Project 01: The Searchin' Pac-Man

Contents

1 Explanation

1.1 Question 1: Depth First Search

Data Structures:

stack	It contains triplets of: (coordinates of node in the fringe list, action, corresponding cost)
visited	It is a Dictionary containing the nodes which have been popped from the stack, and the corresponding action.
solution	It is a List containing the sequence of actions for Pacman to get from the start state to the goal state.
parents	It is a Dictionary containing nodes and their parents.

Time Complexity: O(b^m) [here b = 3]

Memory Usage (Space Complexity)**:** O(bm) [here b = 3]

(b = branching factor i.e. number of successors of each node
m = maximum depth of solution)

1.2 Question 2: Breadth First Search

Data Structures:

queue	It contains triplets of: (coordinates of node in the fringe list, action, corresponding cost)
visited	It is a Dictionary containing the nodes which have been popped from the stack, and the corresponding action.
solution	It is a List containing the sequence of actions for Pacman to get from the start state to the goal state.
parents	It is a Dictionary containing nodes and their parents.

Time Complexity: $O(b^d)$ [here b = 3]

Memory Usage (Space Complexity)**:** $O(bd)$ [here b = 3]

(b = branching factor i.e. number of successors of each node
d = depth of optimal solution)

1.3 Question 3: Uniform Cost Search

Data Structures:

Priority Queue	It contains triplets of: (coordinates of node in the fringe list, action, corresponding cost) along with the associated cost as the priority
visited	It is a Dictionary containing the nodes which have been popped from the stack, and the corresponding action.
solution	It is a List containing the sequence of actions for Pacman to get from the start state to the goal state.
parents	It is a Dictionary containing nodes and their parents.
cost	It is a Dictionary containing nodes and their corresponding costs from the start state.

Time Complexity: $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ [here b = 3, ϵ = 1]

Memory Usage (Space Complexity): $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ [here b = 3, ϵ = 1]

(b = branching factor i.e. number of successors of each node
C^* = cost of optimal solution
ϵ = maximum cost of each action)

1.4 Question 4: A Star Search

Data Structures:

Priority Queue	It contains triplets of: (coordinates of node in the fringe list, action, corresponding cost) along with the associated cost as the priority

visited	It is a Dictionary which contains the nodes which have been popped from the stack, and the direction from which they were obtained.
solution	It is a List containing the sequence of directions for Pacman to get to the goal state.
parents	It is a Dictionary containing nodes and their parents.
cost	It is a Dictionary containing nodes and their corresponding costs from the start state.

Time Complexity: O(b^d) [here b = 3]

Memory Usage (Space Complexity): O(b^d) [here b = 3]

(b = branching factor i.e. number of successors of each node
d = depth of optimal solution)

This is the worst case complexity obtained when null heuristic is used(same as BFS). It changes according to the heuristic used.

1.5 Question 5: Corners Problem - Breadth First Search

Our state is comprised of Pacman's current position and a list of non-visited corners.
Every time a new corner is visited, it is removed from the list.
When all the corners are visited, the list becomes empty and the state satisfies the goal state condition.

1.6 Question 6: Corners Problem - A Star Search

The heuristic function works as follows:

1.	The heuristic is initialized to zero.
2.	The Manhattan distances between Pacman's current position and all the remaining corners are calculated.

3.	The corner having minimum Manhattan distance is chosen, and this distance is added to the heuristic.
4.	Then the Manhattan distances between the chosen corner and all other corners are calculated.
5.	The corner having the maximum Manhattan distance is chosen, and this distance is added to the heuristic.
6.	The heuristic is returned.

Admissibility:

The heuristic is admissible because in any case, Pacman must cover the distance from the current position to the closest corner, and also the distance from the closest corner to the farthest corner from there. (There are no exceptions.)

Consistency:

The heuristic is consistent because as Pacman reaches closer to the corner, the Manhattan distance between the current position and the closest corner keeps decreasing.

1.7 Question 7: Food Search Problem - A Star Search

The heuristic function works as follows:

1.	The heuristic is initialized to zero.
2.	The Manhattan distances between Pacman's current position and all the remaining foods are calculated.
3.	The food having minimum Manhattan distance is chosen, and this distance is added to the heuristic.
4.	Then the Manhattan distances between the chosen food and all other foods are calculated.
5.	The food having the maximum Manhattan distance is chosen, and this distance is added to the heuristic.
6.	The heuristic is returned.

Admissibility:

The heuristic is admissible because in any case, Pacman must cover the distance to the closest food, and also the distance from the closest food to the farthest food from there. (There are no exceptions.)

Consistency:

The heuristic is consistent because as Pacman reaches closer to the food, the Manhattan distance between the current position and the closest food keeps decreasing.

2 Other efforts

Before coming up with this heuristic, we tried and implemented various other heuristics:

1. Euclidean distance between current position and closest food/corner.
   Number of nodes expanded in food search (tricky) = 14191

2. Euclidean distance between current position and farthest food/corner.
   Number of nodes expanded in food search (tricky) = 10383

3. Manhattan distance between current position and closest food/corner.
   Number of nodes expanded in food search (tricky) = 13930

4. Manhattan distance between current position and farthest food/corner.
   Number of nodes expanded in food search (tricky) = 9928

5. Sum of Manhattan distances between: (current position, closest food), (closest food, next closest food) and so on.
   On inspection, we realized that this heuristic is not admissible, as the actual distance may be less than the calculated heuristic in some cases (for example, food_heuristic_15.test)

3 Statistics

Algorithm	Maze	Cost	Nodes Expanded	Running Time (seconds)	Score
Depth First Search	Tiny	10	15	0.0	500
	Medium	130	146	0.0	380
	Big	210	390	0.1	300
Breadth First Search	Tiny	8	15	0.0	502
	Medium	68	269	0.0	442
	Big	210	620	0.1	300
Uniform Cost Search	Tiny	8	15	0.0	502
	Medium	68	269	0.0	442
	Big	210	620	0.1	300
	Medium Dotted Maze	1	186	0.0	646
	Medium Scary Maze	68719479864	108	0.0	418
A star	Tiny	8	14	0.0	502
	Medium	68	222	0.0	442
	Big	210	549	0.1	300
Corner Problem - Breadth First Search	Tiny Corners	28	252	0.0	512
	Medium Corners	106	1966	0.5	434
	Big Corners	162	7949	7.4	378
Corner Problem - A Star	Tiny Corners	28	165	0.0	512
	Medium Corners	106	783	0.1	434
	Big Corners	162	1945	0.6	378
Food Search	Test Search	7	12	0.0	513
	Tricky Search	60	8366	16.3	570

4 Outputs

This section contains the outputs obtained on running all the algorithms on various maze layouts.

4.1 Depth First Search

4.1.1 Tiny Maze

```
python pacman.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
```

4.1.2 Medium Maze

```
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
```

4.1.3 Big Maze

```
python pacman.py -l bigMaze -p SearchAgent -z 0.5
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

4.2 Breadth First Search

4.2.1 Tiny Maze

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.2.2 Medium Maze

```
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
```

4.2.3 Big Maze

```
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z 0.5
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

4.3 Uniform Cost Search

4.3.1 Tiny Maze

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.3.2 Medium Maze

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[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
```

4.3.3 Big Maze

```
python pacman.py -l bigMaze -p SearchAgent -a fn=ucs -z 0.5
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

4.3.4 Medium Dotted Maze (StayEastSearchAgent)

```
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
```

4.3.5 Medium Scary Maze (StayWestSearchAgent)

```
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
```

4.4 A Star Search

4.4.1 Tiny Maze

```
python pacman.py -l tinyMaze -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 8 in 0.0 seconds
Search nodes expanded: 14
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.4.2 Medium Maze

```
python pacman.py -l mediumMaze -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 68 in 0.0 seconds
Search nodes expanded: 222
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.4.3 Big Maze

```
python pacman.py -l bigMaze -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic -z 0.5
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 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
```

4.5. Corners Problem - Breadth First Search

4.5.1 Tiny Corners Maze

```
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
```

4.5.2 Medium Corners Maze

```
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 0.5 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
```

4.5.3 Big Corners Maze

```
python pacman.py -l bigCorners -p SearchAgent -a
fn=bfs,prob=CornersProblem -z 0.5
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 162 in 7.4 seconds
Search nodes expanded: 7949
Pacman emerges victorious! Score: 378
Average Score: 378.0
Scores:        378.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.6 Corners Problem - A Star Search

4.6.1 Tiny Corners Maze (AStarCornersAgent)

```
python pacman.py -l tinyCorners -p AStarCornersAgent
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 165
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.6.2 Medium Corners Maze (AStarCornersAgent)

```
python pacman.py -l mediumCorners -p AStarCornersAgent
Path found with total cost of 106 in 0.1 seconds
Search nodes expanded: 783
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.6.3 Big Corners Maze (AStarCornersAgent)

```
python pacman.py -l bigCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 162 in 0.6 seconds
Search nodes expanded: 1945
Pacman emerges victorious! Score: 378
Average Score: 378.0
Scores:        378.0
Win Rate:      1/1 (1.00)
Record:        Win
```

4.7 Food Search (AStarFoodSearchAgent)

```
python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 16.3 seconds
Search nodes expanded: 8366
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```