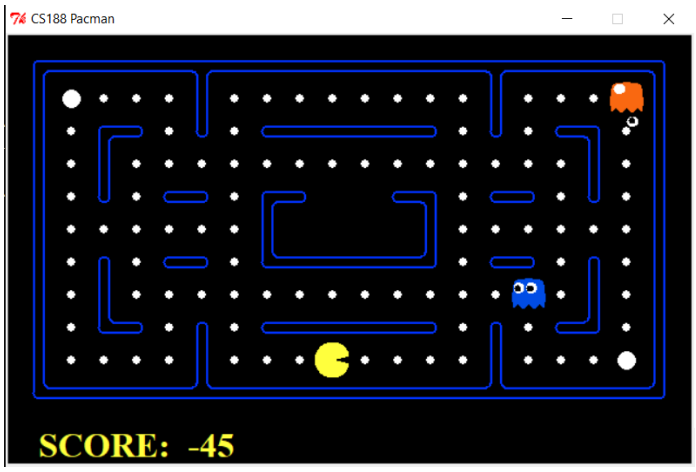
CS2180 – Artificial Intelligence

Lab 2 – Pacman Search

Project Report

Sneha Bhattacharjee, 112001056



Abstract:

The aim of this project is to design an intelligent Pacman agent that moves optimally through the Pacman world and eats all the food dots in as few steps as possible. For this part of the project the ghosts are neglected. The first section involves implementing graph search algorithms such as DFS, BFS, UCS, A* search. The second section involves writing a state space representation for the corners problem and designing and implementing consistent heuristics for the corners problem and the food search problem.

Generic Search:

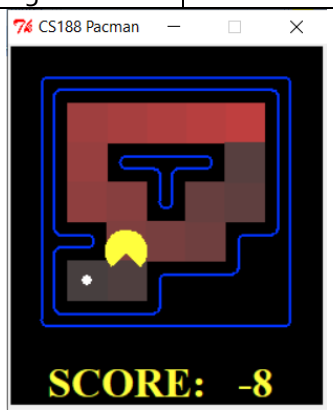
All the algorithms here, DFS, BFS, UCS, A* search share the same structure; they expand nodes in the frontier list until the agent reaches the goal state or there are no more nodes to expand, in which case there exists no solution. They differ in their implementation on the parameters they choose to expand nodes from the frontier list.

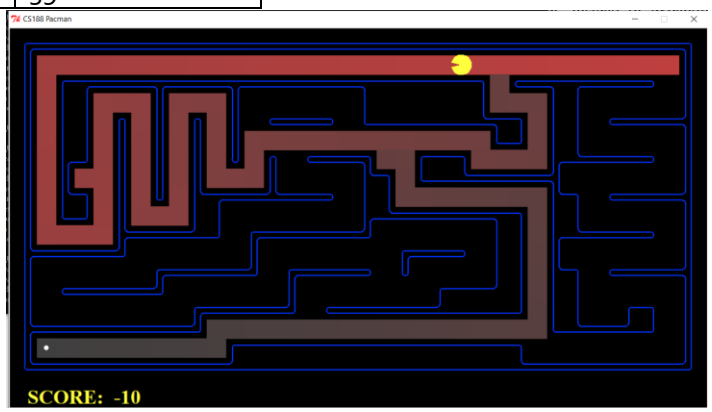
Depth First Search

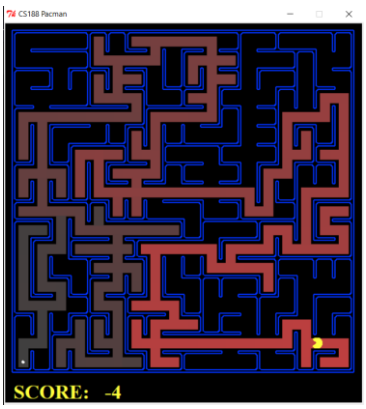
DFS expands the deepest nodes in the tree first. It is not optimal or complete. DFS uses a LIFO data structure, the stack.

Maze	Total Cost	Nodes expanded
tinyMaze	10	15
mediumMaze	130	146
bigMaze	210	390

One can observer that DFS expands lots of unnecessary nodes which doesn't lead it to the goal. Shown below are the nodes expanded in tinyMaze, meduimMaze and bigMaze respectively.







DFS doesn't promise optimality as the solution is not the least cost solution, since the search terminates if a deeper goal state is reached first.

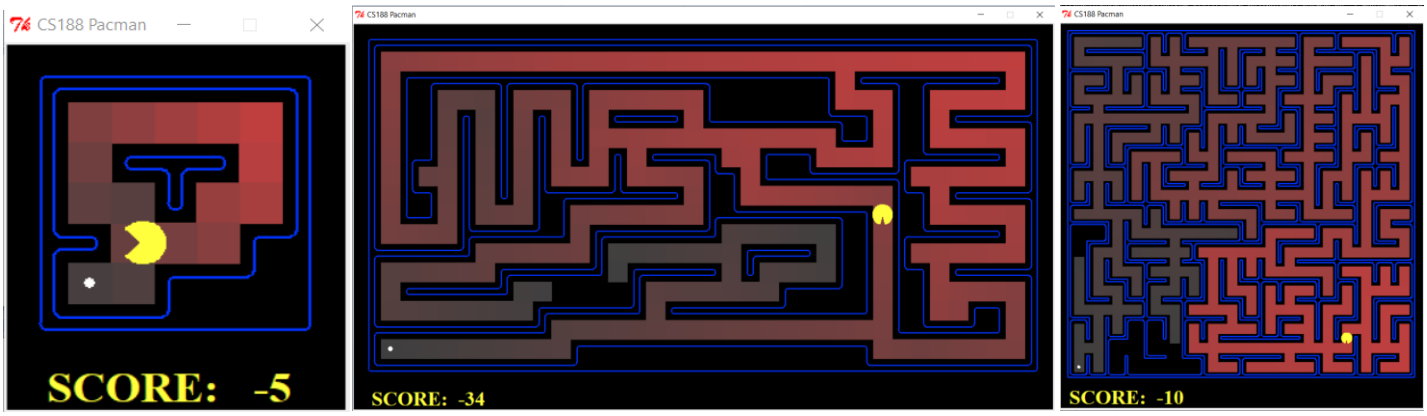
Breadth First Search

BFS expands all nodes at the same level first before proceeding to deeper levels. It is complete and promises an optimal solution, since if a solution exists at a shallower depth, BFS will find it. It uses a FIFO data structure, called queue.

Maze	Total Cost	Nodes Expanded
tinyMaze	10	15
mediumMaze	130	269
bigMaze	210	620

BFS does find the optimal path, but in the process it expands a very large number of nodes, larger than DFS, as shown below.

The BFS algorithm also works for the 8 puzzle problem.

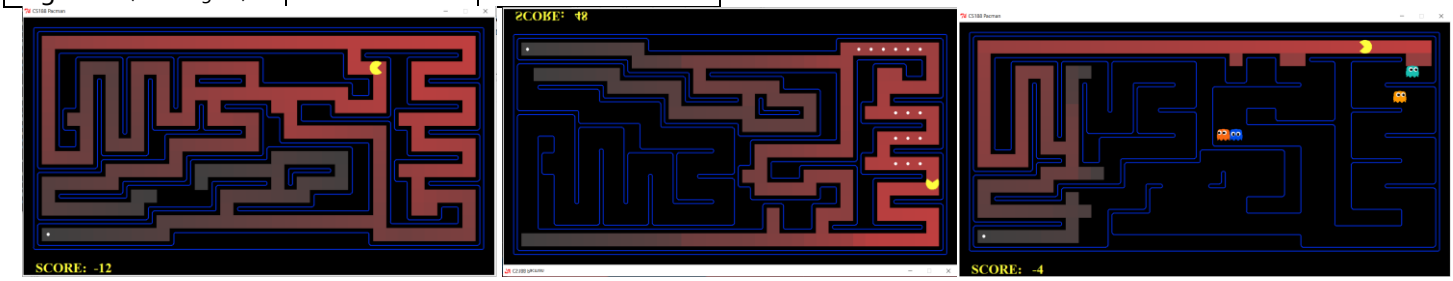


Uniform Cost Search

UCS uses a priority queue. BFS is optimal when the moves are associated by unit cost. UCS is much like BFS, but expands the node with the least path cost. The priority queue selects the node with the least cost path. This guarantees an optimal solution if we use the cost from the initial state to the current state at the cost function.

Maze	Total Cost	Nodes Expanded
mediumMaze	68	269
mediumDottedMaze	1	186
mediumScaryMaze	68719479864	108
bigMaze (not in figure)	210	620

Note the very low and very high path costs for the StayEastSearchAgent and StayWestSearchAgent, which is due to the exponential cost function 0.5^x and 2^x for x in (x, y).



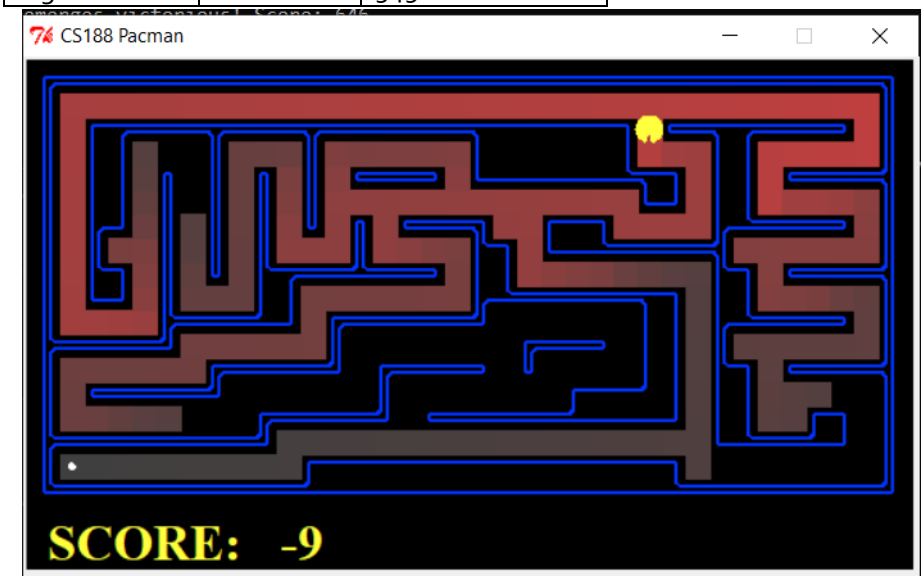
The search agents in all cases managed to find the least cost path and hence are optimal. Although search in mediumScaryMaze was hindered by the ghosts!

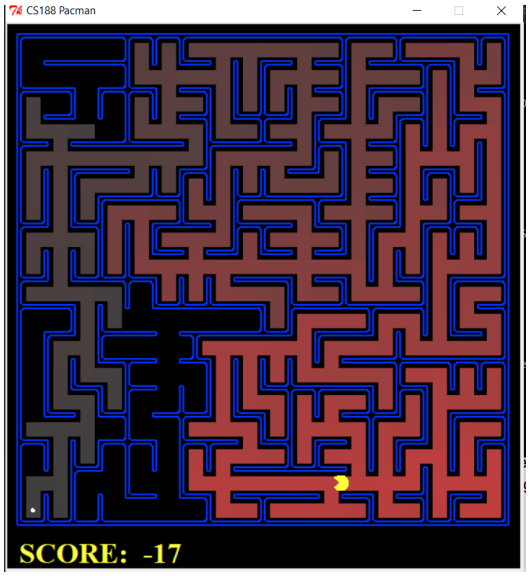
A search*

It is a kind of best first search which evaluates the cost function f(n) = g(n) + h(n), where g(n) is the cost to reach the current state and h(n) is the cost to reach the goal. Hence f(n) is the estimated cost of the cheapest path from the initial state to the goal. A trivial heuristic is 0. We used the manhattanDistance heuristic while running A* search.

Maze	Total Cost	Nodes Expanded
mediumMaze	68	221
bigMaze	210	549

This is a significant improvement over UCS, and this works as long as the heuristic function is admissible and consistent.





openMaze results

Algorithm	Total Cost	Nodes Expanded
DFS	298	576
BFS	54	682
UCS	54	682
A*	54	535

We see that A* with Manhattan distance heuristic performs the best in openMaze

Corners Problem:

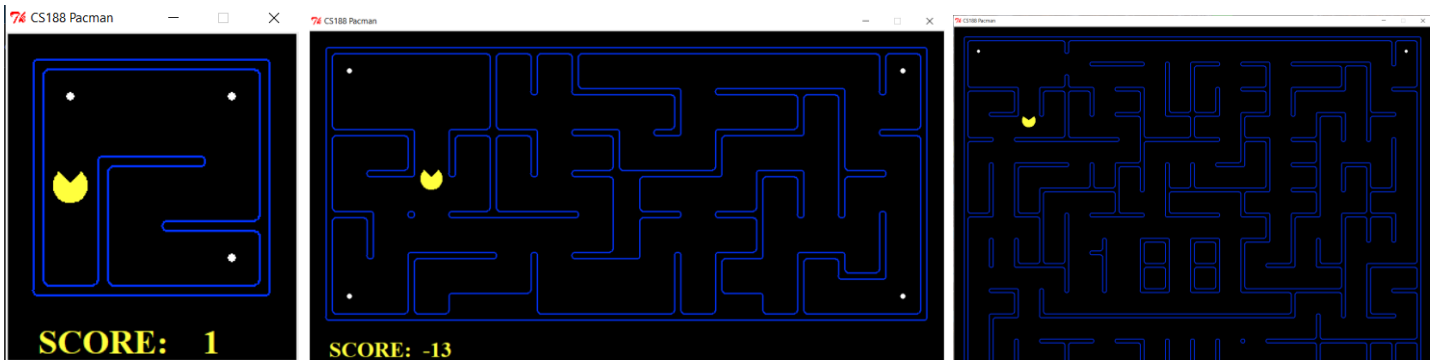
Find the corners

The goal here is to search for food dots present at the four corners of the maze. In order to encode the corners visited in the state space representation, I have represented the states using a two member tuple, whose first member is the current state of the agent and the second member is a set that will contain the visited vertices. Initially, the current state will be the initial state and the visited set will be an empty set. This representation doesn't encode any unnecessary information.

The results using BFS are tabulated below.

Maze	Total Cost	Nodes Expanded
tinyCorners	28	252
mediumCorners	106	1966
bigCorners	162	7949

Now we shall compare this with A* search, with a consistent heuristic.



Corners Heuristic

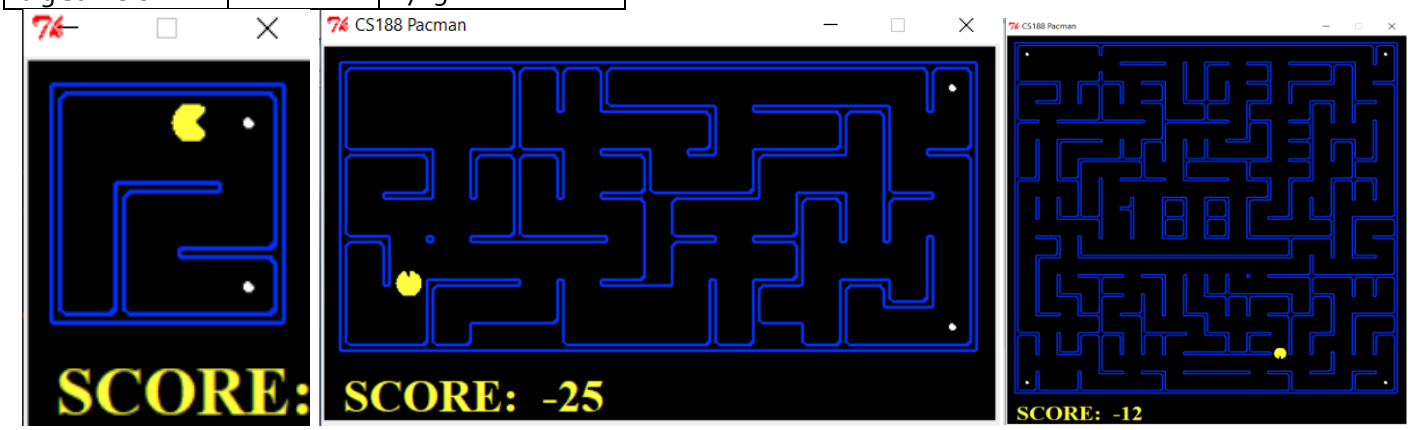
We need to design an admissible and consistent heuristic. We use the Manhattan distance. Precisely the total estimated Manhattan distance to reach the final goal state, goal state being having reach all four corners. First we calculate the Manhattan distance between the current position and the nearest corner. To it we add the Manhattan distance from this corner to the next nearest corner, and from that corner to the next nearest unvisited corner and so on until all the corners are have been accounted for. The sum of these distances weaves an approximate path from the initial state to the goal state. We have ignored walls here to make the problem simpler. Since Manhattan distance is the lower limit on the actual distance, it can never over estimate the actual path cost. Also, the heuristic is zero at the goal state.

The shortest possible actual path would be to first travel to the nearest corner (note that the agent can't travel diagonally) and then to travel along the shorter edge to the second corner, then along the longer edge to the third corner and along another shorter edge to the final corner, which is the goal state. The sum of distances we have calculated as a heuristic is equivalent to this distance. This is the shortest possible path and hence is admissible. It will never be overestimated, and the actual distance in the Pacman world is more, due to the presence of walls. This gives a good approximation on the actual distance to be covered by the agent.

The heuristic is consistent, since we have used Manhattan distances to find the distance. The Manhattan distance is an analogue to the actual physical world distance in the Pacman world. The costs of an action are non-decreasing, just like the actual distance. Hence the heuristic is consistent.

Maze	Total Cost	Nodes Expanded
tinyCorners	28	154
mediumCorners	106	692
bigCorners	162	1725

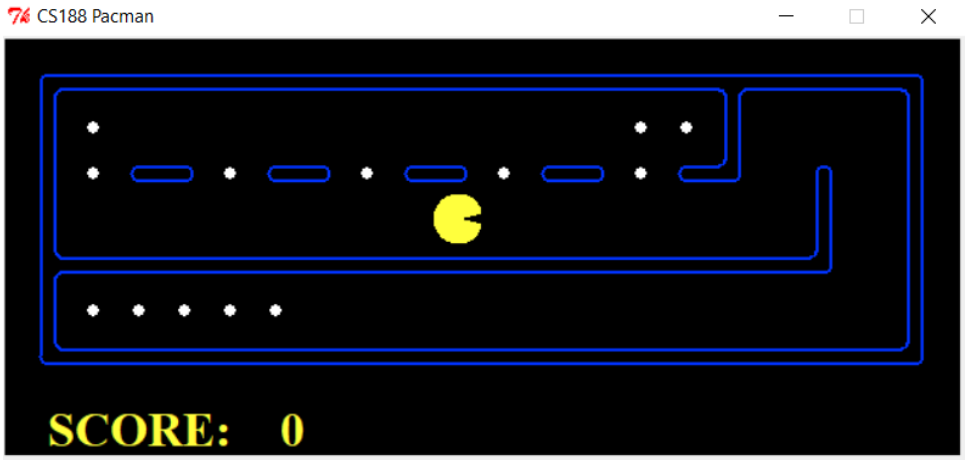
Using A* shows significant improvement over BFS, the total cost is same for both implementations, confirming to a consistent heuristic.



Food heuristic

This problem demonsrates the efficiency of A* search, since our agent has to eat all the food dots this time. We have to use a gamestate this time, to account for the location of all the food dots. In our heuristic we shall be ignoring the walls to make it simpler. Here to get a more accurate value of the distance, we shall use mazeDistance instead of manhattanDistance. We use a similar approach as the previous heuristic. We compute the mazedistance between the current state and all the food points. Of these points we choose the largest distance as the heuristic value, since the farthest dot is the final dot to be visited and eaten, and that is the final state.

This heuristic is certainly admissible as we have taken the distance to the farthest food dot, and this cannot be overestimated. The value of the heuristic is zero at the goal state, as in our implementation, zero is a default value in the distance list. This heuristic is also consistent due to reason stated previously, we have used the maze distance, which indeed is the actual distance between the current state and any other food point. With every subsequent action, the drop in the heuristic will be at most, the cost of the action itself. This holds since we are solving a distance based problem and the naze distance is a true representative of the actual physical distance in the pacman world. Actual distances to be covered to the reach a goal state are always consistent.



For trickySearch 4137 nodes are expanded with a total cost of 60.