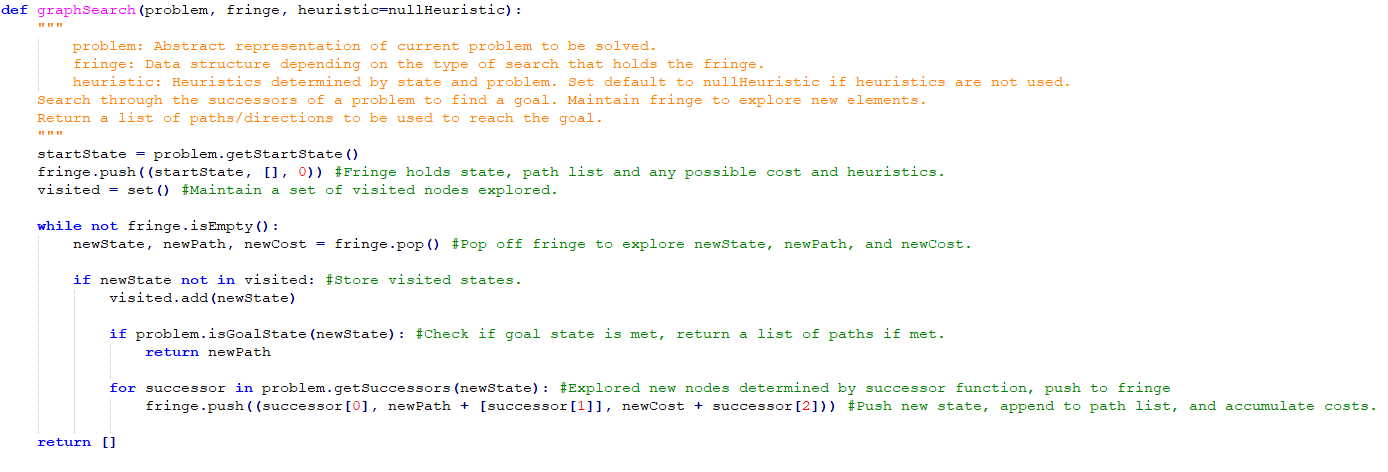
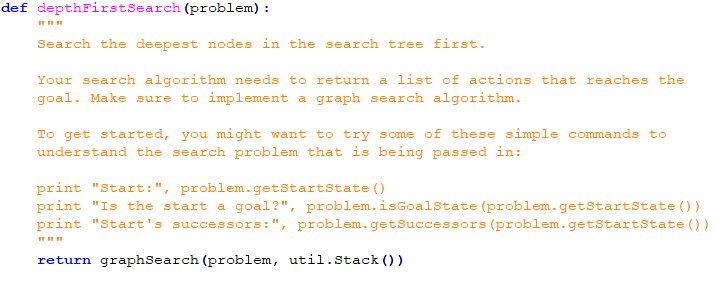
**Documentation for A.I Pacman Project**

**graphSearch():**



**Question 1.**

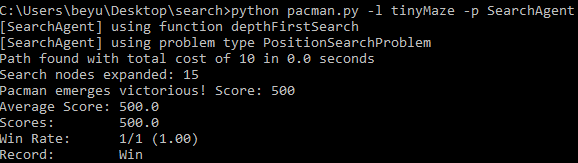


Depth First searches adjacent nodes first until reaching the bottom of a branch before it starts backtracking again. Depth First Search uses a LIFO structure, hence the use of a Stack to pop from the fringe of exploration elements.

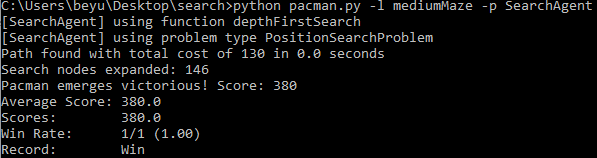
We store the fringe in a Stack data structure and return graphSearch(problem, util.Stack) for Depth First Search result. The exploration order is expected because already visited squares in DFS are not to be visited again. This is not a least cost solution. DFS does not take into account the cost of traversing through a node, rather it traverses through adjacent nodes first.

The exploration order Pacman takes is expected. Pacman does not actually go through all the explored squares on his way to the goal. Pacman only goes through the paths that match the list of paths returned by graphSearch. The overlay of states explored as well as the order are displayed only.

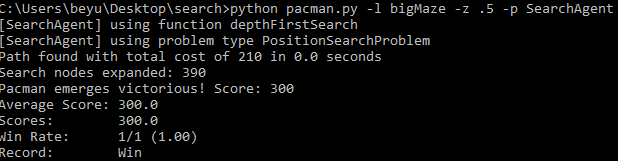
python pacman.py -l tinyMaze -p SearchAgent



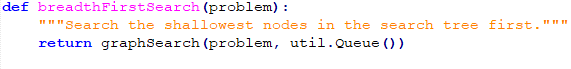
python pacman.py -l mediumMaze -p SearchAgent



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



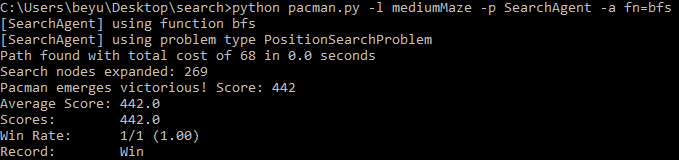
**Question 2.**



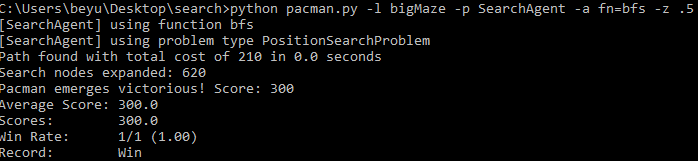
Breadth First Search is an algorithm that prioritizes searching for neighbor nodes first before moving to next level neighbors. We use a Queue structure to maintain fringe exploration.

Yes, my implementation finds a least cost solution. We return structuregraphSearch(problem, util.Queue) for Breadth First Search result.

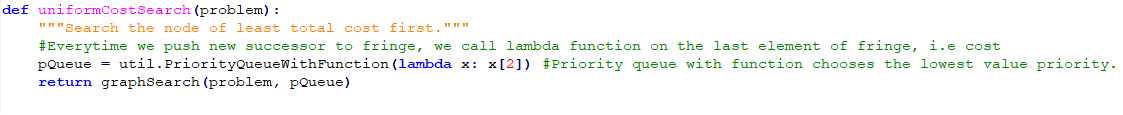
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs



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

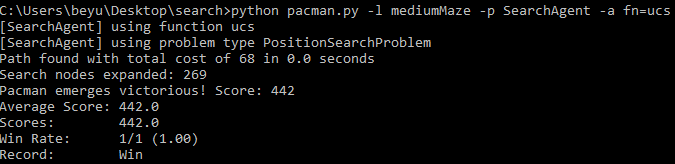


**Question 3.**

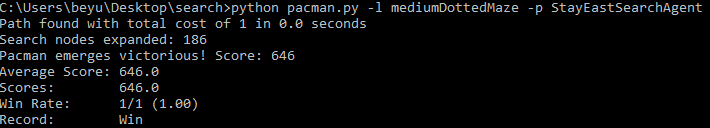


Uniform Cost Search is an algorithm that searches based on the most optimal cost. The minimum cumulative cost has priority, thus we use a priority queue.

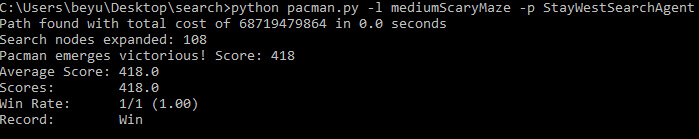
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs



python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

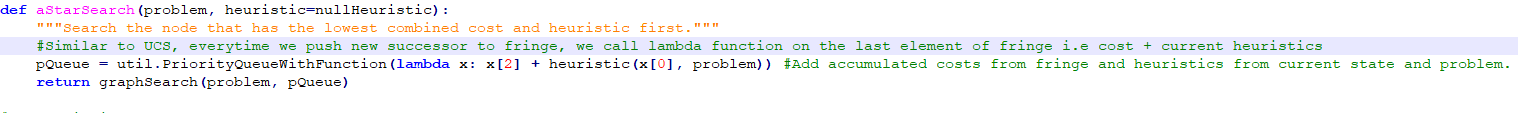


python pacman.py -l mediumScaryMaze -p StayWestSearchAgent



Very low path cost for StayEast, and very high for StayWest.

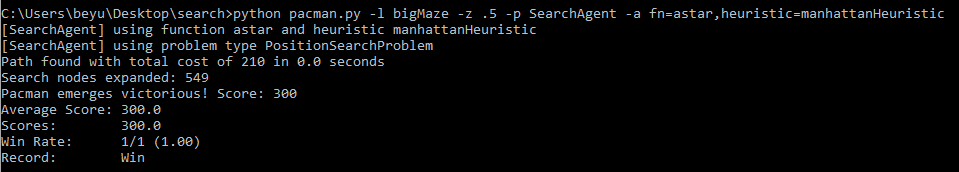
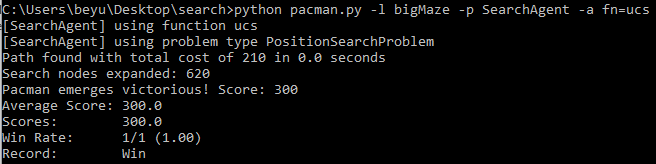
**Question 4.**



A\* search is an algorithm that is a combination of heuristics from greedy search and cost from uniform cost search. Like UCS it uses a Priority Queue to maintain the fringe.

A\* Result:

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

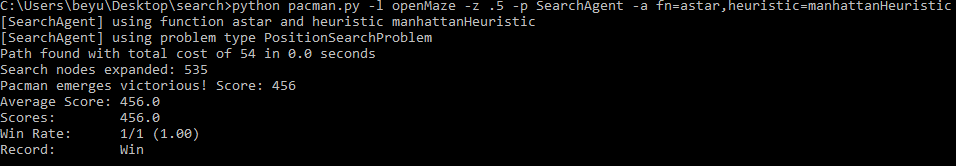


A\* finds the optimal solution with 549 expanded search nodes as opposed to the 620 expanded search nodes from the uniform cost algorithm.

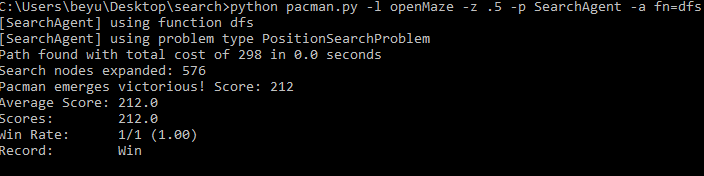
python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

For openMaze using the various search strategies, A\* expands the least amount of search nodes. A\*, UCS, and BFS find the solution with the same amount of cost, while the path solution for DFS takes a larger cost.

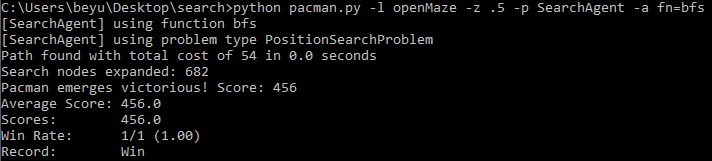
A\*: python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic



DFS: python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=dfs



BFS: python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=bfs



UCS: python pacman.py -l openMaze -z .5 -p SearchAgent -a fn=ufs

