Student Name: **A Z Hasnain Kabir**
Student ID: **200042102**

CSE 4618
Artificial Intelligence Lab
Lab 1

Introduction

In this lab, we demonstrate the applications of uninformed search particularly in the case of Pacman and its path through the maze world.

Question 1

We need make pacman find a fixed dot through the Depth First Search algorithm

Analysis of the problem

We need to implement an algorithm that lets pacman navigate through its maze world and find a fixed dot. The algorithm must be Depth First Search. According to the taskbook, *searchAgents.py* will contain all the necessary code to execute the path step by step. All we need to do is try to implement Depth First Search as a search algorithm inside *search.py*.

Explanation of the solution

Since this is the first problem, it is provided by our teacher. Inside the solution, our course teacher makes a function called depthFirstSearch which uses a stack as a fringe, the start state is taken as a root node. A list called *closed* is used to keep track of the nodes visited through DFS. We evaluate our task using autograder.

Interesting findings

When applying this search algorithm in medium maze, the total cost it takes is 130 which is almost double the cost when Breadth First Search algorithm is applied.

Problems faced

Running autograder for the first time when trying to evaluate code gives error in Python as autograder requires Python version 3.6 to work.

Solution of the problem

Since there is already python 3.6 installed as a virtual environment inside my conda environment, all I needed to do was activate the environment corresponding to Python 3.6.

Question 2

We need make pacman find a fixed dot through the Breadth First Search algorithm

Analysis of the problem

Previously, we implemented Depth First Search algorithm inside *search.py* file. Now we need to implement Breadth First Search algorithm and execute the steps required to make pacman find a dot.

Explanation of the solution

As the solution is BFS, we needed a Queue. An already implemented queue data structure is given to us in the *utils.py* file and we use that as a fringe. The start state is stored as our root node. Inside a loop we continuously check if there are no more ways to go and continue to push new nodes inside our fringe. We keep track of the visited nodes through a list called closed. Eventually, the algorithm will find a solution.

Interesting findings

When applying this search algorithm in medium maze, the total cost it takes is 68 which is almost half than that of Depth First Search algorithm.

Problems faced

No major problems came up as the implementation of this algorithm is mostly similar to that of DFS algorithm with a minor change of fringe data structure.

Question 3

We need make pacman find a fixed dot through the Uniform Cost Search algorithm.

Analysis of the problem

We have implemented BFS and DFS algorithm to find a fixed dot for pacman. For this question, we need to implement Uniform Cost Search and find the node of the least total cost. Different agents are provided for our problem which are all UCS agents but differ in case of cost function. For instance, StayEastSearchAgent and StayWestSearchAgent.

Explanation of the solution

At first, we selected PriorityQueue from *utils.py* as our fringe. Unlike the previous ones, where we defined root node as only the start state, this time we introduce cost to our root node which is initially 0. Iteration is done until we have reached our goal state or our fringe is empty. Each time our current cost is updated along with the current node. We use *Set()* to store the already visited nodes.

Interesting findings

We again apply this algorithm in medium maze where the total cost taken is 68 which is the same as what we obtained when applying BFS.

Problems faced

As instructed in the lab class, I implemented this algorithm according to instructions without facing any issues.