Project 01: The Searching Pac-Man

COURSE: ARTIFICIAL INTELLIGENCE (CSE 537)

INSTRUCTOR: Professor I.V. Ramakrishnan

PROJECT PARTNERS

GULSHAN BHATIA (111491348)

gbhatia@cs.stonybrook.edu

HIMANI SINGH (111491995)

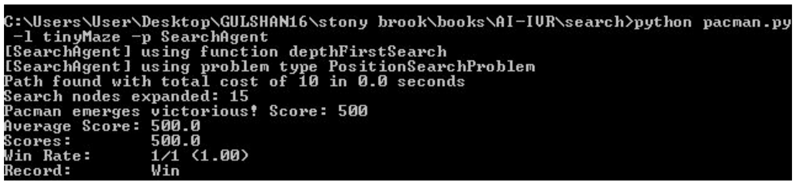
hisingh@cs.stonybrook.edu

1. Question1 -DEPTH FIRST SEARCH

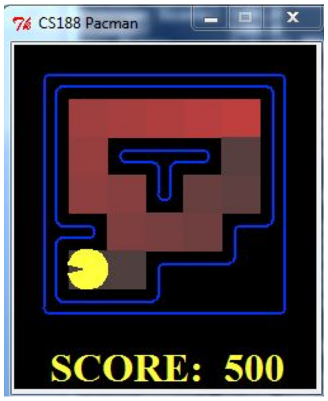
Searches the deepest nodes first in the search tree for expansion. We used Stacks from util.py to store successors.
Space complexity of DFS = O(bd)

1.2 Output:

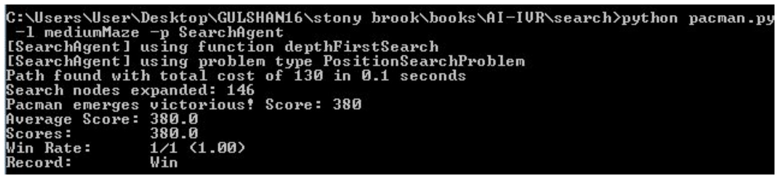
1.2.1 python pacman.py -l tinyMaze -p SearchAgent



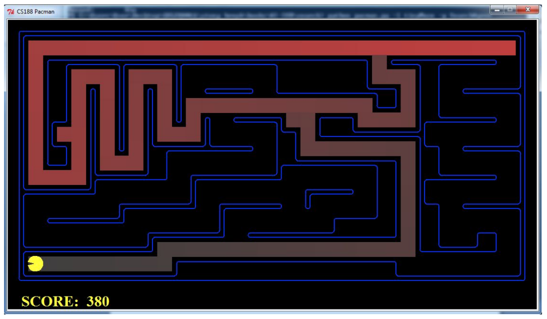
```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
```



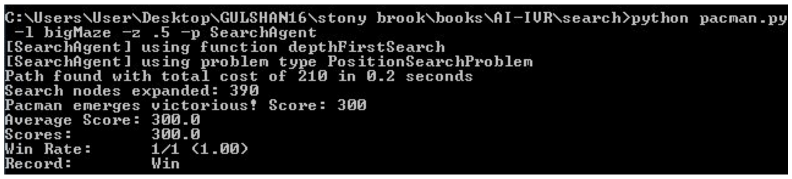
1.2.2 python pacman.py -l mediumMaze -p SearchAgent



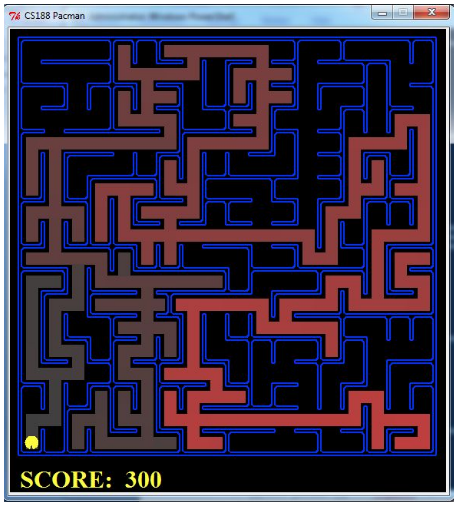
```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.1 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```



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



```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>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 0.2 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
```



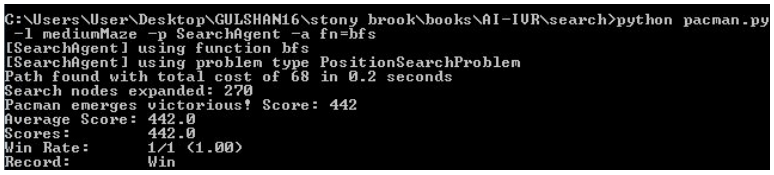
2. Question2 -BREADTH FIRST SEARCH

Searches the shallowest nodes first in the search tree for expansion. Used Queues from util.py to store successors. The only difference between the implementation of bfs & dfs is the use of queues instead of stacks.

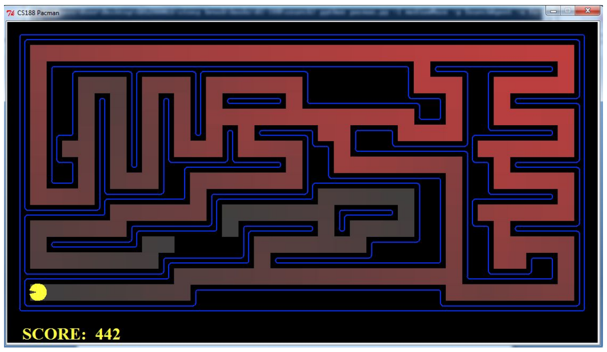
Space complexity of BFS = O(b^d)

2.2 Output:

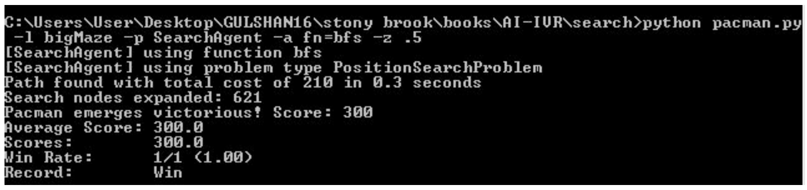
2.2.1 python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

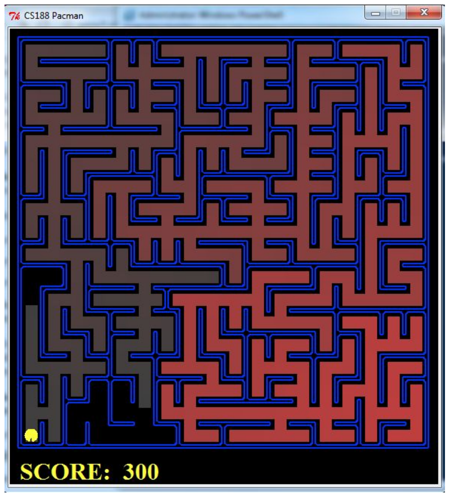


```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.2 seconds
Search nodes expanded: 270
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```



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


```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.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 0.3 seconds
Search nodes expanded: 621
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

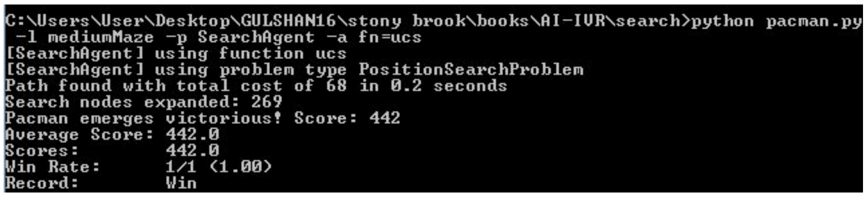


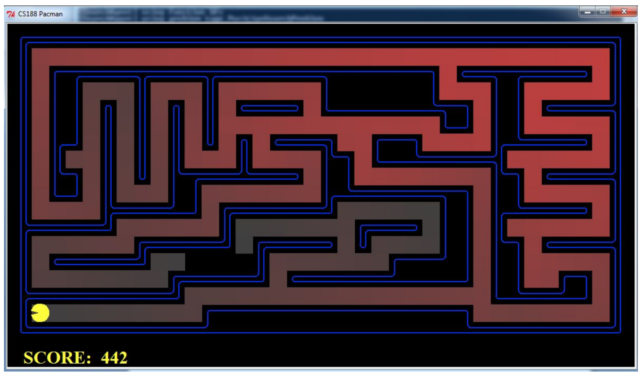
3. Question3- UNIFORM COST SEARCH

Searches the node with the least g(n) value to expand, where g(n) is the total cost to reach the node. We used Priority Queues from util.py to store successors as the next node to be taken from the queue and expanded will be the one with the highest priority, i.e. the lowest score.

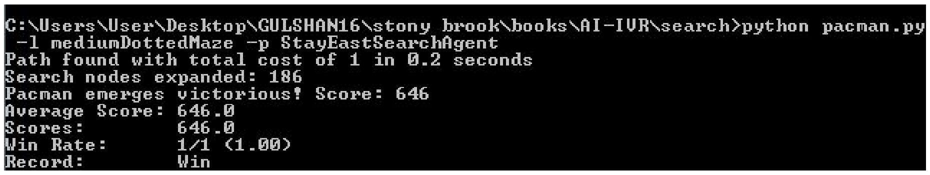
3.2 Output:

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

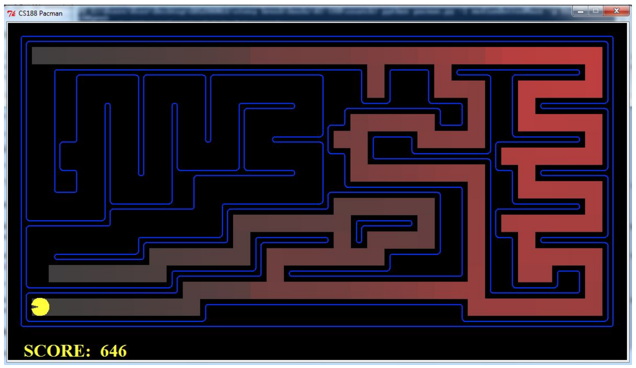

```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.2 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```



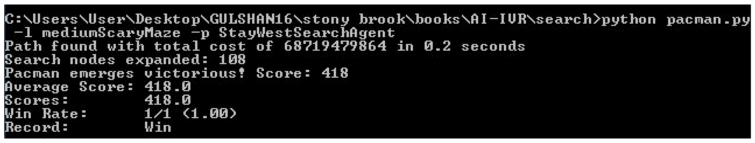
3.2.2 python pacman.py -l mediumDottedMaze -p StayEastSearchAgent



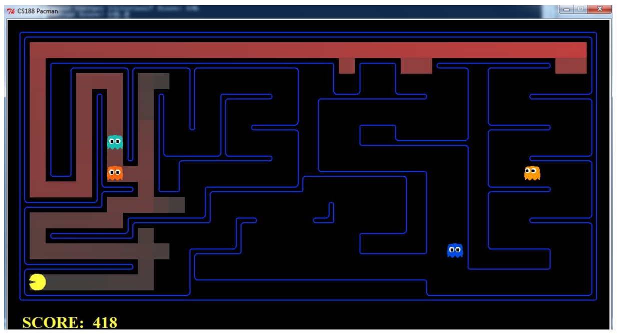
```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.2 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```



3.2.3 python pacman.py -l mediumScaryMaze -p StayWestSearchAgent



```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.2 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
```

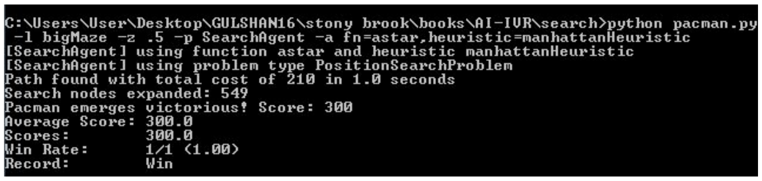


4. Question4- A * SEARCH

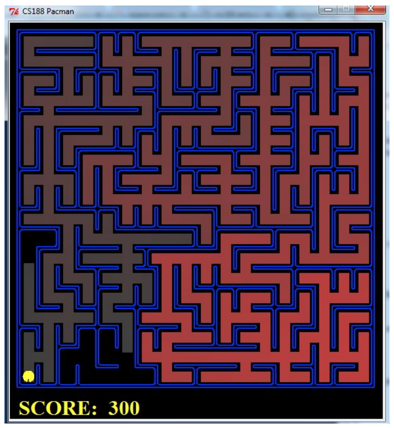
Searches the node which has lowest f(n) value to expand, where f(n) is equal to sum of combined cost(g(n)) and heuristic cost(h(n)). We used Priority Queues from util.py to store successors. The only difference in the implementation of A star and uniform cost search algorithm is to evaluate whether to use f(n) or g(n) cost to reach the node.

4.2 Output:

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



```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>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 1.0 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
```



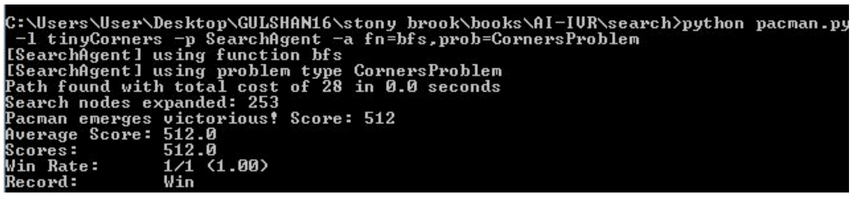
5. Question 5- CORNERS PROBLEM

The corner search problem finds paths through all four corners of a layout. The implemented solution takes the current position and the unvisited four corners as the start state. The successors functions adds the next position to the successors and passes the list of unvisited corners with the state. If the next position is a corner, it removes it from the unvisited corners list and passes the rest. This continues till the pac man reaches the goal state when the length of unvisited corners is zero.

We have expanded a total of **1967** nodes to visit all four corners in the field of a medium maze.

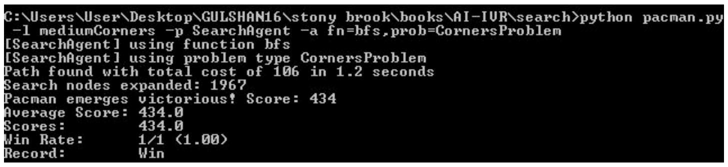
5.2 Output:

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

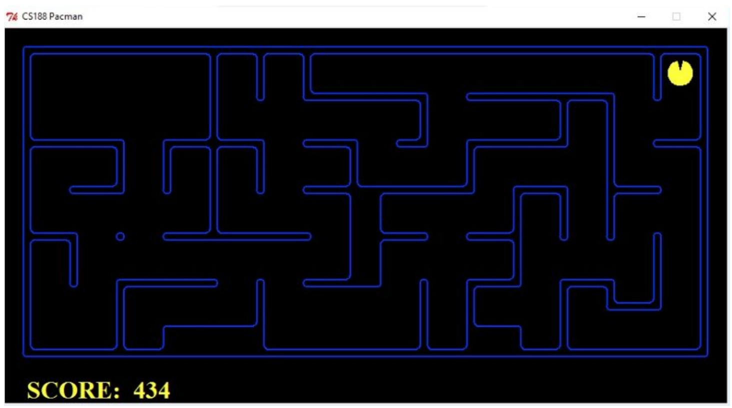


```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 253
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```

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



```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>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 1.2 seconds
Search nodes expanded: 1967
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```



6. CORNERS HEURISTIC

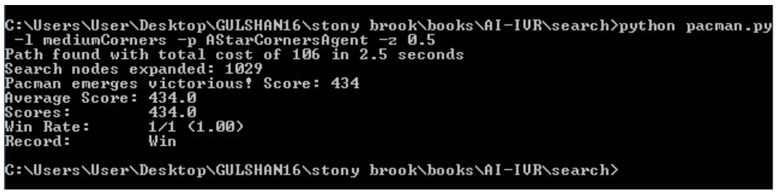
The implemented heuristic function list outs the unvisited corners for the current position and computes the Manhattan distance to each of them. We select the corner with minimum manhattan distance and add it to the list of distances. Then we update the current position of pacman to this corner and removes this corner from the unvisited corners list. We loop over until the unvisited corners is empty.

This heuristic is admissible as it is derived for a relaxed version of the problem which does not include walls to find the paths through all four corners of the layout

We have expanded a total of **1029** nodes in **2.5** seconds for a Medium maze.

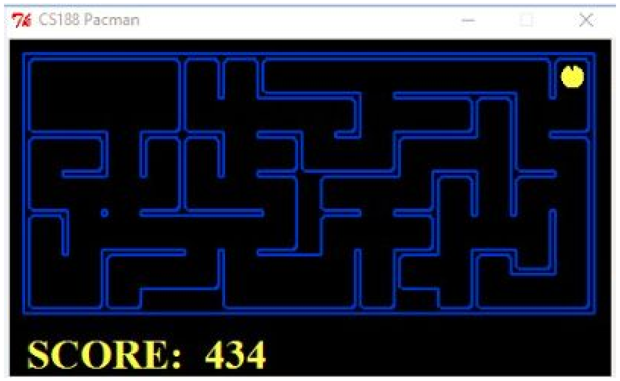
6.2 Output:

6.2.1 python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.



```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 2.5 seconds
Search nodes expanded: 1029
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win

C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>
```



7. FOOD SEARCH HEURISTICS

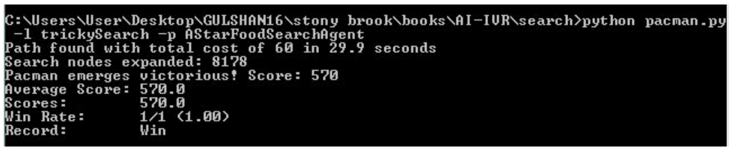
The basic implementation of food search problem is that the Pacman should eat all the food in the grid. The nodes in the state space are the locations of food and the Pacman. The idea behind our heuristic function is to calculate the manhattan distance from the location of Pacman to the location of the food coordinates. In the first for loop, the least distance from the pacman to the food is calculated, i.e. position of closest food is determined. In the second for loop, manhattan distance from closest food to the farthest food is calculated. The sum of all the distances is then returned.

The heuristic is ***admissible*** because the computed heuristic value from pacman position to the food coordinate is less than the manhattan distance between them. To prove that the heuristic is admissible, assume that the pacman traverse through the grid such that the path traversed is less than the sum of all the distances of the nodes in the grid. That means, there exists a path, length of which is less than the sum of all the distances of the nodes(food coordinates, here). This is a contradiction since the pacman needs to eat all the food in the grid and have to traverse the entire grid. Thus, the heuristic is admissible where the heuristic value is less than the true shortest path of the grid.

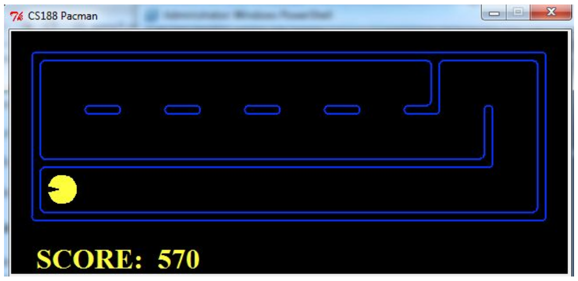
The food search heuristic is calculated after each step pacman takes. If the pacman eats the food, it is one step away from the food. And if the pacman does not eat the food, it moves a step closer to the food. Thus, the heuristic is ***consistent*** because there is no case where it is greater than the heuristic at the next state plus the cost of getting to the next state.

7.2 Output

7.2.1 python pacman.py -l trickySearch -p AStarFoodSearchAgent



```
C:\Users\User\Desktop\GULSHAN16\stony brook\books\AI-IVR\search>python pacman.py
 -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 29.9 seconds
Search nodes expanded: 8178
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```



8. CONSOLIDATED STATISTICS:

Algorithm	Type of Maze	Agent	Running time(in seconds)	Search nodes expanded	Total Cost	Score
depthFirstSearch	tinymaze	SearchAgent	0	15	10	500
depthFirstSearch	mediumMaze	SearchAgent	0	146	130	380
depthFirstSearch	bigMaze	SearchAgent	0.1	390	210	300
breadthFirstSearch	mediumMaze	SearchAgent	0.1	269	68	442
breadthFirstSearch	bigMaze	SearchAgent	0.2	620	210	300
uniformCostSearch	mediumMaze	SearchAgent	0.1	269	68	442
uniformCostSearch	mediumDottedMaze	StayEastSearchAgent	0.1	186	1	646
uniformCostSearch	mediumScaryMaze	StayWestSearchAgent	0	108	68719479864	418
aStarSearch	bigMaze	SearchAgent	1.0	549	210	300
CornersProblem	tinyCorners	SearchAgent	0	253	28	512
CornersProblem	mediumCorners	SearchAgent	1.2	1967	106	434
CornersHeuristic	mediumCorners	AStarCornerAgent	1.6	1029	106	434
FoodSearchproblem	trickySearch	AStarFoodSearchAgent	29.9	8178	60	570

9. REFERENCE:

1. *Artificial Intelligence: A Modern Approach, 3e, Stuart Russell and Peter Norvig*