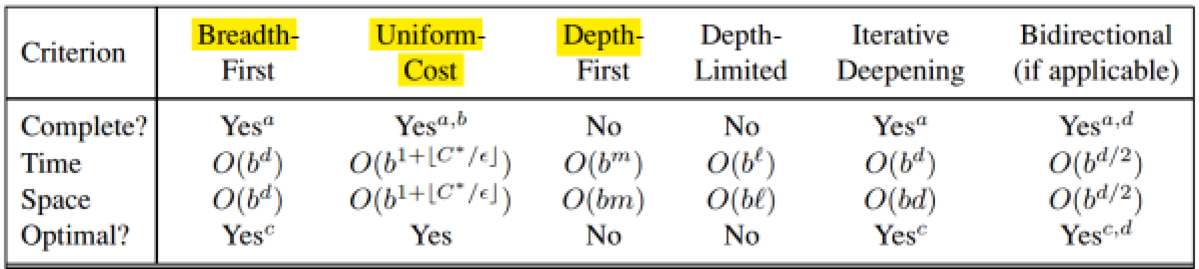
Huan Nguyen 111652587
Han Le 111679478

	MediumMaze			bigMaze		
	#of nodes expanded	Total Cost	Running time	#of nodes expanded	Total Cost	Running time
DFS	**146**	130	**0.0112550258636**	**390**	**210**	**0.0317499637604**
BFS	269	**68**	0.0323598384857	620	**210**	0.0531179904938
UCS	269	**68**	0.0189228057861	620	**210**	0.0515739917755
A*	221	**68**	0.0168349742889	549	**210**	0.0448999404907

The number of expanded nodes depends on the test case. Here we have mediumMaze with start node (34, 16) far away from goal node (1,1), so the goal is at a deep layer. Following the analysis of algorithms below, BFS is not efficient in test case where goal is at deep layer because we need to exhaustively expand all less deeper layer. That explains why DFS outperform other searches.

The processing time mostly depends on number of expanded nodes, so DFS also outperforms other searches. Based on the analysis below, DFS cannot guarantee that the result is optimal while BFS and UCS will output optimal result.

Substantially, BFS and UCS are the same if every cost is unit cost. However, UCS utilizes Priority Queue which takes a little more time than Queue FIFO, hence the performance is better for BFS.



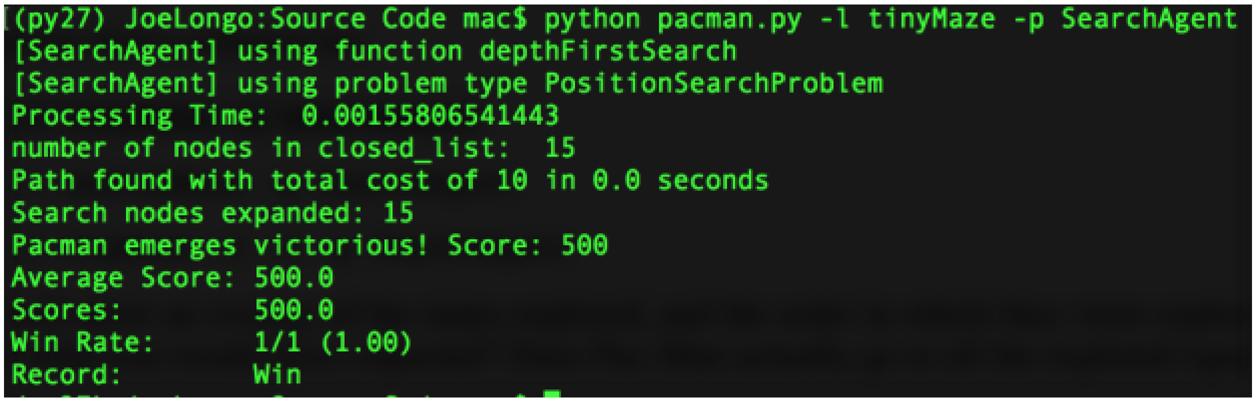
Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs $\geq \epsilon$ for positive ϵ; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

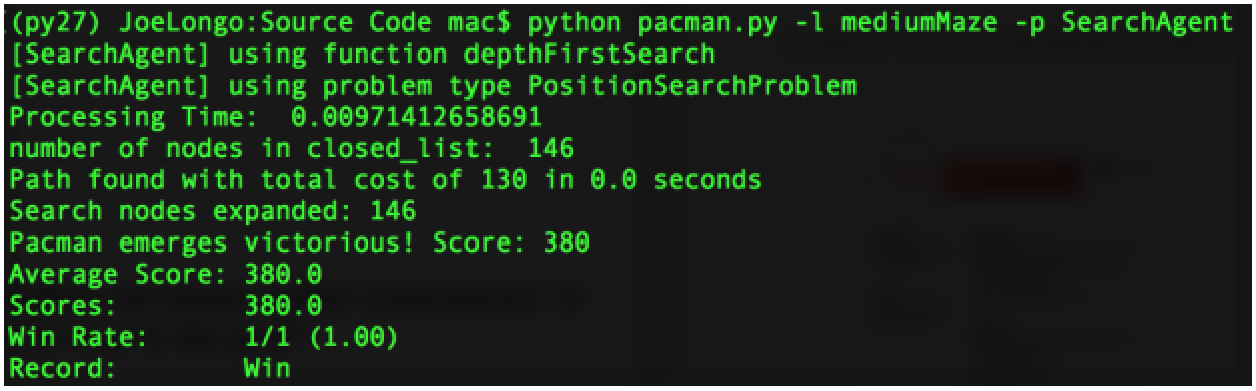
Question 1: Depth First Search:

- **Data structure**: Stack (FILO)
- **Complete**: No. Although we can check whether new nodes are on the path from root to current node to avoid infinite loops in finite space, the algorithm still fails in case it reaches an infinite non-goal path.
- **Optimal**: No. Technically, DFS expands the deepest nodes of the current frontier of the tree, so we cannot verify if a path reaching the goal is an optimal solution.

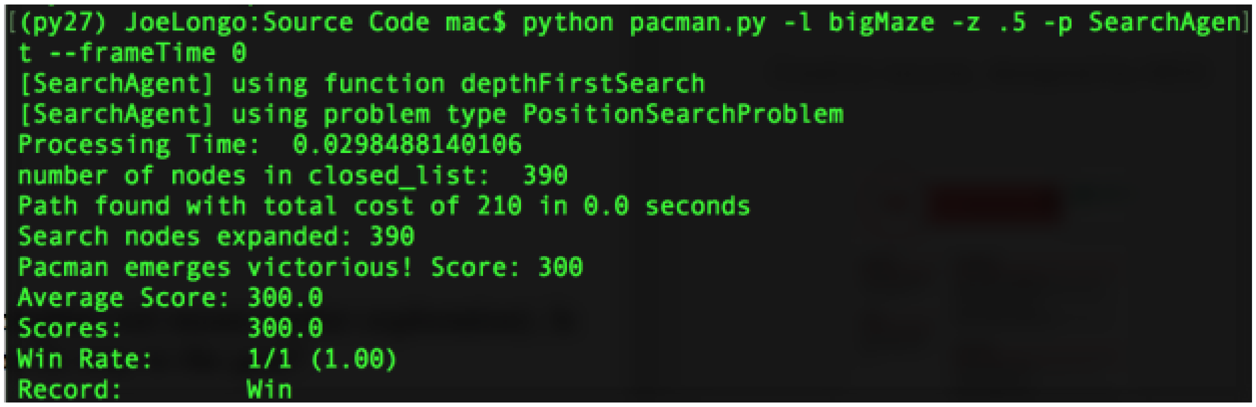
Besides, if we use DFS while iteratively increasing the threshold of maximum depth until we reach the goal, the threshold will be the optimal solution. Such algorithm is called Iterative Deepening Search.



```
(py27) JoeLongo:Source Code mac$ python pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Processing Time:  0.00155806541443
number of nodes in closed_list:  15
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
```



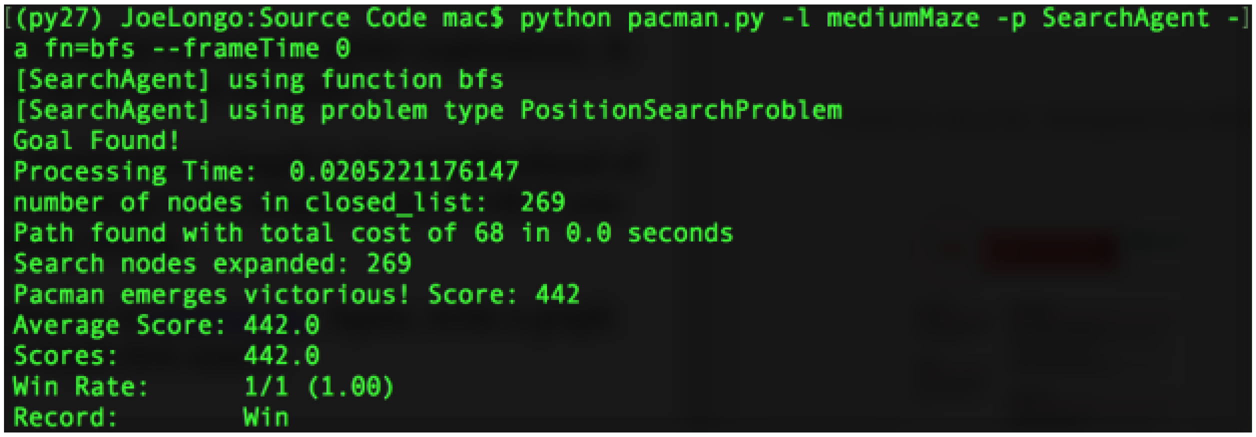
```
(py27) JoeLongo:Source Code mac$ python pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Processing Time:  0.00971412658691
number of nodes in closed_list:  146
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```



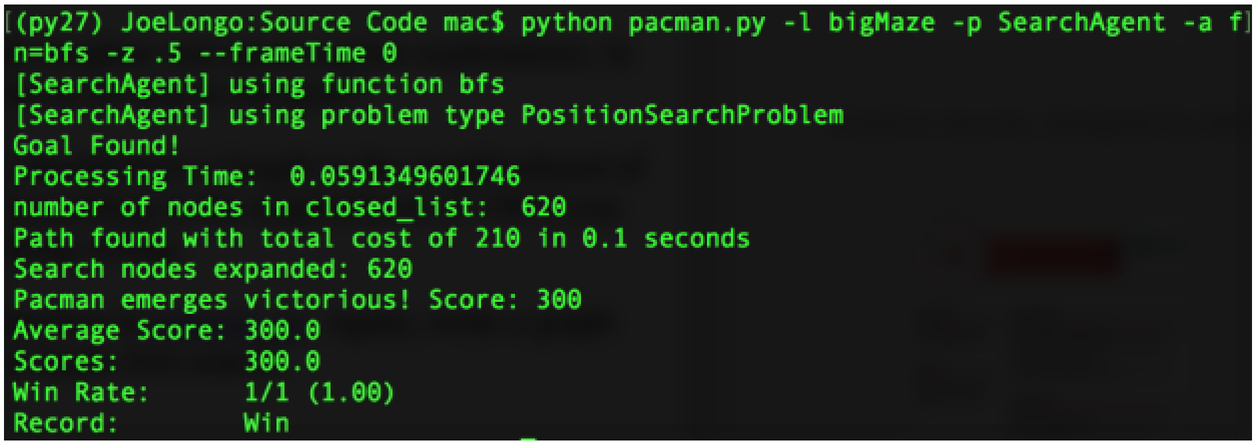
```
(py27) JoeLongo:Source Code mac$ python pacman.py -l bigMaze -z .5 -p SearchAgen
t --frameTime 0
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Processing Time:  0.0298488140106
number of nodes in closed_list:  390
Path found with total cost of 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
```

Question 2: Breadth First Search:

- **Data structure**: Queue (FIFO)
- **Complete**: Yes, as long as there is a path from root to goal with finite depth.
- **Optimal**: BFS expands nodes at a given depth before nodes whose depth is higher. Thus, if all actions have the same cost, the answer is Yes. If each action has a different cost, the answer is No and in order to resolve this problem, we use Uniform Cost Search.
  Both visited list and closed list are good to reach optimal result.

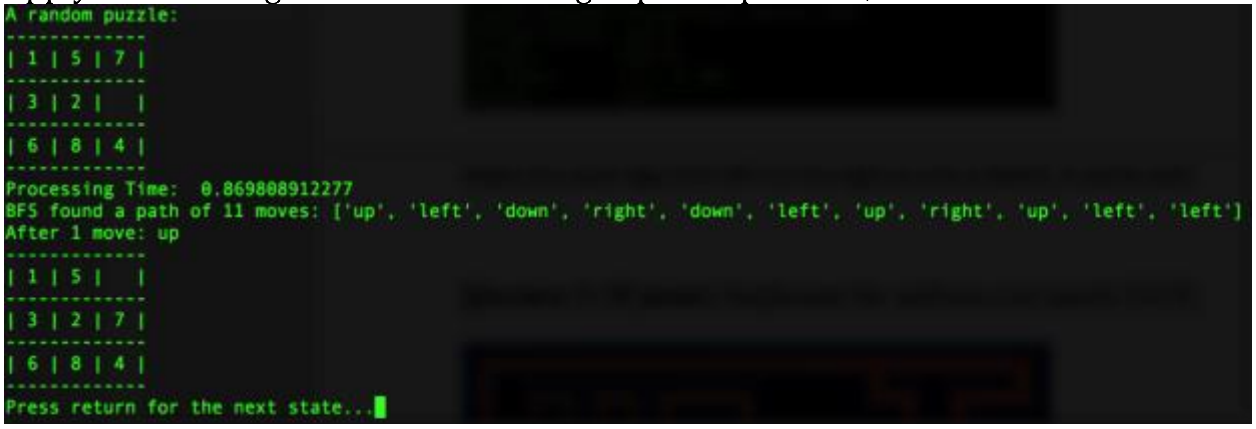


```
[(py27) JoeLongo:Source Code mac$ python pacman.py -l mediumMaze -p SearchAgent -]
a fn=bfs --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Goal Found!
Processing Time:  0.0205221176147
number of nodes in closed_list:  269
Path found with total 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
```

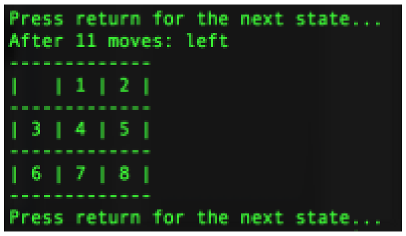


```
[(py27) JoeLongo:Source Code mac$ python pacman.py -l bigMaze -p SearchAgent -a f]
n=bfs -z .5 --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Goal Found!
Processing Time:  0.0591349601746
number of nodes in closed_list:  620
Path found with total 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
```

Apply the exact algorithm BFS to the eight-puzzle problem, it works well.



```
A random puzzle:
-------------
| 1 | 5 | 7 |
-------------
| 3 | 2 |   |
-------------
| 6 | 8 | 4 |
-------------
Processing Time:  0.869808912277
BFS found a path of 11 moves: ['up', 'left', 'down', 'right', 'down', 'left', 'up', 'right', 'up', 'left', 'left']
After 1 move: up
-------------
| 1 | 5 |   |
-------------
| 3 | 2 | 7 |
-------------
| 6 | 8 | 4 |
-------------
Press return for the next state...
```



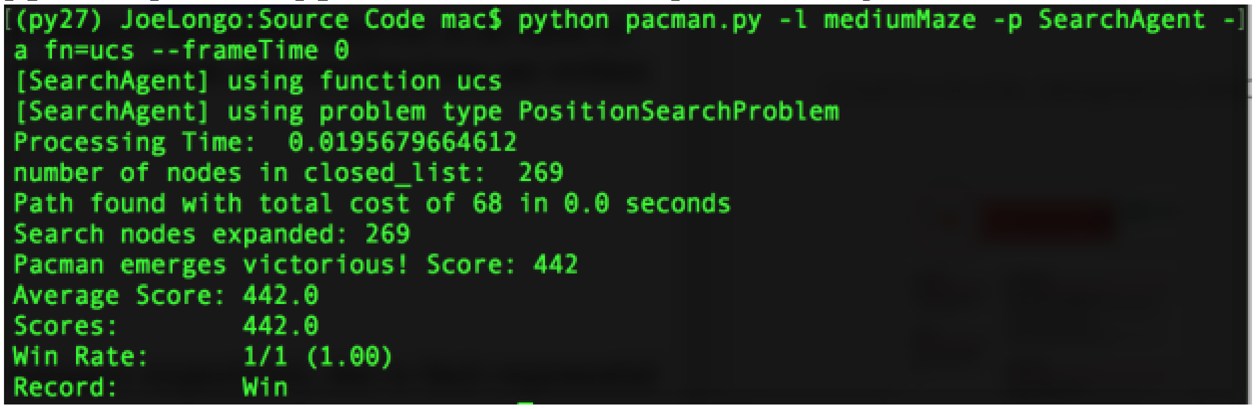
```
Press return for the next state...
After 11 moves: left
-------------
|   | 1 | 2 |
-------------
| 3 | 4 | 5 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
```

Question 3: Uniform Cost Search:

- **Data structure**: Priority Queue
- **Complete**: Yes, as long as the graph is finite and all actions have positive cost.

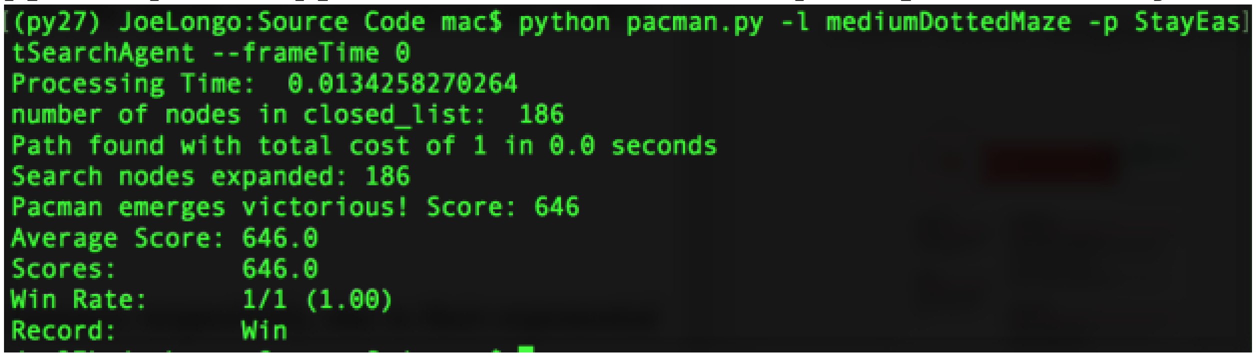
- **Optimal**: This is used for the problem where all actions have different cost. Technically, UCS expands the node whose path to the root has minimum cost. In other words, it expands node which cannot be optimized if all costs are positive.
  Importantly, we cannot use visited list in this algorithm because it skips every node to be updated. In other words, if we find a path from root to node x with cost y previously and find another path from root to node x with cost z < y, we cannot perform any update; hence, the algorithm fails to find an optimal solution.

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



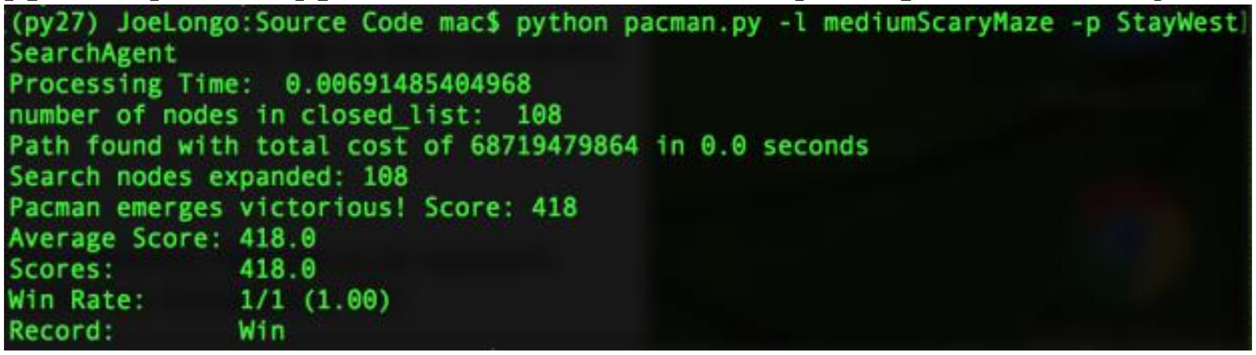
```
(py27) JoeLongo:Source Code mac$ python pacman.py -l mediumMaze -p SearchAgent -
a fn=ucs --frameTime 0
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Processing Time:  0.0195679664612
number of nodes in closed_list:  269
Path found with total cost of 68 in 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
```

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



```
(py27) JoeLongo:Source Code mac$ python pacman.py -l mediumDottedMaze -p StayEas
tSearchAgent --frameTime 0
Processing Time:  0.0134258270264
number of nodes in closed_list:  186
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
```

  `python pacman.py -l mediumDottedMaze -p StayWestSearchAgent`



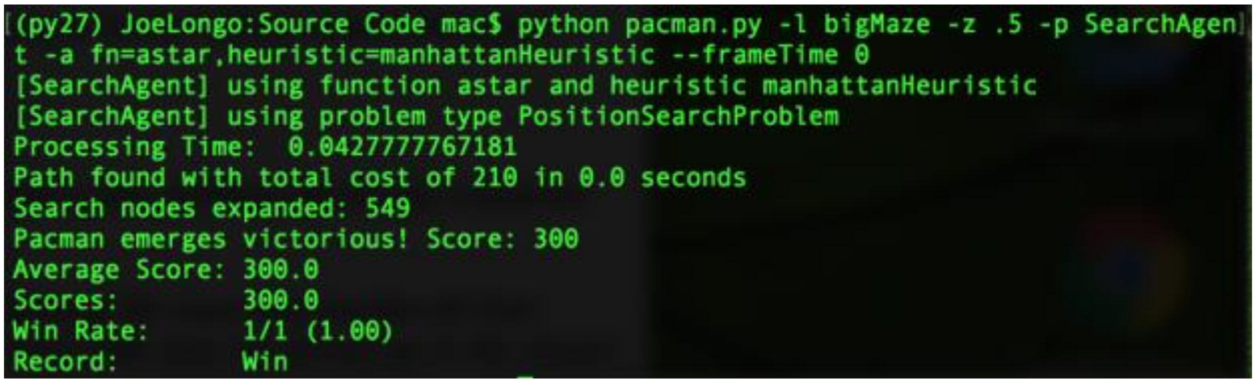
```
(py27) JoeLongo:Source Code mac$ python pacman.py -l mediumScaryMaze -p StayWest
SearchAgent
Processing Time:  0.00691485404968
number of nodes in closed_list:  108
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 4: A* Search:

- Almost as same as Uniform Cost Search except that it seeks the best f(n)=g(n)+h(n) instead of the best g(n), where g(n) is the actual smallest path from root to current node n and h(n) represents an estimated cost of reaching the goal.
- **Optimal**:
  . Admissibility: If h(n) <= the cost to reach the goal from n. **Optimal for A* tree.**

. **Consistency**: If h(n) <= c(n,a,n') + h(n'), where n' is a successor of n and c(n,a,n') is the cost from n to n'. Because we can prove f(n) <= f(n'), so A* is **Optimal for the original graph**.

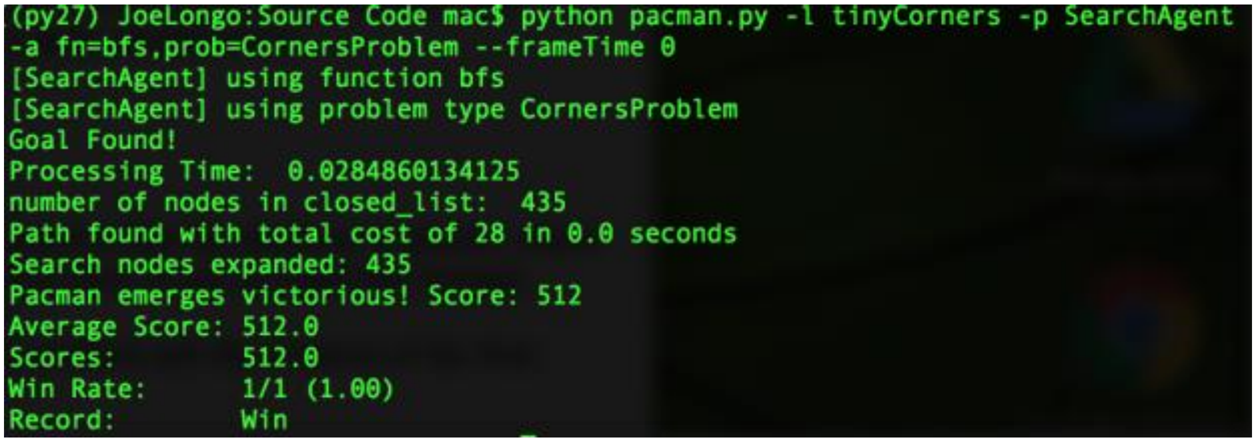
- If our heuristic is good, the A* search can significantly outperform UCS in big test case. Need to say that we are only considering the problem where all actions have different cost, and DFS and BFS cannot be used.



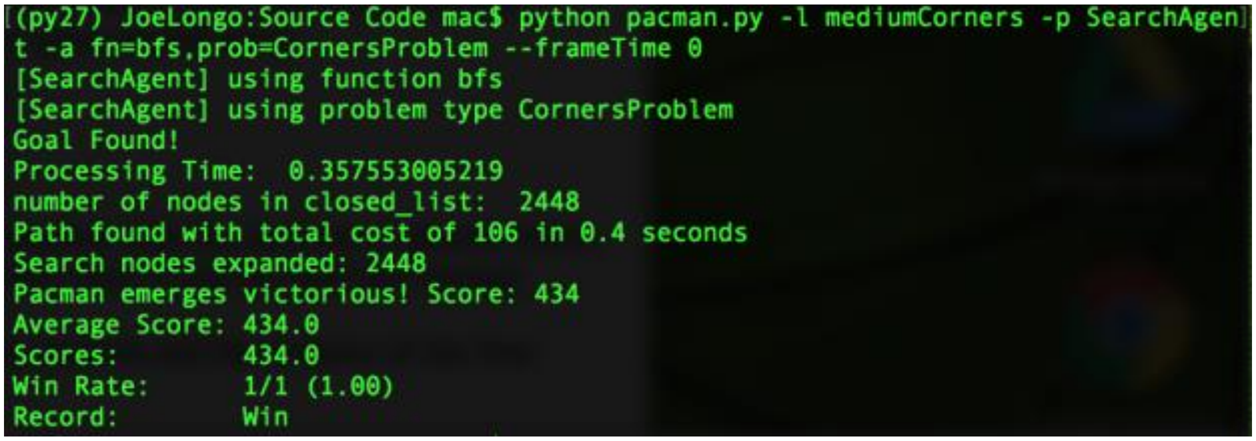
```
(py27) JoeLongo:Source Code mac$ python pacman.py -l bigMaze -z .5 -p SearchAgen
t -a fn=astar,heuristic=manhattanHeuristic --frameTime 0
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Processing Time:  0.0427777767181
Path 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
```

Question 5: BFS and Modify Goal-State for Corner-Problem

- The bfs solve the tinyCorners in 28 steps with 435 nodes expanded.
- The bfs solve the mediumCorners in 106 steps with 2448 nodes expanded.
- Because BFS expands all low-depth nodes before high-depth node, BFS (and also UCS) should not be utilized in big problem.



```
(py27) JoeLongo:Source Code mac$ python pacman.py -l tinyCorners -p SearchAgent
-a fn=bfs,prob=CornersProblem --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Goal Found!
Processing Time:  0.0284860134125
number of nodes in closed_list:  435
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 435
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:        512.0
Win Rate:      1/1 (1.00)
Record:        Win
```



```
(py27) JoeLongo:Source Code mac$ python pacman.py -l mediumCorners -p SearchAgen
t -a fn=bfs,prob=CornersProblem --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Goal Found!
Processing Time:  0.357553005219
number of nodes in closed_list:  2448
Path found with total cost of 106 in 0.4 seconds
Search nodes expanded: 2448
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Question 6: A* for Corner-Problem

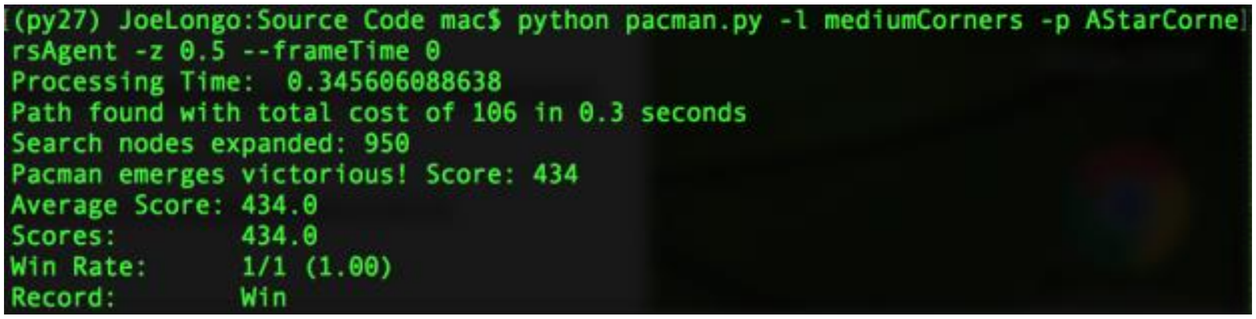
- Let's call c1 c2 c3 c4 are 4 corners and M(a,b) is manhattan distance from node a to b. Our heuristic function is defined as follow:

$$h(n) = \min \sum_{\substack{i_a \in \{1,2,3,4\} \\ i_a \neq i_b \text{ if } a \neq b}} M(n, c_{i_1}) + M(n, c_{i_2}) + M(n, c_{i_3}) + M(n, c_{i_4})$$

- h(n) will be modified in case we reach a number of corners. For example, if we already reached corner #4, the heuristic function is adjusted as follow:

$$h(n) = \min \sum_{\substack{i_a \in \{1,2,3\} \\ i_a \neq i_b \text{ if } a \neq b}} M(n, c_{i_1}) + M(n, c_{i_2}) + M(n, c_{i_3})$$

- Obviously, the proposed heuristic function above is admissible, since manhattan distance is the most optimistic distance to each corner (must not exceed the actual cost) and we still have to reach all corners.



```
[(py27) JoeLongo:Source Code mac$ python pacman.py -l mediumCorners -p AStarCorne]
rsAgent -z 0.5 --frameTime 0
Processing Time:  0.345606088638
Path found with total cost of 106 in 0.3 seconds
Search nodes expanded: 950
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
```

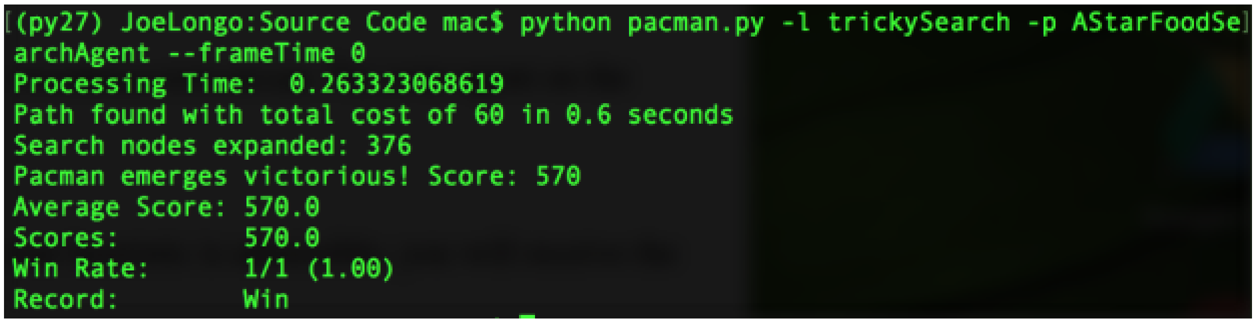
Question 7: A* for All-Food-Problem

- At one state, assume we have at least two food pieces left. We define:

$$h(n) = Q(n,x) + Q(x,y)$$

with x, y are remaining nodes that furthest away from each other and Q(a,b) is actual min cost from a to b (precomputed using Floyd algorithm)

- Obviously, the proposed heuristic is admissible because we still have to visit these two nodes and Q(a,b) is the shortest cost from x to y. This Q(a,b) function, precomputed using Floyd algorithm, is definitely much better than the manhattan distance.
- Handle the state where only 1 food piece left.
- We tried the case where x and y as furthest point away from n, but the number of expanded nodes is higher.



```
[(py27) JoeLongo:Source Code mac$ python pacman.py -l trickySearch -p AStarFoodSe]
archAgent --frameTime 0
Processing Time:  0.263323068619
Path found with total cost of 60 in 0.6 seconds
Search nodes expanded: 376
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```