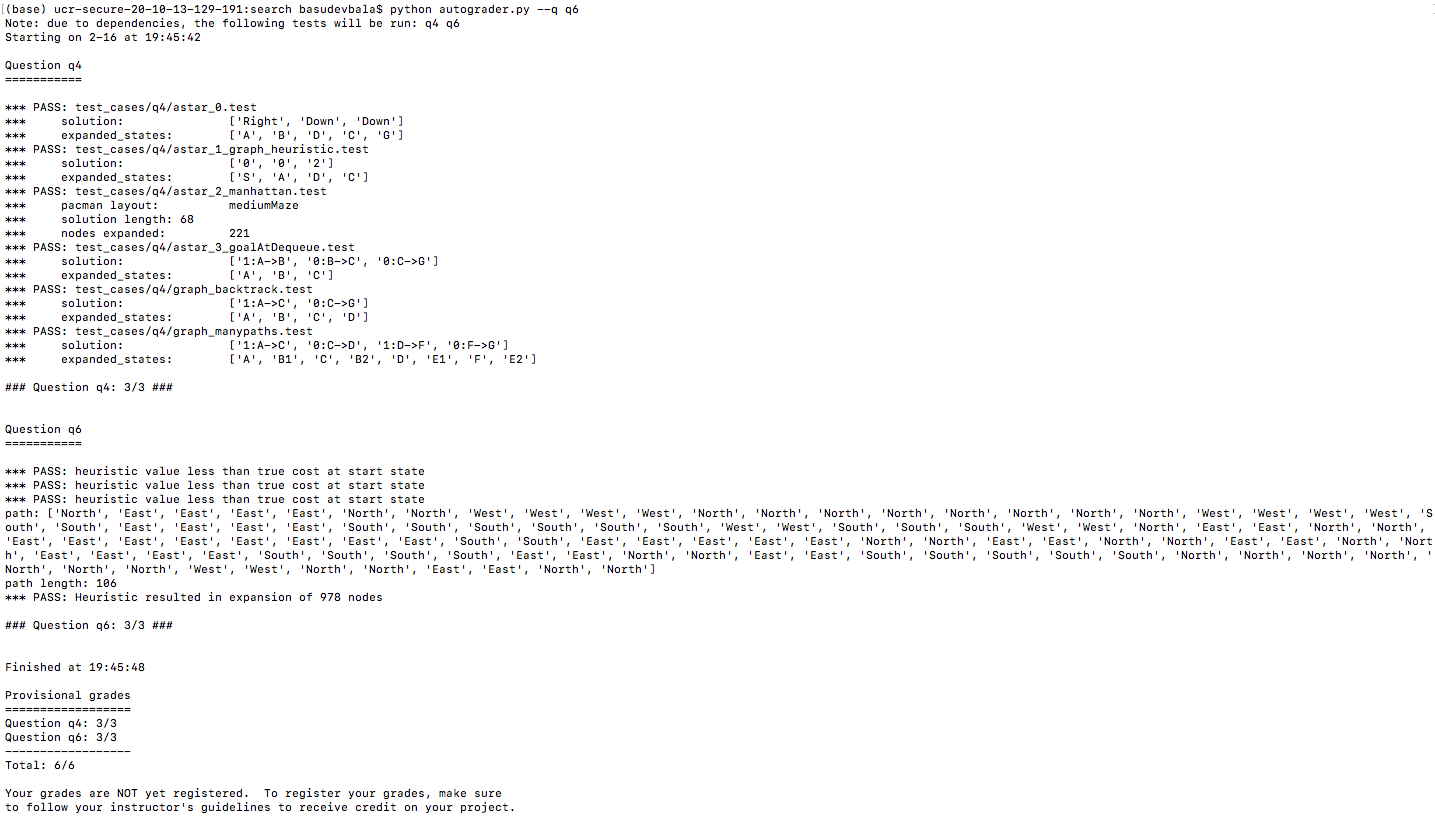
When we run our game we will check the successor nodes and if the node is a corner and it has not been visited then it is added to the list of visited corners. The goal of the corners problem is to visit every corner of the maze and hence it is checked if the number of visited corners is equal to the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

\*\*Output:\*\* 'True' state will be returned only if the goal has been reached, otherwise 'False' state will be returned.



Week 6:

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 the 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.

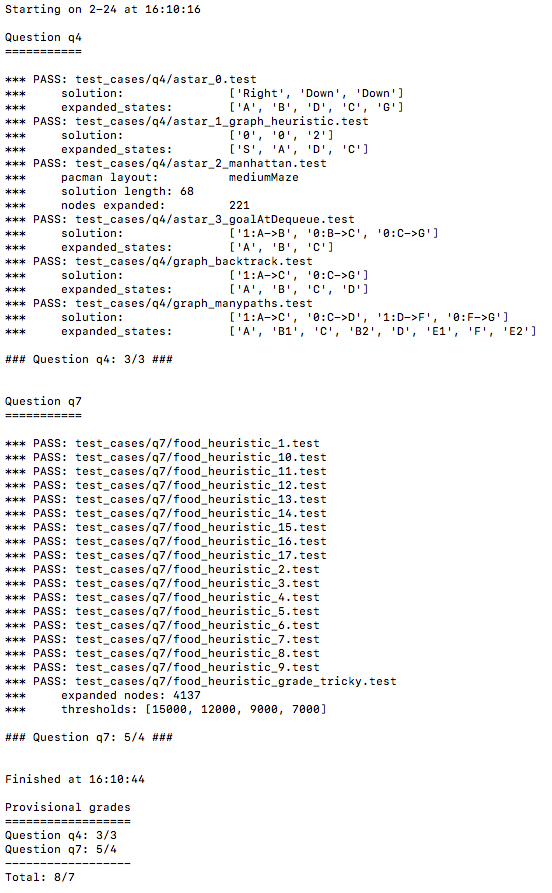


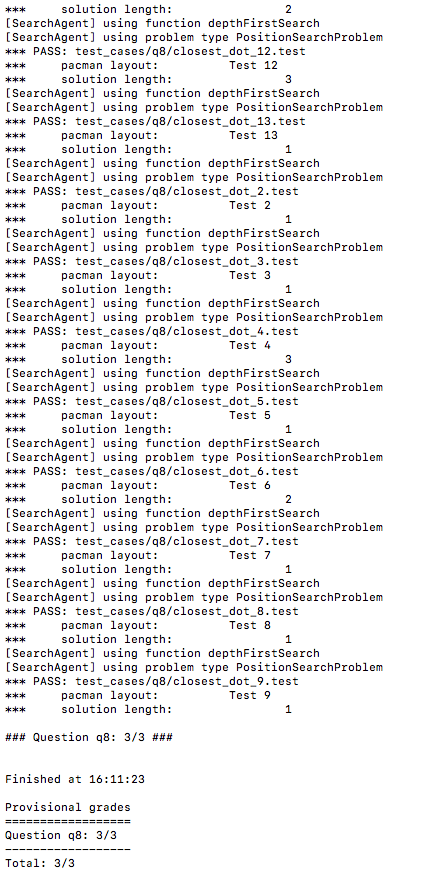
Week 7:

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.





Team Members:

Anjali Ramchandani - 862188655

Gowtham Tumati - 862186477

Sudip Bala - 862188812