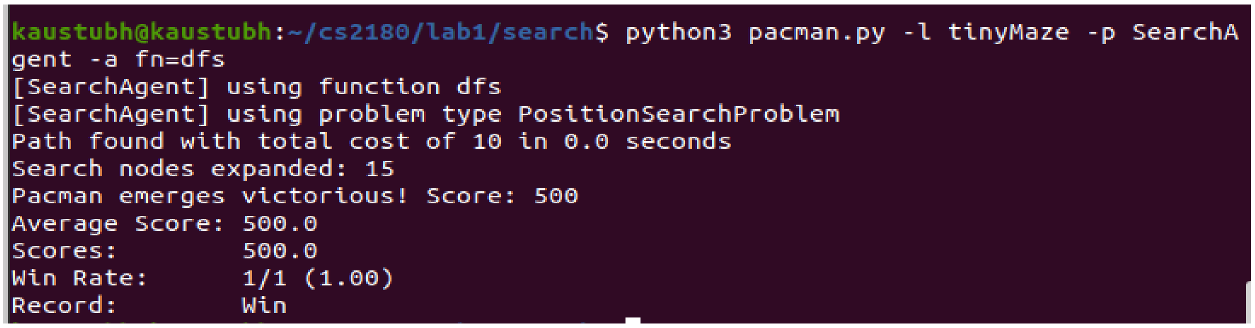
```
# search.py
# ---------
# Licensing Information:  You are free to use or extend these projects for
# educational purposes provided that (1) you do not distribute or publish
# solutions, (2) you retain this notice, and (3) you provide clear
# attribution to UC Berkeley, including a link to http://ai.berkeley.edu.
#
# Attribution Information: The Pacman AI projects were developed at UC
Berkeley.
# The core projects and autograders were primarily created by John DeNero
# (denero@cs.berkeley.edu) and Dan Klein (klein@cs.berkeley.edu).
# Student side autograding was added by Brad Miller, Nick Hay, and
# Pieter Abbeel (pabbeel@cs.berkeley.edu).


"""
In search.py, you will implement generic search algorithms which are
called by
Pacman agents (in searchAgents.py).
"""

import queue
from shutil import move
from numpy import append
import util

class SearchProblem:
   """
   This class outlines the structure of a search problem, but doesn't
implement
   any of the methods (in object-oriented terminology: an abstract class).

   You do not need to change anything in this class, ever.
   """

   def getStartState(self):
       """
       Returns the start state for the search problem.
       """
       util.raiseNotDefined()
   def isGoalState(self, state):
       """
         state: Search state

       Returns True if and only if the state is a valid goal state.
       """
       util.raiseNotDefined()

   def getSuccessors(self, state):
       """
         state: Search state

       For a given state, this should return a list of triples,
(successor,
       action, stepCost), where 'successor' is a successor to the current
       state, 'action' is the action required to get there, and 'stepCost'
is
       the incremental cost of expanding to that successor.
       """
       util.raiseNotDefined()

   def getCostOfmoves(self, moves):
       """
        moves: A list of moves to take

       This method returns the total cost of a particular sequence of
moves.
       The sequence must be composed of legal moves.
       """
       util.raiseNotDefined()


def tinyMazeSearch(problem):
   """
   Returns a sequence of moves that solves tinyMaze.  For any other maze,
the
   sequence of moves will be incorrect, so only use this for tinyMaze.
   """
   from game import next_moves
```

```python
    s = next_moves.SOUTH
    w = next_moves.WEST
    return  [s, s, w, s, w, w, s, w]

def depthFirstSearch(problem: SearchProblem):
    """
    Search the deepest nodes in the search tree first.

    Your search algorithm needs to return a list of moves that reaches the
    goal. Make sure to implement a graph search algorithm.

    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:

    print("Start:", problem.getStartState())
    print("Is the start a goal?",
problem.isGoalState(problem.getStartState()))
    print("Start's successors:",
problem.getSuccessors(problem.getStartState()))
    """
    "*** YOUR CODE HERE ***"
    from util import Stack
    start = problem.getStartState()
    curr = problem.getStartState()
    visited = []
    visited.append(start)
    stk = Stack()
    state_dir = (start, [])
    stk.push(state_dir)
    while (stk.isEmpty() == False) :
        if (problem.isGoalState(curr) == True):
            break
        state = stk.pop()
        current = state[0]
        moves  = state[1]
        visited.append(current)
        successor = problem.getSuccessors(current)
        for succ in successor:
            next = succ[0]
            if next in visited:
                continue
            else:
                curr = succ[0]
                list = []
                next_move = succ[1]
                list.append(next_move)
                upd_moves = moves + list
                stk.push((next, upd_moves))
    return upd_moves

def breadthFirstSearch(problem: SearchProblem):
    """Search the shallowest nodes in the search tree first."""
    "*** YOUR CODE HERE ***"
    from util import Queue
    start = problem.getStartState()
    curr = problem.getStartState()
    visited = []
    visited.append(start)
    queue = Queue()
    state_dir = (start, [])
    queue.push(state_dir)
    while (queue.isEmpty() == False) :
        state = queue.pop()
        current = state[0]
        moves  = state[1]
        if (problem.isGoalState(current) == True):
            return moves
        successor = problem.getSuccessors(current)
        for succ in successor:
            next = succ[0]
            if next in visited:
                continue
            else:
                curr = succ[0]
                next_move = succ[1]
                visited.append(next)
                list = []
                list.append(next_move)
                upd_moves = moves + list
                queue.push((next, upd_moves))
    return moves
    util.raiseNotDefined()

def uniformCostSearch(problem: SearchProblem):
    """Search the node of least total cost first."""
    "*** YOUR CODE HERE ***"
    from util import PriorityQueue
    start = problem.getStartState()
    visited = []
    PQueue = PriorityQueue()
    PQueue.push((start, []) ,0)
    while (PQueue.isEmpty() == False):
        state = PQueue.pop()
        current = state[0]
        moves = state[1]
        if problem.isGoalState(current):
            return moves
        if current in visited:
            continue
        else:
            successors = problem.getSuccessors(current)
            for succ in successors:
                next = succ[0]
                if next not in visited:
                    next_move = succ[1]
                    upd_moves = moves + [next_move]
                    PQueue.push((next, upd_moves),
problem.getCostOfActions(upd_moves))
        visited.append(current)
    return moves
    util.raiseNotDefined()

def nullHeuristic(state, problem=None):
    """
    A heuristic function estimates the cost from the current state to the
nearest
    goal in the provided SearchProblem.  This heuristic is trivial.
    """
    return 0
```

```
def aStarSearch(problem: SearchProblem, heuristic=nullHeuristic):
   """Search the node that has the lowest combined cost and heuristic
first."""
   "*** YOUR CODE HERE ***"
   from util import PriorityQueue
   start = problem.getStartState()
   visited = []
   PQueue = PriorityQueue()
   PQueue.push((start, []), nullHeuristic(start, problem))
   nCost = 0
   while (PQueue.isEmpty() == False):
       state = PQueue.pop()
       current = state[0]
       moves = state[1]
       if problem.isGoalState(current):
           return moves
       if current in visited:
           continue
       else:
           successors = problem.getSuccessors(current)
           for succ in successors:
               next = succ[0]
               if next not in visited:
                   next_move = succ[1]
                   nmoves = moves + [next_move]
                   nCost = problem.getCostOfActions(nmoves) +
heuristic(next, problem)
                   PQueue.push((next, moves + [next_move]), nCost)
       visited.append(current)
   return moves
   util.raiseNotDefined()


# Abbreviations
bfs = breadthFirstSearch
dfs = depthFirstSearch
astar = aStarSearch
ucs = uniformCostSearch
```

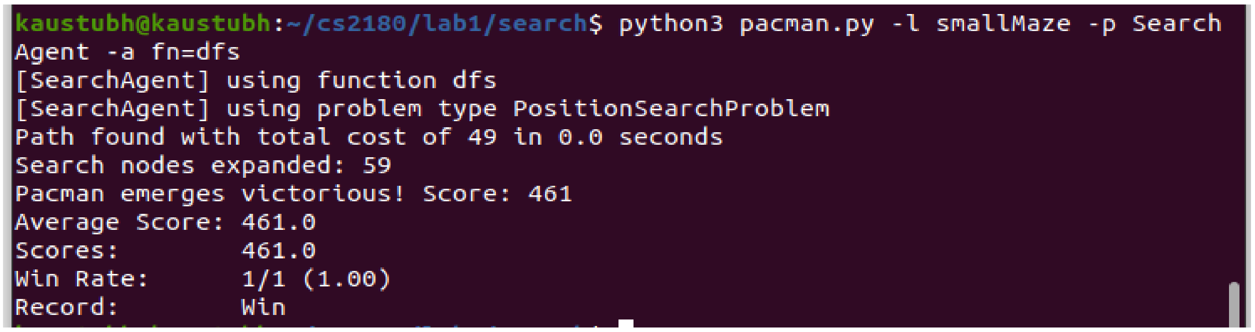
Q1 DFS

python3 pacman.py -l tinyMaze -p SearchAgent -a fn=dfs



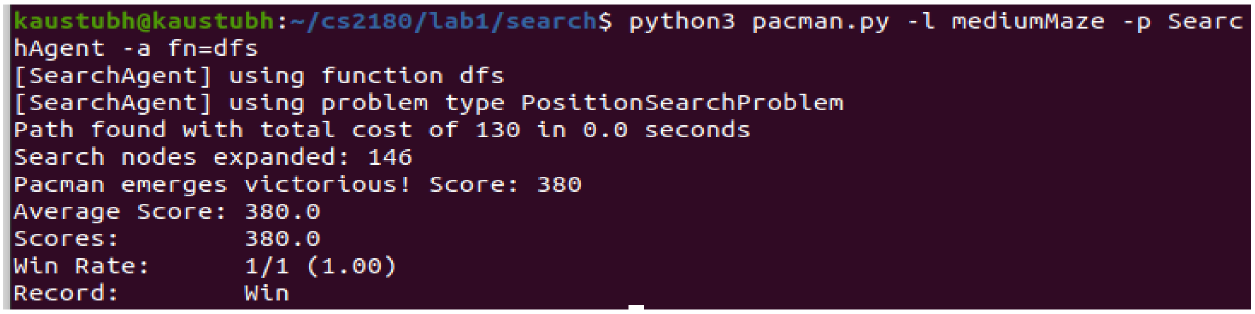
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l tinyMaze -p SearchA
gent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l smallMaze -p SearchAgent -a fn=dfs



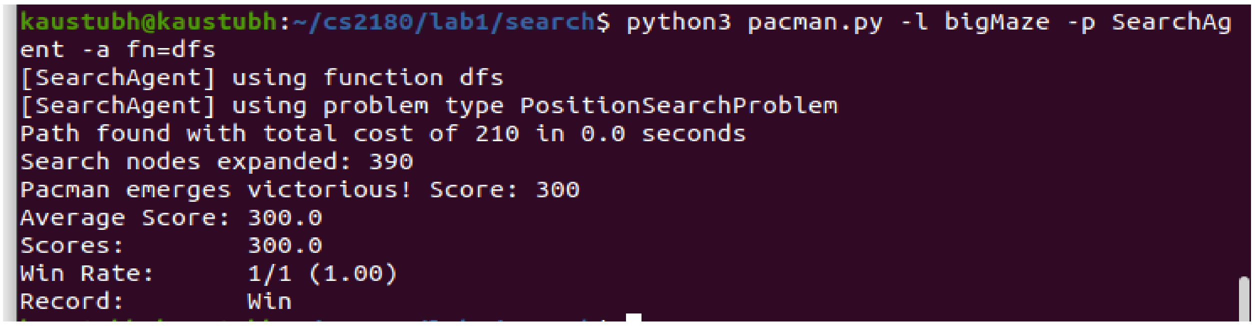
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l smallMaze -p Search
Agent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 49 in 0.0 seconds
Search nodes expanded: 59
Pacman emerges victorious! Score: 461
Average Score: 461.0
Scores:        461.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l mediumMaze -p SearchAgent -a fn=dfs



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l mediumMaze -p Searc
hAgent -a fn=dfs
[SearchAgent] using function dfs
[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
```

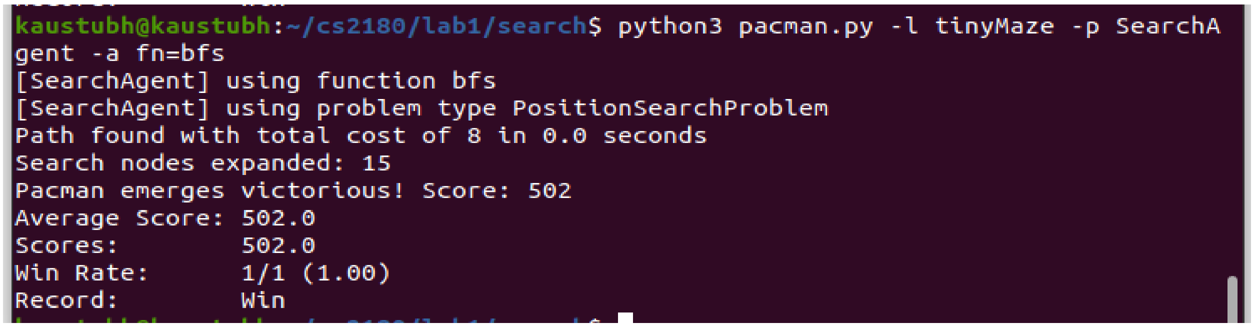
python3 pacman.py -l bigMaze -p SearchAgent -a fn=dfs



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l bigMaze -p SearchAg
ent -a fn=dfs
[SearchAgent] using function dfs
[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
```

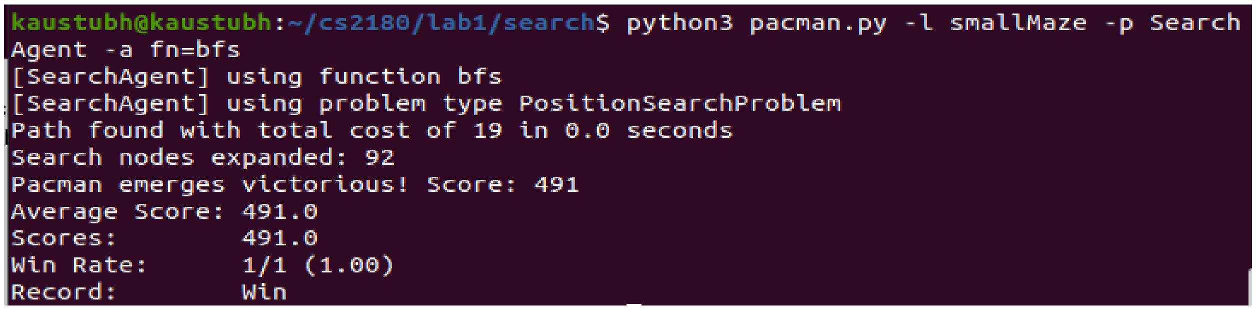
Q2 BFS

python3 pacman.py -l tinyMaze -p SearchAgent -a fn=bfs



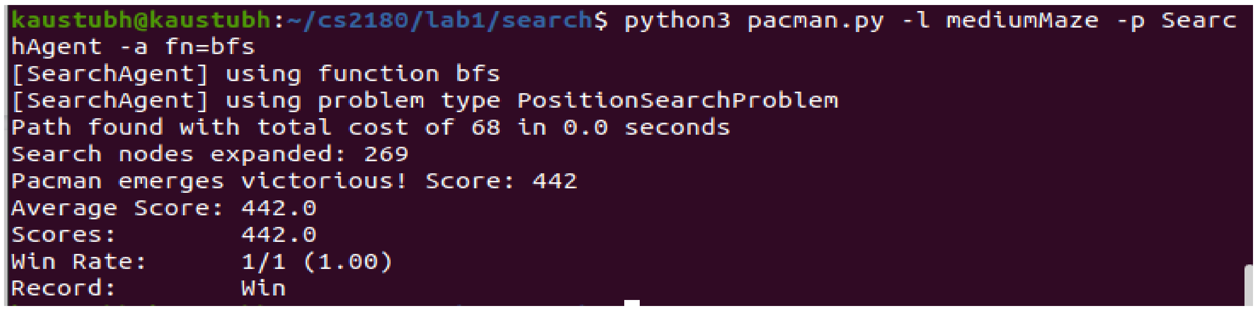
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l tinyMaze -p SearchA
gent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l smallMaze -p SearchAgent -a fn=bfs



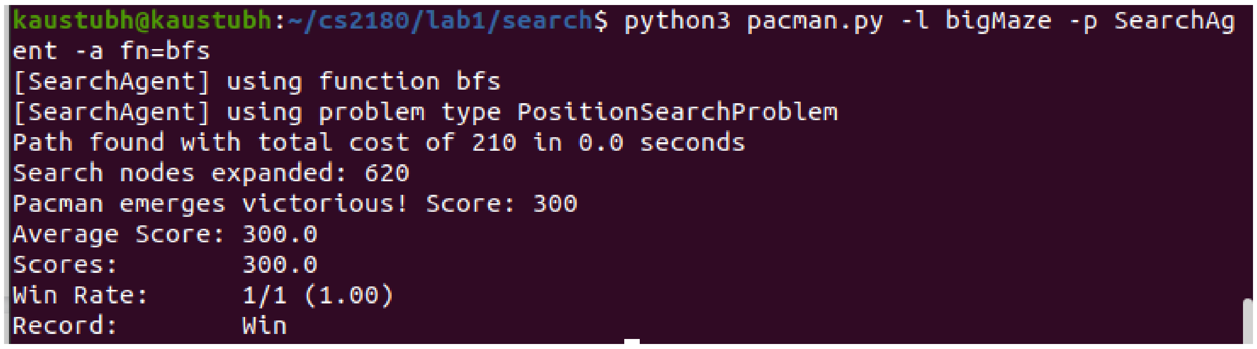
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l smallMaze -p Search
Agent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 19 in 0.0 seconds
Search nodes expanded: 92
Pacman emerges victorious! Score: 491
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l mediumMaze -p SearchAgent -a fn=bfs



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l mediumMaze -p Searc
hAgent -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
```

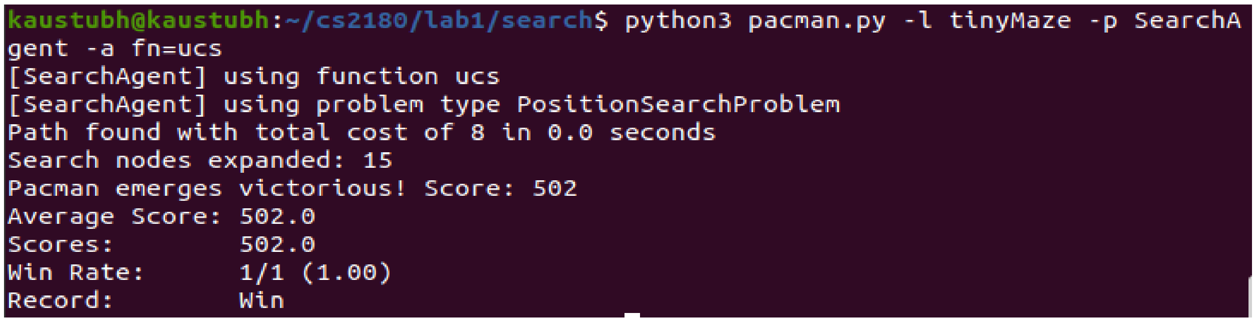
python3 pacman.py -l bigMaze -p SearchAgent -a fn=bfs



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l bigMaze -p SearchAg
ent -a fn=bfs
[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
```

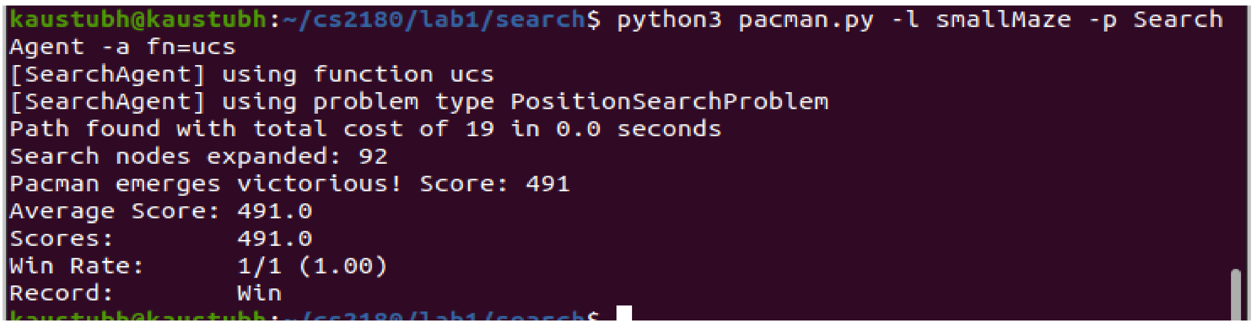
Q3 Uniform Cost Search

python3 pacman.py -l tinyMaze -p SearchAgent -a fn=ucs



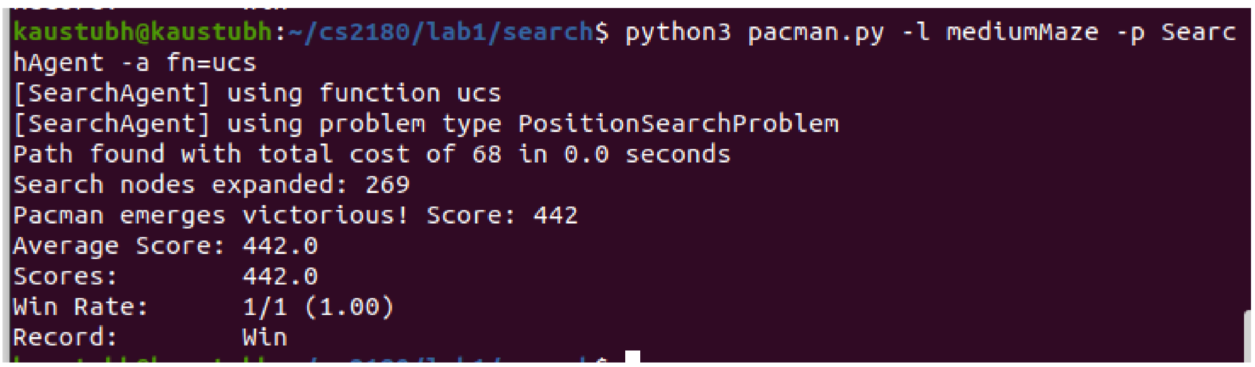
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l tinyMaze -p SearchA
gent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l smallMaze -p SearchAgent -a fn=ucs



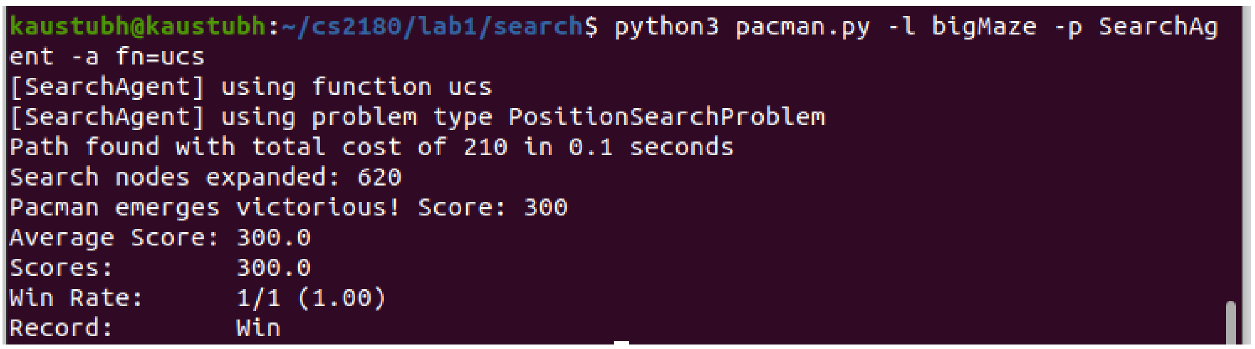
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l smallMaze -p Search
Agent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 19 in 0.0 seconds
Search nodes expanded: 92
Pacman emerges victorious! Score: 491
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l mediumMaze -p Searc
hAgent -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
```

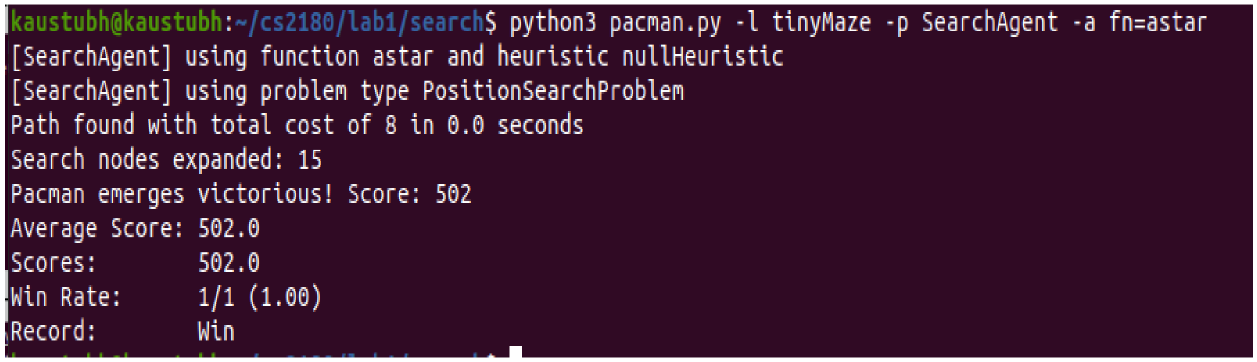
python3 pacman.py -l bigMaze -p SearchAgent -a fn=ucs



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l bigMaze -p SearchAg
ent -a fn=ucs
[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
```

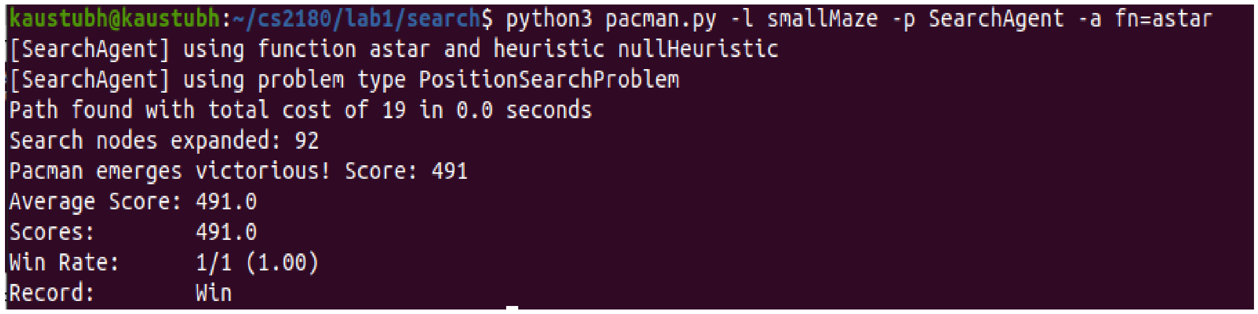
Q4 aStart search

python3 pacman.py -l tinyMaze -p SearchAgent -a fn=astar



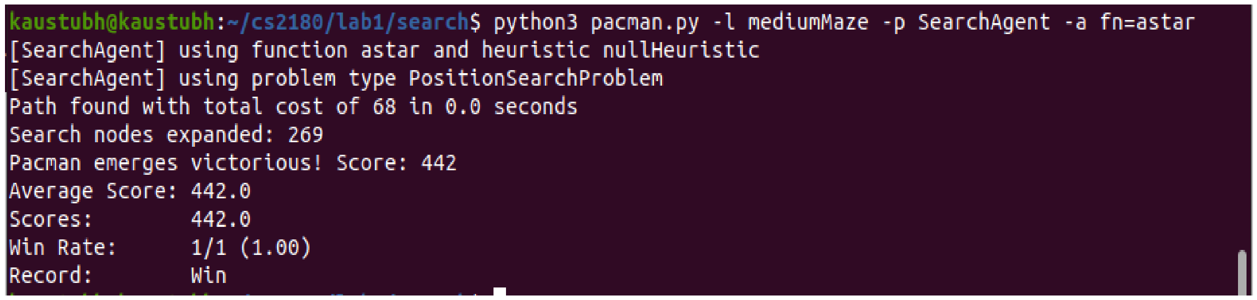
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l tinyMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l smallMaze -p SearchAgent -a fn=astar



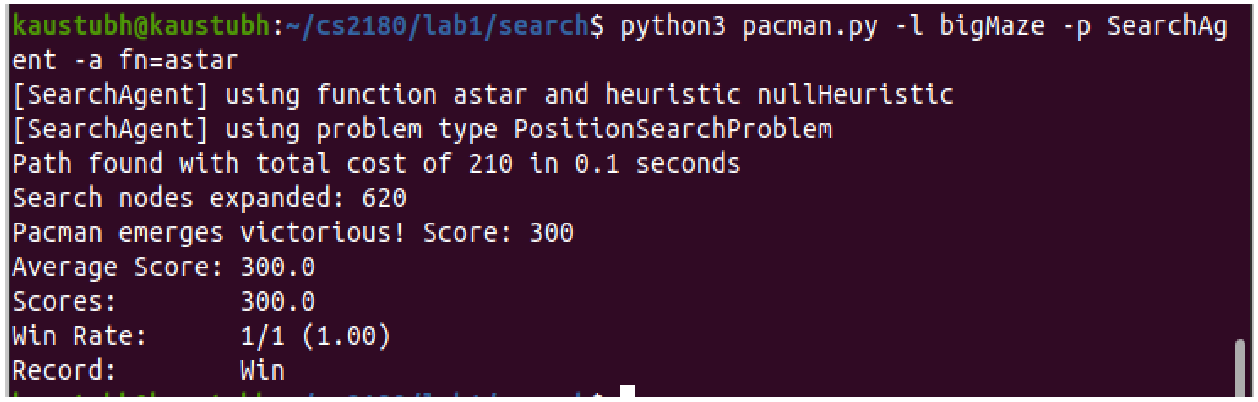
```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l smallMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 19 in 0.0 seconds
Search nodes expanded: 92
Pacman emerges victorious! Score: 491
Average Score: 491.0
Scores:        491.0
Win Rate:      1/1 (1.00)
Record:        Win
```

python3 pacman.py -l mediumMaze -p SearchAgent -a fn=astar



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l mediumMaze -p SearchAgent -a fn=astar
[SearchAgent] using function astar 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
```

python3 pacman.py -l bigMaze -p SearchAgent -a fn=astar



```
kaustubh@kaustubh:~/cs2180/lab1/search$ python3 pacman.py -l bigMaze -p SearchAg
ent -a fn=astar
[SearchAgent] using function astar and heuristic nullHeuristic
[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
```