

# PACMAN ASSIGNMENT ON CLASSICAL SEARCH

**Shilpa Mukhopadhyay- 433003777 UIN**

## QUESTION 1:

**1. Is this implementation recursive?**

Yes, the implementation is recursive. Because in this algorithm, we started with a root node and pushed to stack and checked if the node popped from stack is goal state. If yes, we stopped, else we continued with checking the currently processed node's neighbours and again inserted it into stack. Then the same cycle is continued for each node till goal node is found. Hence it is a recursive algorithm.

**2. In building the graph, what information does each node contain?**

Each node contains 3 information:

- a. State
- b. Action
- c. Cost

**3. What is the data structure used in this implementation of DFS?**

Stack data structure is used in DFS.

**4. In utils.py, you are provided with additional implemented data structures useful for completing the next questions. Provide a list of them and briefly explain the pros and cons of each one.**

Data Structure mentioned in util.py	Pros	Cons	Use
Stack	Stack is a LIFO(Last-In-First-Out) Data structure. If we want to check along the successors of immediately processed node, Stack is useful because we can use the latest processed nodes Hence its space complexity is relatively lesser when we use stack	However if the depth of the graph reaches to an infinity, the algorithms using Stack (e.g. DFS) gets lost or go on for infinite amount of time. The time complexity is huge and there is no guarantee that any solution state can be found, it might just get stuck in a branch of infinite depth. He	DFS(Depth First Search)
Queue	Queue helps us in level order searching of a graph, it is FIFO(First-In-First-Out) data structure.	Since queue uses FIFO while processing, it might be possible that the algorithm using it can create a huge	BFS(Breadth First Search)

	If we want to search for an element it will take $O(n)$ time. The advantage of using queue is that all nodes of a tree can be for sure covered to find a solution	queue of items to process till the earlier items get processed. Hence it requires huge amount of memory space	
Priority Queue	It helps in storing the root element based on some priority(min heap or max heap) The advantage is that retrieval of element is in $O(1)$	Disadvantage is that it take $O(\log n)$ to store the element at correct place using Heapify. Hence everytime we want to take out k-elements. K-heapify actions would be needed	Uniform Cost Search

**5. As mentioned earlier, the abstract SearchProblem class has four methods. Try to see the functionality of each by printing the outputs.**

The four methods are:

- getStartState: returns the state state position (x,y)
- isGoalState: returns the goal state position (x,y)
- getSuccessors: get the successor lists. Each item has 3 values: position of successor in (x,y), Direction wrt the current node and the cost from current state to the particular successeore node.
- getCostOfActions: returns the cost of action to process one node to get its corresponding successor.

```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
getStartState()= (5, 5)
getSuccessors()= [((5, 4), 'South', 1), ((4, 5), 'West', 1)]
cost of action= 1 for neighbour= (5, 4) to state= (5, 5)
cost of action= 1 for neighbour= (4, 5) to state= (5, 5)
getSuccessors()= [((5, 5), 'East', 1), ((3, 5), 'West', 1)]
cost of action= 1 for neighbour= (5, 5) to state= (4, 5)
cost of action= 1 for neighbour= (3, 5) to state= (4, 5)
getSuccessors()= [((4, 5), 'East', 1), ((2, 5), 'West', 1)]
cost of action= 1 for neighbour= (4, 5) to state= (3, 5)
cost of action= 1 for neighbour= (2, 5) to state= (3, 5)
getSuccessors()= [((3, 5), 'East', 1), ((1, 5), 'West', 1)]
cost of action= 1 for neighbour= (3, 5) to state= (2, 5)
cost of action= 1 for neighbour= (1, 5) to state= (2, 5)
getSuccessors()= [((1, 4), 'South', 1), ((2, 5), 'East', 1)]
cost of action= 1 for neighbour= (1, 4) to state= (1, 5)
cost of action= 1 for neighbour= (2, 5) to state= (1, 5)
getSuccessors()= [((1, 5), 'North', 1), ((1, 3), 'South', 1)]
cost of action= 1 for neighbour= (1, 5) to state= (1, 4)
cost of action= 1 for neighbour= (1, 3) to state= (1, 4)
getSuccessors()= [((1, 4), 'North', 1), ((2, 3), 'East', 1)]
cost of action= 1 for neighbour= (1, 4) to state= (1, 3)
cost of action= 1 for neighbour= (2, 3) to state= (1, 3)
getSuccessors()= [((2, 2), 'South', 1), ((1, 3), 'West', 1)]
cost of action= 1 for neighbour= (2, 2) to state= (2, 3)
cost of action= 1 for neighbour= (1, 3) to state= (2, 3)
getSuccessors()= [((2, 3), 'North', 1), ((2, 1), 'South', 1), ((3, 2), 'East', 1)]
cost of action= 1 for neighbour= (2, 3) to state= (2, 2)
cost of action= 1 for neighbour= (2, 1) to state= (2, 2)
```

```

cost of action= 1 for neighbour= (3, 2) to state= (2, 2)
getSuccessors()= [((4, 2), 'East', 1), ((2, 2), 'West', 1)]
cost of action= 1 for neighbour= (4, 2) to state= (3, 2)
cost of action= 1 for neighbour= (2, 2) to state= (3, 2)
getSuccessors()= [((4, 3), 'North', 1), ((3, 2), 'West', 1)]
cost of action= 1 for neighbour= (4, 3) to state= (4, 2)
cost of action= 1 for neighbour= (3, 2) to state= (4, 2)
getSuccessors()= [((4, 2), 'South', 1), ((5, 3), 'East', 1)]
cost of action= 1 for neighbour= (4, 2) to state= (4, 3)
cost of action= 1 for neighbour= (5, 3) to state= (4, 3)
getSuccessors()= [((5, 4), 'North', 1), ((4, 3), 'West', 1)]
cost of action= 1 for neighbour= (5, 4) to state= (5, 3)
cost of action= 1 for neighbour= (4, 3) to state= (5, 3)
getSuccessors()= [((5, 5), 'North', 1), ((5, 3), 'South', 1)]
cost of action= 1 for neighbour= (5, 5) to state= (5, 4)
cost of action= 1 for neighbour= (5, 3) to state= (5, 4)
getSuccessors()= [((2, 2), 'North', 1), ((1, 1), 'West', 1)]
cost of action= 1 for neighbour= (2, 2) to state= (2, 1)
cost of action= 1 for neighbour= (1, 1) to state= (2, 1)
goalstate= (1, 1)
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 30
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores: 500.0
Win Rate: 1/1 (1.00)
Record: Win

```


## 6. Given the above information, explain the working flow of the implemented DFS

- Get the problem start state store it at root node, assigning action as empty initially and cost as 0 to begin with.
- Push this root node to stack
- While stack is not empty, pop the top element.
- If the element's state denotes the goal state, terminate the algorithm.
- Else update the visited tag of the popped element and get its one neighbour, process it(update the action and cost) and push it in stack.
- Step c-d is continued till stack is empty.

Basically DFS select one particular and goes down its depth, processes all nodes and then moves to the nearest parent's other branch and continues the process till goal state is found.

## 7. Execute the following commands code should quickly find a solution:

```
python pacman.py -l tinyMaze -p SearchAgent
```

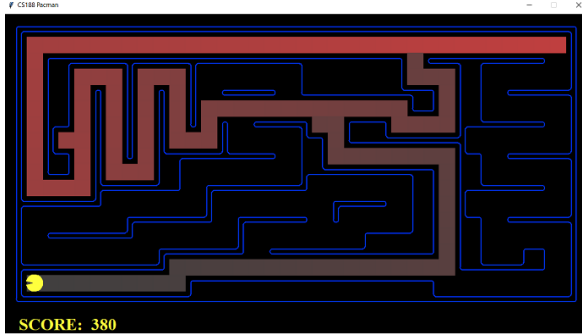


```

(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacm
an.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores: 500.0
Win Rate: 1/1 (1.00)
Record: Win

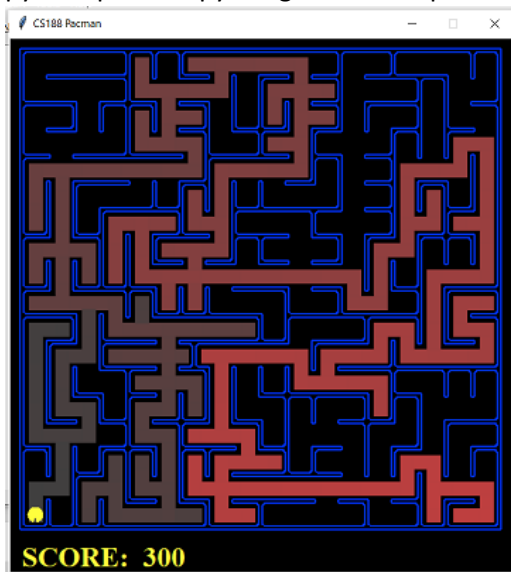
```

```
python pacman.py -l mediumMaze -p SearchAgent
```



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:      380.0
Win Rate:    1/1 (1.00)
Record:      Win
```

```
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

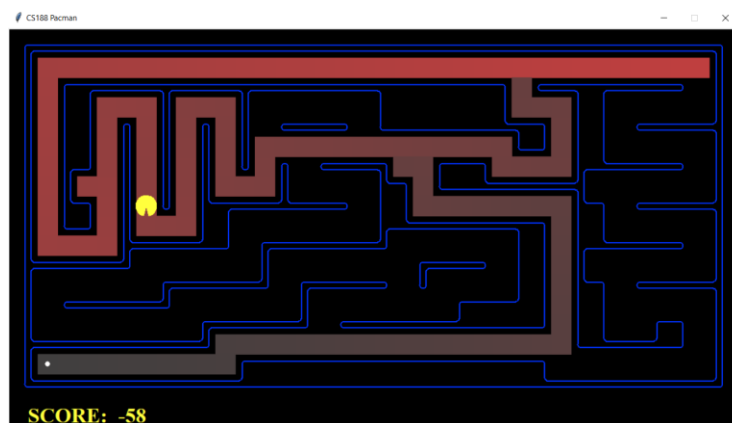


```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l bigMaze -z .5 -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:      300.0
Win Rate:    1/1 (1.00)
Record:      Win
```

8. The Pacman board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected? Does Pacman actually go to all the explored squares on his way to the goal?

Yes, since this is a DFS algorithm the Pacman is taking the first branch it is finding. Its along the depth.

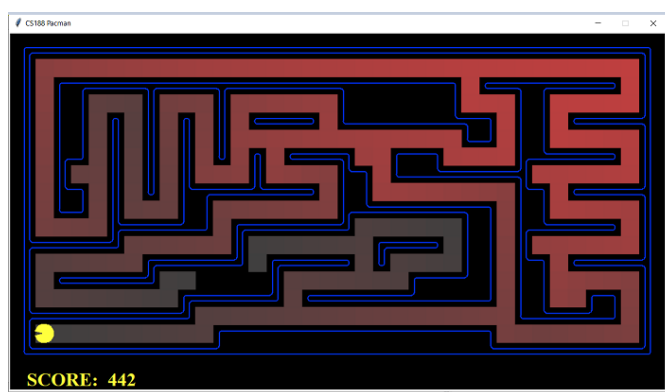
However, to answer the second question, it does not go to all explored squares on its way to the goal. Since it is a Depth First Search, it might be possible that one branch is explored completely and goal is not found, then DFS goes to the nearest parent with an unexplored child and starts from there. Because of this it might be possible that the nodes have been explored , but they have not been considered for the goal path.



## QUESTION 2:

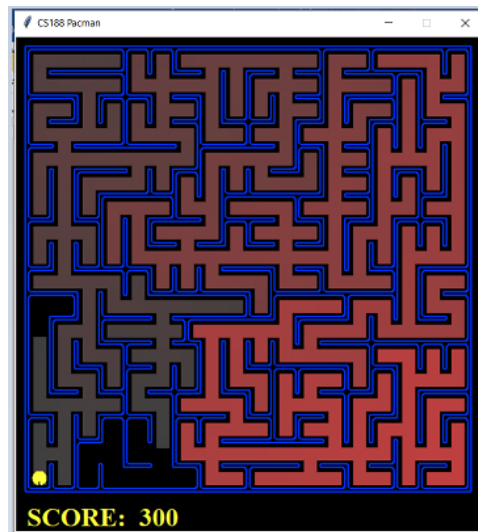
Test your code the same way you did for depth-first search.

- a. `python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win
```

b. `python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacm
an.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:      300.0
Win Rate:    1/1 (1.00)
Record:      Win
```

**Does BFS find a least cost solution? If not, check your implementation**

**Observation:**

Yes. Here are the given comparison from above screenshots:

Algorithm	Mazetype	Cost
BFS	Medium	68
BFS	Large	210
DFS	Medium	130
DFS	Large	210

**Observation:** In both the cases we see that the cost of BFS search  $\leq$  cost of DFS Search

If you've written your search code generically, your code should work equally well for the eight-puzzle search problem without any changes. python eightpuzzle.p

```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python eightpuzzle.py
A random puzzle:
-----
| 4 | 3 | 2 |
-----
|   | 1 | 5 |
-----
| 6 | 7 | 8 |
-----

BFS found a path of 5 moves: ['up', 'right', 'down', 'left', 'up']
After 1 move: up
-----
|   | 3 | 2 |
-----
| 4 | 1 | 5 |
-----
| 6 | 7 | 8 |
-----

Press return for the next state...
After 2 moves: right
-----
| 3 |   | 2 |
-----
| 4 | 1 | 5 |
-----
| 6 | 7 | 8 |
-----

Press return for the next state...
```

```
After 3 moves: down
-----
| 3 | 1 | 2 |
-----
| 4 |   | 5 |
-----
| 6 | 7 | 8 |
-----

Press return for the next state...
After 4 moves: left
-----
| 3 | 1 | 2 |
-----
|   | 4 | 5 |
-----
| 6 | 7 | 8 |
-----

Press return for the next state...
After 5 moves: up
-----
|   | 1 | 2 |
-----
| 3 | 4 | 5 |
-----
| 6 | 7 | 8 |
-----

Press return for the next state...

(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>
```

### QUESTION 3:

#### Observation:

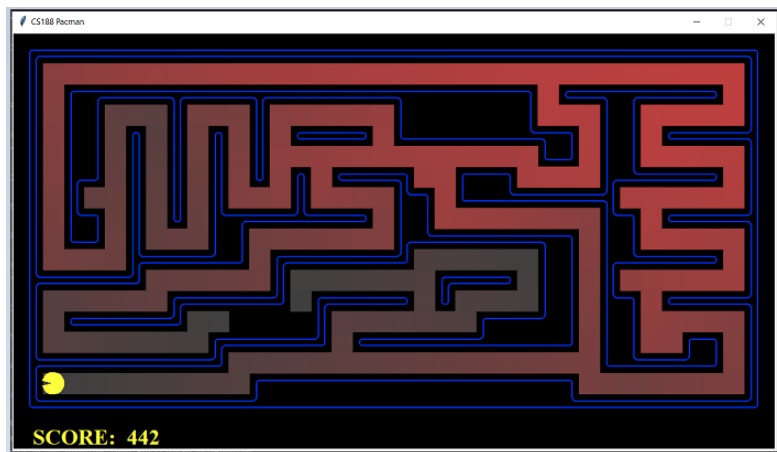
1. As per question it is observed that very low and very high path costs exist for the StayEastSearchAgent and StayWestSearchAgent respectively, due to their exponential cost functions . Below is the given comparison table and the screenshots of the execution along with commands:

Maze path	Cost
StayEastSearchAgent	1
StayWestSearchAgent	68719479864
Basic UCS	68

2. Observed successful behaviours in all three of the following layout. The difference in cost is compared in above table.

#### Basic UCS run on mediumMaze:

a. `python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs`

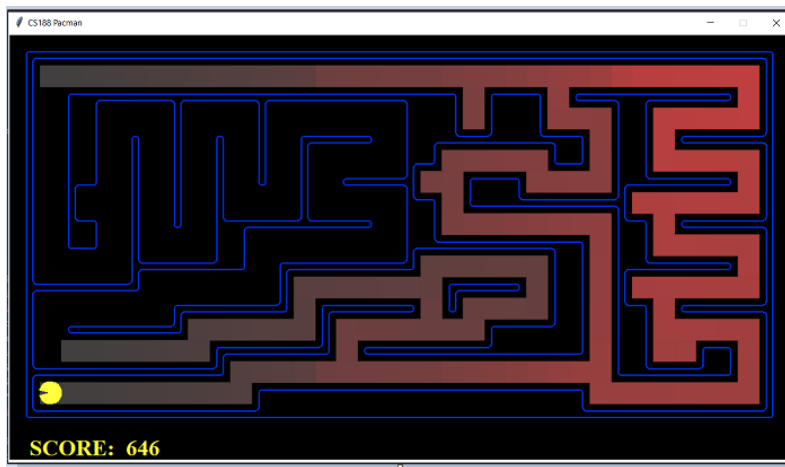


```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:      442.0
Win Rate:    1/1 (1.00)
Record:      Win
```



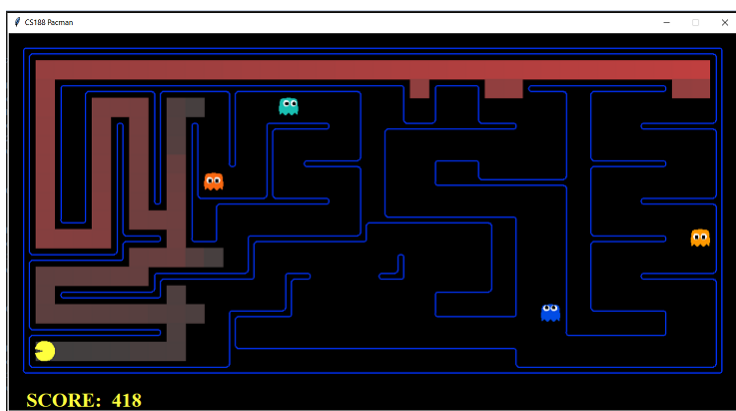
East and West Agent runs:

b. `python pacman.py -l mediumDottedMaze -p StayEastSearchAgent`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:      646.0
Win Rate:    1/1 (1.00)
Record:      Win
```

c. `python pacman.py -l mediumScaryMaze -p StayWestSearchAgent`

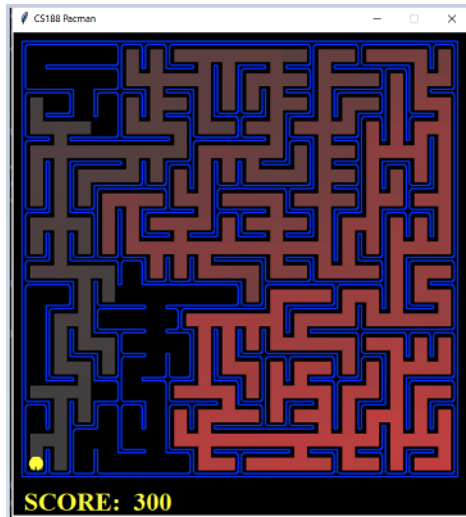


```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:      418.0
Win Rate:    1/1 (1.00)
Record:      Win
```

## QUESTION 4:

Basic implementation testing:

- a. `python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l bigMaze -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 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:      300.0
Win Rate:    1/1 (1.00)
Record:      Win
```

**Observation as per question:** A\* finds the optimal solution slightly faster than uniform cost search (about 549 vs. 620 search nodes expanded in our implementation, but ties in priority may make your numbers differ slightly).

A-star

```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l bigMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:      300.0
Win Rate:    1/1 (1.00)
Record:      Win
```

## UCS

```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l bigMaz
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:      300.0
Win Rate:    1/1 (1.00)
Record:      Win
```

	UCS	A-Star
<b>Nodes expanded</b>	<b>620</b>	<b>549</b>

## What happens on openMaze for the various search strategies

Comparison for all 4 algorithms on openMaze:

Algorithms	Cost	No of Nodes Expanded	Score
DFS	298	576	212
BFS	54	682	456
UCS	54	682	456
A-Star	54	535	456

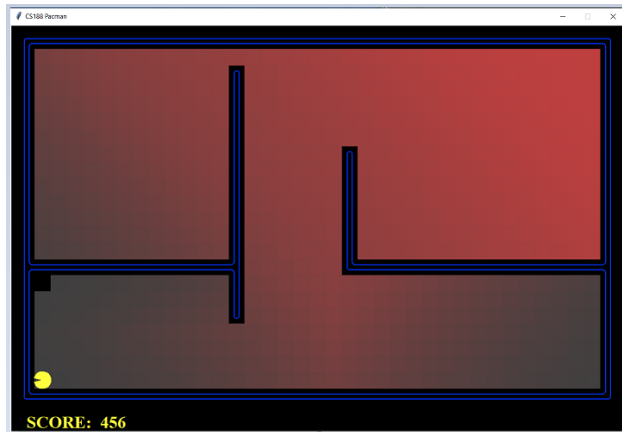
Observation: A-Star performs the best of all with minimum number of nodes expanded, least cost. BFS and UCS has same performance. DFS is not an optimal algorithm, hence its cost is more than the others.

### a. DFS: python pacman.py -l openMaze -p SearchAgent



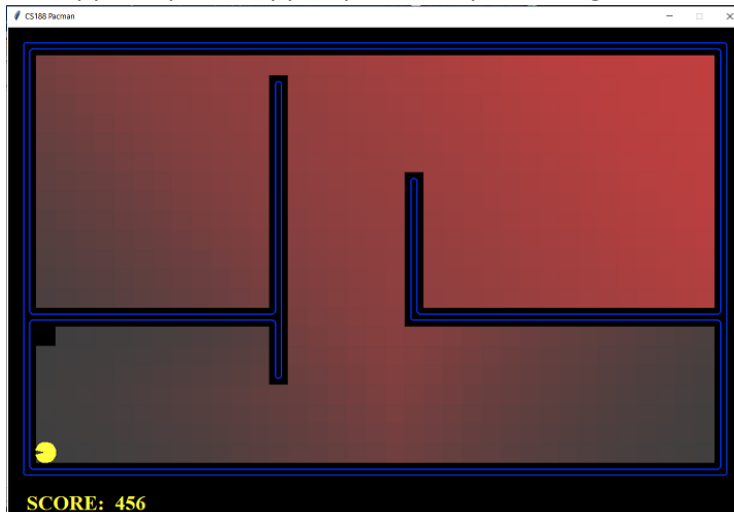
```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l openMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 298 in 0.0 seconds
Search nodes expanded: 576
Pacman emerges victorious! Score: 212
Average Score: 212.0
Scores:      212.0
Win Rate:    1/1 (1.00)
Record:      Win
```

- b. **BFS:** `python pacman.py -l openMaze -p SearchAgent -a fn=bfs`



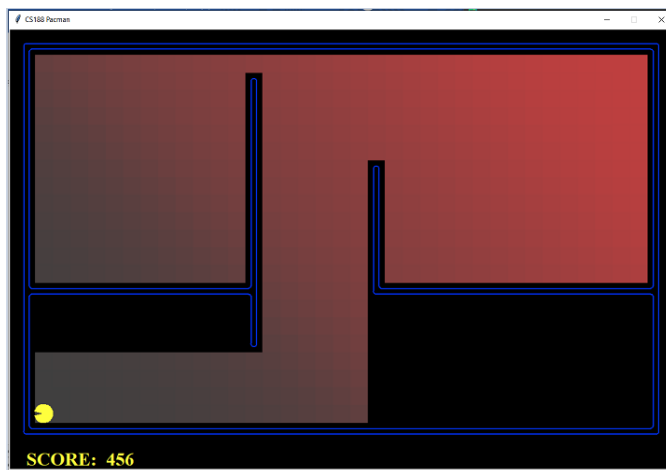
```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l openMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores:      456.0
Win Rate:    1/1 (1.00)
Record:      Win
```

- c. **UCS:** `python pacman.py -l openMaze -p SearchAgent -a fn=ucs`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l openMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs and heuristic nullHeuristic
[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
```

- d. **A-star:** python pacman.py -l openMaze -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l openMaze -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 0.0 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
```

## QUESTION 5:

**1. What is the state representation? In other words, what information are encoded for each state?**

- a. Current location of the agent (x,y)
- b. Details of corners unexplored as list storing the (x,y) coordinate of each corner.

Eg. In the below screenshot, once the agent reaches(6,6) which an unexplored corner enlisted in the corner list, that particular corner is removed from list.s

```
state= ((2, 5), [(1, 1), (1, 6), (6, 1), (6, 6)])  
state= ((6, 6), [(1, 1), (1, 6), (6, 1)])
```

**2. An abstract state representation should not encode irrelevant information (like the position of ghosts, where extra food is, etc.). In particular, using Pacman GameState as a search state should be avoided or your code will be very, very slow if you do (and also wrong). Can you think of alternative state representation satisfying these properties?**

There can be many alternate representations Two kinds of alternate representation that I can think are noted down below and a little discussion about both:

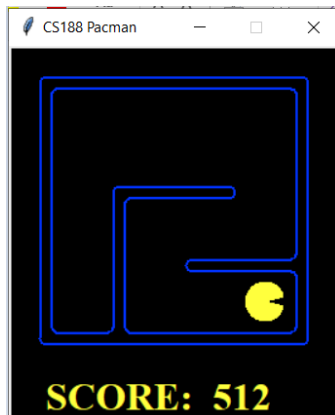
- a. State= ((position x,y), (Manhattan Distance to the nearest Corner), (Direction to that nearest corner))  
We can have a separate list of corner positions, we can keep on checking if any of the goal state item is reached and remove them. The final goal state will be an empty list of corners remaining.
- b. State= ((position x,y), (List of available Actions from the current state))  
In this case, we can check the heuristic value (e.g. Manhattan Distance or Euclidean Distance) of the next state after each item in the action list is completed. Then the action which leads to a state with minimum heuristic value can be taken up.

**3. How does the agent check whether a state is a goal state?**

In order for a state to be a goal state, the number of corners explored has to be zero. Hence the length of remaining number of corner=0

4. Implement `getSuccessors(self, state)` Now, your search agent should be able to solve:

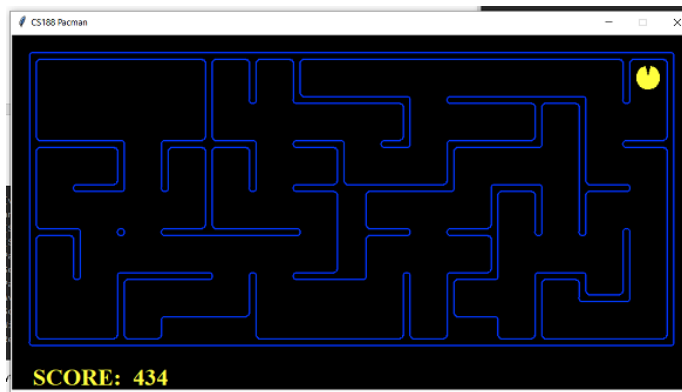
a. `python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 252
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:      512.0
Win Rate:    1/1 (1.00)
Record:      Win
```

**Observation:** As mentioned in the question, this algorithm takes a total cost of 28 for tinyCorners

c. `python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:      434.0
Win Rate:    1/1 (1.00)
Record:      Win
```

**Observation:** As mentioned in the question, this algorithm takes less than 2000 node expansion for the mediumCorners

## QUESTION 6.:

In order for the Corner Heuristic to be consistent, we do the following analysis:

We know that if the calculated heuristic value of a node  $\leq$  the calculated heuristic of its neighbour + cost of path from that neighbour to the node, then the heuristic is consistent. Also the proof of heuristic as per question is to show that UCS and Astar algorithms must return paths of same length.

The analysis based on this is given below:

- Here, heuristic used is the Manhattan distance to the goal.
- Table to show comparison:

Algorithm	Cost	Nodes expanded
UCS	106	1966
A-star	106	692

- Conclusion: Even though the total number of nodes expanded are different, the path to the goal state has same cost. Hence for the same problem, the heuristic algorithm using Manhattan distance is consistent.

The execution commands and screenshots are shows below:

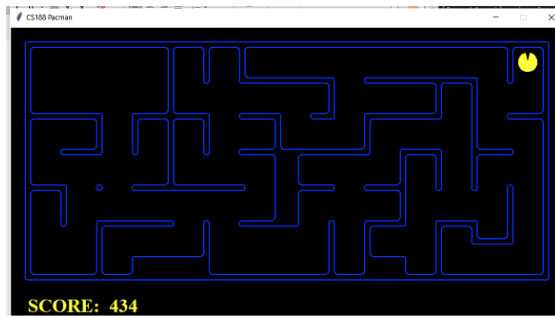
- a. **python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5**



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 0.0 seconds
Search nodes expanded: 692
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:      434.0
Win Rate:    1/1 (1.00)
Record:      Win
```



UCS: python pacman.py -l mediumCorners -p SearchAgent -a fn=ucs,prob=CornersProblem,heuristic=cornersHeuristic



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l mediumCorners -p SearchAgent -a fn=ucs,prob=CornersProblem,heuristic=cornersHeuristic
[SearchAgent] using function ucs and heuristic cornersHeuristic
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 0.2 seconds
Search nodes expanded: 1966
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:      434.0
Win Rate:    1/1 (1.00)
Record:      Win
```

## QUESTION 7:

This algorithm has been tested on two Searches- Test Search and Tricky Search as per question and the analysis results and observations are discussed first. Later the screenshots of the run has also been shown :

### Comparison Table:

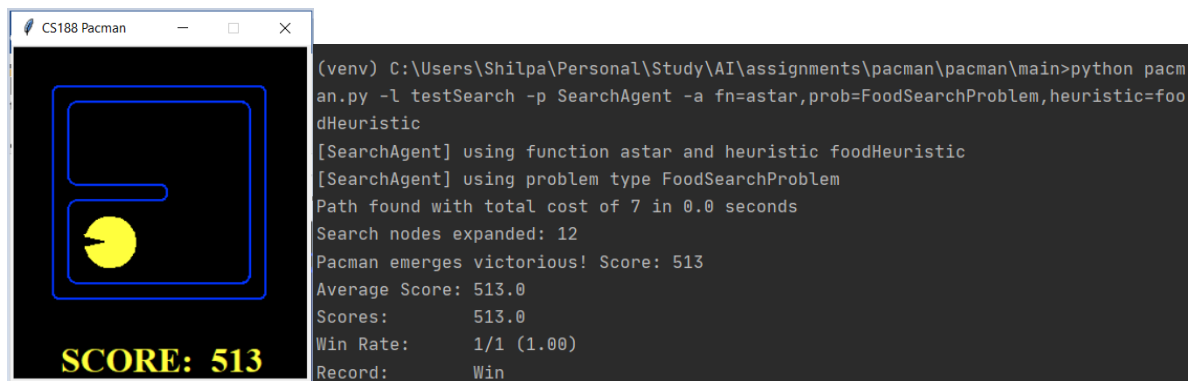
Food Placement Type	Search Algorithm	Cost	Number of Nodes	Time Taken
testSearch	Astar	7	12	0.0 seconds
testSearch	UCS	7	14	0.0 seconds
trickySearch	Astar	60	9551	6.3 seconds
trickySearch	UCS	60	16688	21.7 seconds

### Observations:

- We have found UCS slowing down as the problem got more complex even though the cost remains same, the number of nodes expanded and time taken is better in case of A-star than UCS.

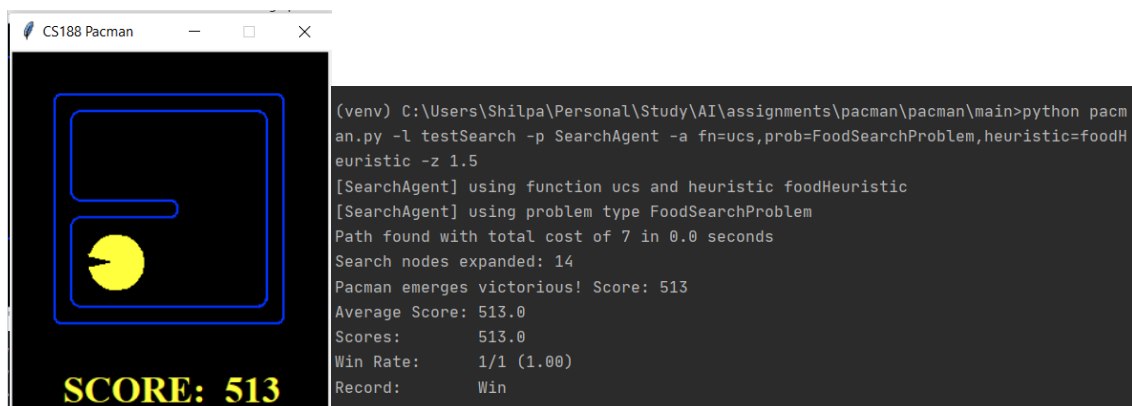
## TestSearch- Astar:

- a. `python pacman.py -l testSearch -p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic`



- b. Test Search- UCS:

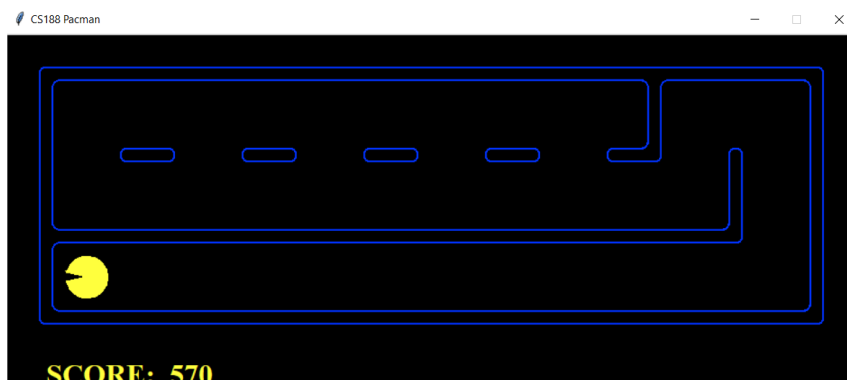
`python pacman.py -l testSearch -p SearchAgent -a fn=ucs,prob=FoodSearchProblem,heuristic=foodHeuristic -z 1.5`



## Tricky Search:

- c. Tricky Search Sstar:

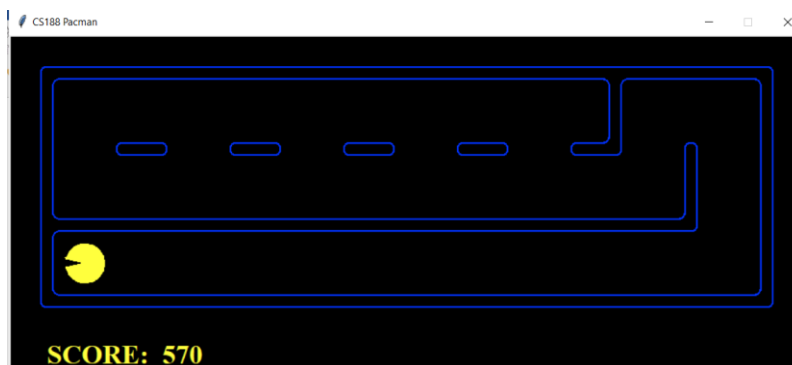
`python pacman.py -l trickySearch -p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic -z 1.5`



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l trickySearch -p SearchAgent -a fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic -z 1.5
[SearchAgent] using function astar and heuristic foodHeuristic
[SearchAgent] using problem type FoodSearchProblem
Path found with total cost of 60 in 6.3 seconds
Search nodes expanded: 9551
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:      570.0
Win Rate:    1/1 (1.00)
Record:      Win
```

#### d. Tricky Search- UCS:

```
python pacman.py -l trickySearch -p SearchAgent -a
fn=ucs,prob=FoodSearchProblem,heuristic=foodHeuristic -z 1.5
```



```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman.py -l trickySearch -p SearchAgent -a fn=ucs,prob=FoodSearchProblem,heuristic=foodHeuristic -z 1.5
[SearchAgent] using function ucs and heuristic foodHeuristic
[SearchAgent] using problem type FoodSearchProblem
Path found with total cost of 60 in 21.7 seconds
Search nodes expanded: 16688
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:      570.0
Win Rate:    1/1 (1.00)
Record:      Win
```

## QUESTION 8:

In the given question, I have been asked to select a search algorithm which will make the Food search problem a greedy one in terms of selecting closest path to the food. That means we need to find a shortest route optimal algorithm in searching food dot so that the entire food search function becomes a greedy algorithm. Therefore, A-star should be our solution because A-star algorithm is optimal in nature and hence guaranteed to return the lowest cost path to a food. The function gets this return value and chooses the path accordingly in a greedy manner.

In answer to the next question as to why ClosestDotSearchAgent wont always find the shortest possible path through the maze. Let us try to understand from the below screenshots, taken at a time during running the command: `python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5`



In the second scenario (picture with score 1817) shown above, the agent has eaten all the closest point dots and reached the bottom left. However, there are two clusters of dot which stayed behind without visiting because none of these dots fell under the closet dot category when the agent was crossing that area (picture with score 1264). Sadly, now the agent has to go back a huge distance till the nearest bunch of points first (whose surroundings have already been covered once), process all the nodes in that bunch. Then again, the agent has to traverse to the second unexplored area containing a single dot. This will result in unnecessary addition to path cost of retraversal in the same area making the algorithm suboptimal.

```
(venv) C:\Users\Shilpa\Personal\Study\AI\assignments\pacman\pacman\main>python pacman
an.py -l bigSearch -p ClosestDotSearchAgent -z .5
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:      2360.0
Win Rate:    1/1 (1.00)
Record:      Win
```

As mentioned in the question, indeed the cost is 350.

## Acknowledgements:

1. Artificial Intelligence: A Modern Approach, by Russell and Norvig, 4th edition.
2. <https://github.com/vishwapardeshi/AI-Pacman-Projects>
3. <https://github.com/karlapalem/UC-Berkeley-AI-Pacman-Project>