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

CSE 4618
Artificial Intelligence Lab
Lab 2

Introduction

In this lab, we want to demonstrate the applications of informed search through pacman game and its journey

Question 1

We need to demonstrate $A*$ search and make pacman find its path through a maze.

Analysis of the problem

We are to implement an algorithm called $A*$ search in an empty function *aStarSearch* in *Search.py*. We will test our implementation using an already defined heuristics function called *manhattanHeuristic*.

Explanation of the solution

In the steps, there is a fringe which is a priority queue. This is used because we need to ensure that nodes with the lowest combined cost and heuristic value are explored first. The initial state has a cost of 0 to signify the starting node. We are continuously checking if our current state is a goal state inside our loop to indicate the end of our iterations.

Interesting findings

$A*$ search is optimal if the heuristic used is admissible and consistent. This is because it never overestimates the cost of reaching a goal. So finding the shortest path is guaranteed depending on the heuristic function.

Question 2

We are to find the shortest path through a maze that finds all the corners.

Analysis of the problem

In this search problem, there will be four dots within a maze. We are to find the shortest path through the maze that touches all four corners. For this, we need to use already provided *breadthFirstSearch* inside *Search.py*.

Explanation of the solution

The solution was already provided by course teacher. The function *getSuccessors* is implemented. In order to explore possible paths in a maze, it will iterate through four directions and check if moving in that direction hits a wall or not. If not, an action will be applied and a successor state will be generated. If we hit a wall, it will not generate a successor as we can't move through a wall. If a corner is visited, the *nextVisitedCorner* will be updated accordingly. Each cost is set to 1 inside code indicating that moving from one state to another has an equal cost.

Question 3

We are to implement a heuristic for cornersProblem

Analysis of the problem

In this problem, we are to implement the function *cornersHeuristic* for cornersProblem where pacman is finding corner dots efficiently.

Explanation of the solution

I have implemented Euclidean distance as my heuristic function to solve this problem. In different scenarios for instance, when there are unvisited corners, the heuristics function calculates Euclidean distance from current state and each unvisited corner. The maximum distance will be selected as reaching any of the unvisited corners would require at least that much movement. The heuristic function will remain admissible and encourage the algorithm to visit the unexplored nodes first. When the corners are done being visited, heuristic function will return 0 indicating goal state.

Interesting findings

If we use minimum instead of maximum when estimating distance, it may not guarantee admissibility. If the closest unvisited corner is obstructed by walls, the actual cost may be higher than estimated by the heuristic. It leads to suboptimal solutions.

Question 4

Eat the Pacman foods in as few steps as possible.

Analysis of the problem

We need to implement *foodHeuristic* inside *searchAgents.py* using a consistent heuristic. Using this heuristic, pacman will eat the dots with as few steps as possible.

Explanation of the solution

By selecting the maximum distance, the heuristic remains admissible as it never underestimates the actual cost. It is based on the assumption that pacman needs to travel at least the distance to the farthest food pellet to eat all the foods.

Interesting findings

In the case where the food pellets are located close together, the heuristic may overestimate the actual number of steps needed. This may lead to suboptimal behavior.

Question 5

In this question, we are to write an agent that finds the closest dots.

Analysis of the problem

We are to implement the function *findPathToClosestDot* inside *searchAgents.py*. The function calls *AnyFoodSearchProblem* class which is also missing a goal test function.

Explanation of the solution

The function *findPathToClosestDot* aims to find a path from the Pacman's current position to the closest dot. It uses BFS to search for the shortest path to the closest dot. This is a good choice because to find the shortest path, it explores all the nodes at each level before moving to the next ensuring that the first goal is the closest one.

Interesting findings

BFS can be inefficient for large spaces as it explores all the neighboring nodes.