Project 1 Pacman

Papanikolaou George

November 14, 2023

1 Q1 - Q4

I used the code given during the second lecture by translating into python and using the correct frontier (stack,PQ,Q,..)

https://cgi.di.uoa.gr/~ys02/frontistiria2023/Search.pdf page 28

2 Q5

2.1 getStartState

I initialize the starting state of the problem and give it the form: tuple(tuple(x,y),tuple(visited corners) where the tuple(x,y) named self.startingPosition is the position of pacman and tuple(visited corners) named visited_c are the visited corners

2.2 isGoalState

I check if the visited_c has length less than 4

2.3 getSuccessors

I create the successors of the position the pacman is by checking the legal moves from that position and create the following states ((nextx,nexty),visited_c) and if the next state is a corner I update the visited corners inside the created state.

(State is tuple of the position and the tuple visited , in my code the state is the tuple of position and the visited tuple and also contains the action to that state/position and the cost to go there)

3 Q6

For the heuristic I take 4 cases :

(In all cases i assume that there are no walls in order to simplify the problem and ensure its admissibility by using the Manhattan distance)

First case(0 visited corners):
I find the closest corner to pacman, I calculate the cost to that corner using the Manhattan distance function and added to the estimated cost. Then I assume that pacman has gone to that corner and calculate the cost to the next closest corner and so on until the (simulated) pacman has gone to all corners and I add the cost from one corner to the other. So the heuristic is the cost to the closest corner from each corner plus the cost to go to the first/closest corner from the starting position of the pacman.
The heuristic is consistent because it only moves by one node/square and uses the Manhattan distance which only affected by the position of pacman which can only changes by one.
It is admissible because first I do not acknowledge the existence of walls and because the shortest way to go through the 4 corners of a rectangle is to always go to the closest corner from each corner

Second case(1 visited corner):
In this case instead of going to the closest corner (because it might not be admissible in one case, if

pacman is between two corners, one edge and the middle one, and closer to the middle one then the heuristic we overestimate the needed steps) I take all the paths that pacman can take from one corner to the others creating all the possible ways that pacman can visit the 3 corners (using the Manhattan distance) and i add the distance to the first corner of each case. So i create 6 paths for pacman to follow and I take the cost of the sortest one. (minimum spanning tree between the 3 corners + the cost to go to the first/edge corner)
It is consistent for the same reason as the first case.
It is admissible because it takes the minimum spanning and I do not acknowledge the existence of walls.

Third and Fourth case (2 and 3 visited corners):
I calculate the distance to the closest corner and if another one exist I just add the cost to go from the closest corner to that further corner.
It is consistent for the same reason as the first case.
It is admissible because the fastest way to go to two point its to go to the closest first and then to the other one.

4 Q7

The heuristic assumes that the all the foods are on the path to the furthest food so it only uses the real cost (using maze distance) to go to the furthest food and assumes that in the way to that food pacman will it all the other ones.
The heuristic is consistent because it only moves by one node/square at each time and uses the position of pacman and the real shortest path to each food which can not change more than one when moving one node/square away or toward that food.
It is admissible because in the best case scenario the foods would be in the path to the furthest one so in all the other cases it will undervalue the real cost to go to all foods.
(Also in order for the program to run faster I save the cost that pacman needs from one position to a specific food so when pacman goes again to the same node, even if he has eaten a food in his return meaning the state has change all the other distances are saved and can be reused.)

5 Q8

5.1 isGoalState

I check if pacman is in the position where a food is.

5.2 findPathToClosestDot

It does breadth first search to find the dot closest to pacman.