CS512

Lab 1

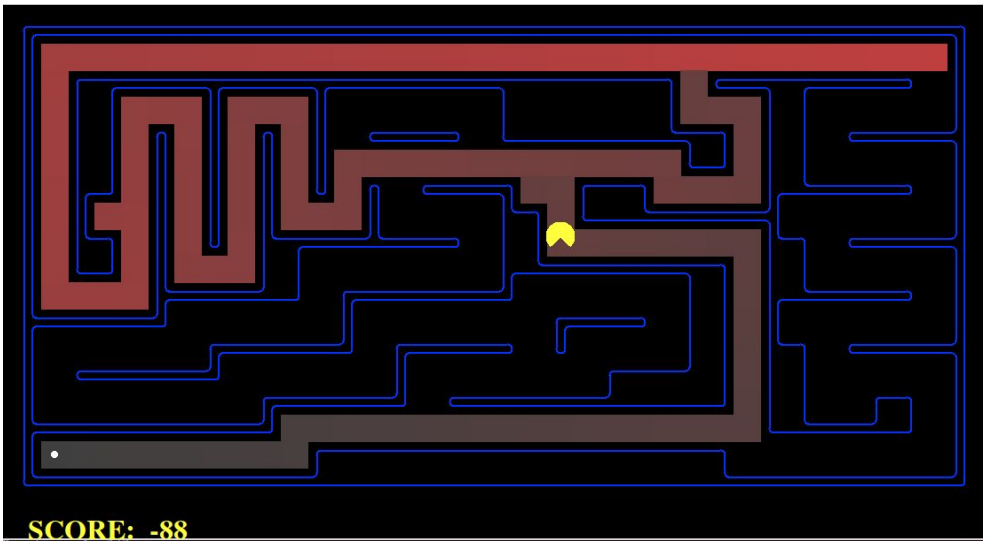
2016csb1050

In this assignment I have implemented various search algorithms, search agents and heuristics to create an agent that plays pacman game efficiently and under different conditions.

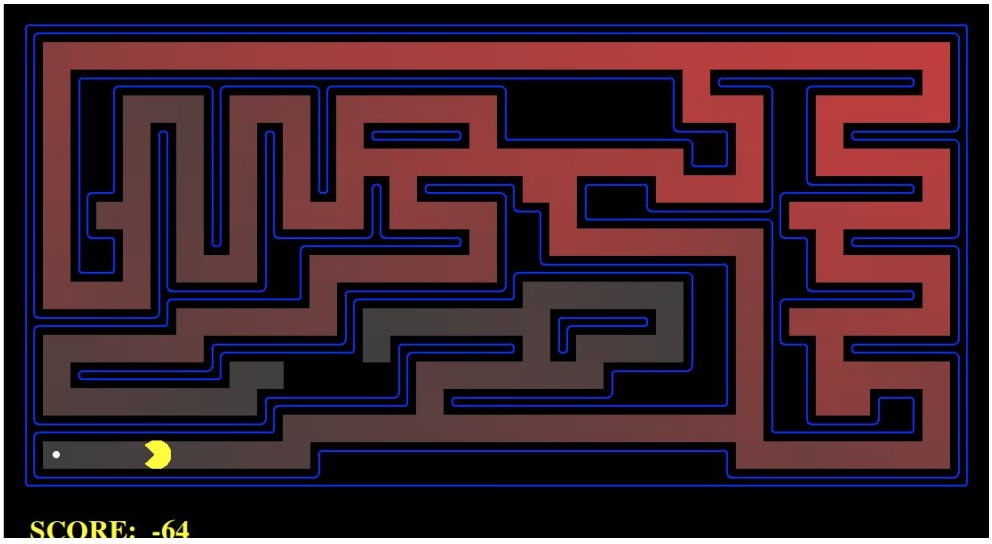
***Question 1* : Depth First Search - Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?**

In Question 1, I have used Depth First Search Algorithm to find the goal state for the pacman. However, the exploration order is not what we expected. The pacman takes a longer route than required. The reason to this behaviour is that DFS is not an optimistic search algorithm, it expands the leftmost node first going all the way deep down without checking the siblings. If the goal state is reached it stops even though there may be a shorter path that has not been discovered. DFS does not take into account path length or path cost while finding the solution.

No, the pacman does not go to all the explored squares in its way. For example, for the mediumMaze the pacman explores 146 nodes while the path cost is 130 (each move costs 1, therefore 130 nodes will be visited). Similarly for bigMaze the pacman expands 390 nodes while it actually goes to only 210.



DFS



BFS

The pacman state for Depth First Search vs Breadth First Search

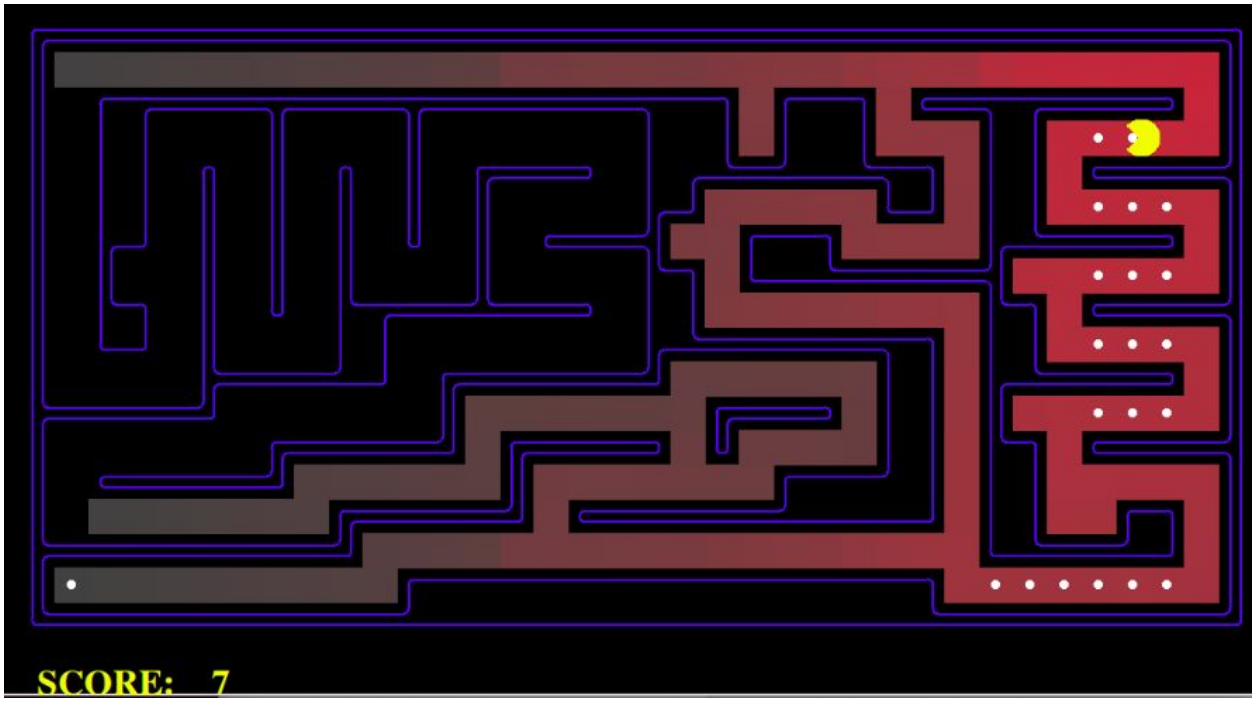
The explored number of nodes is more for BFS than DFS, but BFS gives the least cost solution

***Question 2* : Breadth First Search : Does BFS find a least cost solution?**

Yes BFS finds a least cost solution. BFS explores the graph level-by-level, hence whenever the goal node is found, it is bound to be the shortest path from root to it. Note that BFS finds a path that has shortest length, therefore, the solution is least cost because all the path weights (costs) are identical. However, if the path weight varies then the shortest path will not definitely be the least cost solution.

***Question 3* : Uniform Cost Search**

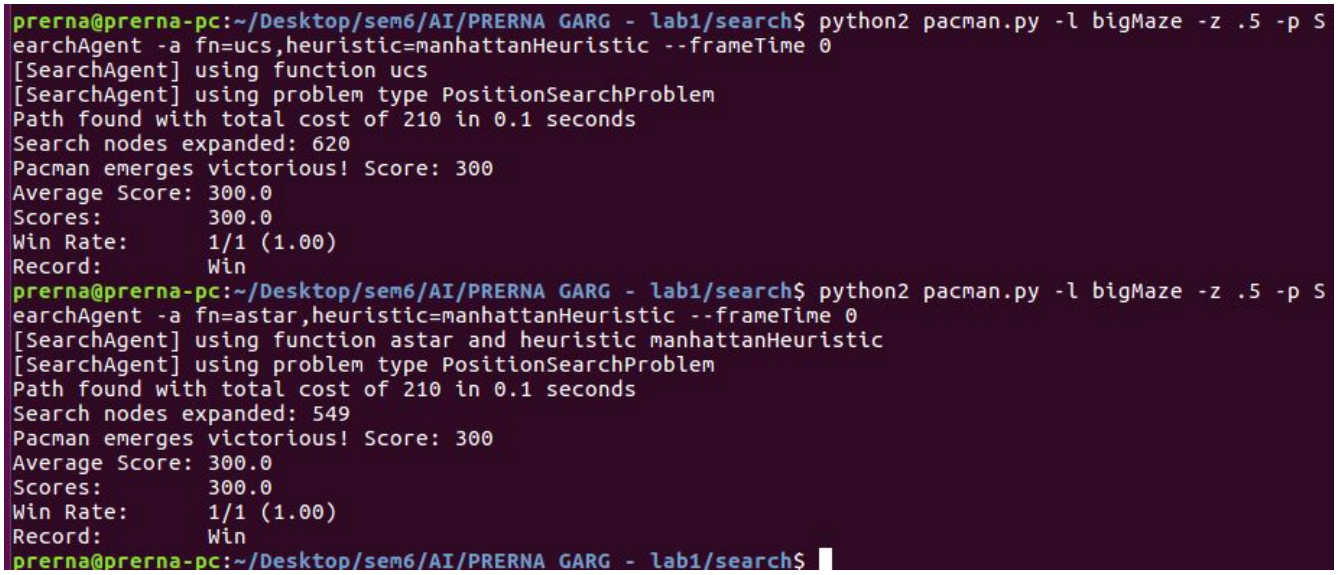
UCS works as expected and finds the solution with least path costs. It works correctly for medium dotted maze as shown below.



Medium Dotted Maze Pacman State

Question 4 : A* Search

For bigMaze, A* finds the optimal solution earlier than UCS *i.e.* it expands lesser number of nodes than UCS. A* not only considers the cost so far for a particular path but also tries to approximate how far the goal is from the current node. This algorithm works better than UCS because UCS only sees cost so far and hence is not as smart as A* .



As we can see in the above picture: for bigMaze, **UCS** finds the solution by expanding ***620*** nodes while ***A**** finds the solution by expanding ***549*** nodes only. A* correctly considers not only the path cost upto now but also the proper estimation of the future path costs.

Question: What happens on openMaze for the various search strategies?

```
prerna@prerna-pc:~/Desktop/sem6/AI/PRERNA GARG - lab1/search$ python2 pacman.py -l openMaze -z .5 -p
SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.1 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
prerna@prerna-pc:~/Desktop/sem6/AI/PRERNA GARG - lab1/search$ python2 pacman.py -l openMaze -z .5 -p
SearchAgent -a fn=ucs,heuristic=manhattanHeuristic
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.1 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
prerna@prerna-pc:~/Desktop/sem6/AI/PRERNA GARG - lab1/search$ python2 pacman.py -l openMaze -z .5 -p
SearchAgent -a fn=bfs,heuristic=manhattanHeuristic
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:        456.0
Win Rate:      1/1 (1.00)
Record:        Win
prerna@prerna-pc:~/Desktop/sem6/AI/PRERNA GARG - lab1/search$ python2 pacman.py -l openMaze -z .5 -p
SearchAgent -a fn=dfs,heuristic=manhattanHeuristic
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 298 in 0.0 seconds
Search nodes expanded: 576
Pacman emerges victorious! Score: 212
Average Score: 212.0
Scores:        212.0
Win Rate:      1/1 (1.00)
Record:        Win
prerna@prerna-pc:~/Desktop/sem6/AI/PRERNA GARG - lab1/search$
```

For openMaze, **UCS** finds the solution by expanding ***682*** nodes while ***A**** finds the solution by expanding ***535*** nodes only. Also, DFS gives a solution with cost 298 while the other algorithms give a solution of cost 54, even though it expands less nodes then BFS and UCS, this is consistent with our understanding of DFS.

Question 5 : Corner Search Problem : Defining state representation

I have used the following representation for the states of pacman for the corner search problem.

Tuple = ((coordinates) , (visited corners))
i.e. the current position of the pacman, and the corners that the pacman has already visited.
Goal State = ((coordinates of the last visited corner) , [corner 1 , corner 2 , corner 3 , corner 4])
i.e. a list containing all the 4 corners and the coordinates of the position when pacman reaches at the goal which will be the coordinates of the last visited corner.

Question 6 : Corners Problem: Heuristic

For this heuristic, I find the the list of unvisited corners first. Then I find the corner that has the minimum manhattan distance from the current state. This distance is added to the heuristic variable and the corner is removed from the list of unvisited corners. The process continues till all the corners are removed from the list of unvisited corners.

Question 7 : Eating All the Dots

For this problem, I have used the heuristic that gives the maximum maze distance to all the dots from the current state. The maze distance here refers to the cost of BFS solution to reach that particular state.

Question 8 : Suboptimal Search

The goal test for this problem is simple. We just have to check whether the current state is one of the states in the list of food states. Once the goal state is defined, I have used Breadth First Search as the search algorithm because BFS finds the goal state with minimum path length. Thus the output of the BFS everytime will be the goal state closest to the current state.

This approach will not work efficiently for all scenarios. Consider the following Pacman Example:

Underscore = space (empty cell)
(.) = dot (food)
% = walls

For the following example
Pacman will move east (cost 1)
Next closest dot is to its west (cost 3)
Next closest dot is to its east (cost 7)
Total = 11
Whereas the optimal is 9 (move west till the end and then east till the end)

%%%%%%%%%%

%(.)__P(.)_____(.)%

%%%%%%%%%%