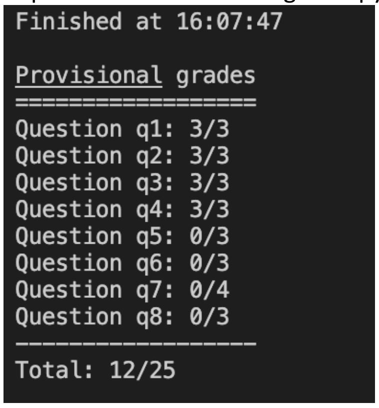
COSE361-03 assignment 1
컴퓨터학과 2022320006 최진혁

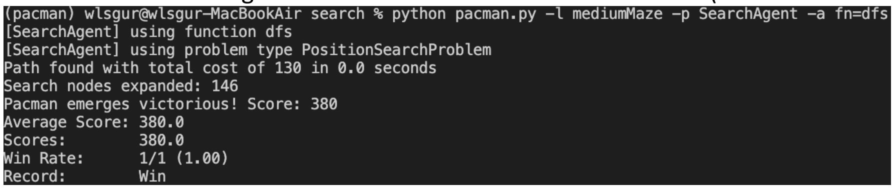
a. Capture the result of autograder.py in terminal



```
Finished at 16:07:47

Provisional grades
==================
Question q1: 3/3
Question q2: 3/3
Question q3: 3/3
Question q4: 3/3
Question q5: 0/3
Question q6: 0/3
Question q7: 0/4
Question q8: 0/3
------------------
Total: 12/25
```

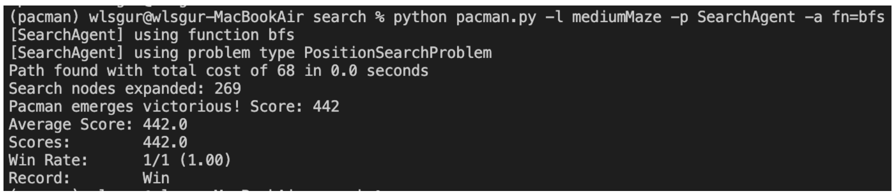
b.

b-1. Discuss which algorithm is better between DFS or BFS. (In mediumMaze)



```
(pacman) wlsgur@wlsgur-MacBookAir search % python pacman.py -l mediumMaze -p SearchAgent -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
```

→ DFS result



```
(pacman) wlsgur@wlsgur-MacBookAir 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 0.0 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
```

→ BFS result

As shown in the capture, DFS is better in mediumMaze. Because DFS expanded 146 nodes, while BFS expanded 269 nodes.

b-2. The proposed heuristic function is Manhattan. Is there any other heuristic function that is more efficient? Discuss specific cases where your algorithm works effectively.

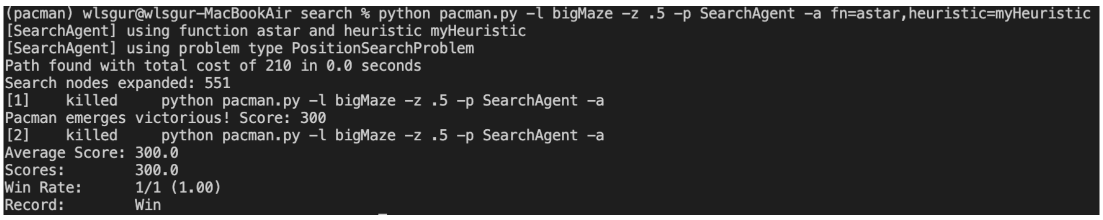
I think the average of Manhattan and Euclidean can be a one of the possible options. Because sometimes such as the path is simple (like if there is a path just go down or left in mediumMaze), Manhattan is better. However, the path is more complicated, both are not good. So just add Manhattan and Euclidean and divide by 2, we can simply get another heuristic function (Let say f1).

There is another heuristic function, which is the multiple of f1 and the cosine similarity of current direction and the direction to the goal(f2). This also can reflect the direction to the goal position.

b-3. Is it possible to implement the DFS with recursion in thin pacman case?

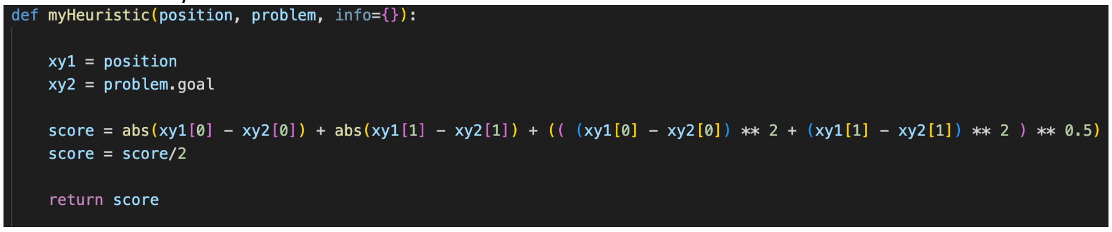
I think it is not possible if we can not change the function's declaration, because the depthFirstSearch in search.py has only problem as a parameter. For the recursion, node and list of actions should be a parameter as well. I think it works if we can modify the function's parameter.

c. (IF YOU IMPLEMENT ADDITIONAL QUESTION) Capture the result of A* algorithm with your i mplemented heuristic function.



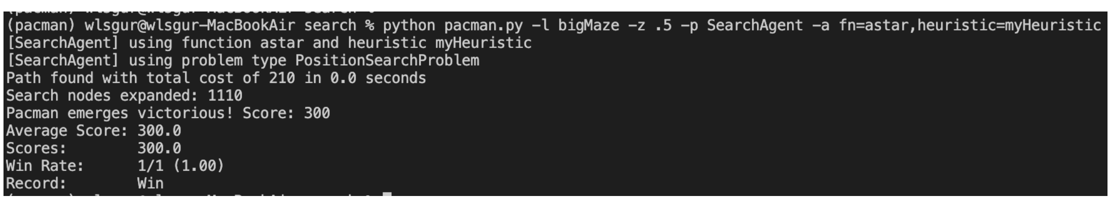
```
(pacman) wlsgur@wlsgur-MacBookAir search % python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=myHeuristic
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 551
[1]    killed     python pacman.py -l bigMaze -z .5 -p SearchAgent -a
Pacman emerges victorious! Score: 300
[2]    killed     python pacman.py -l bigMaze -z .5 -p SearchAgent -a
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

→ result with myHeuristic



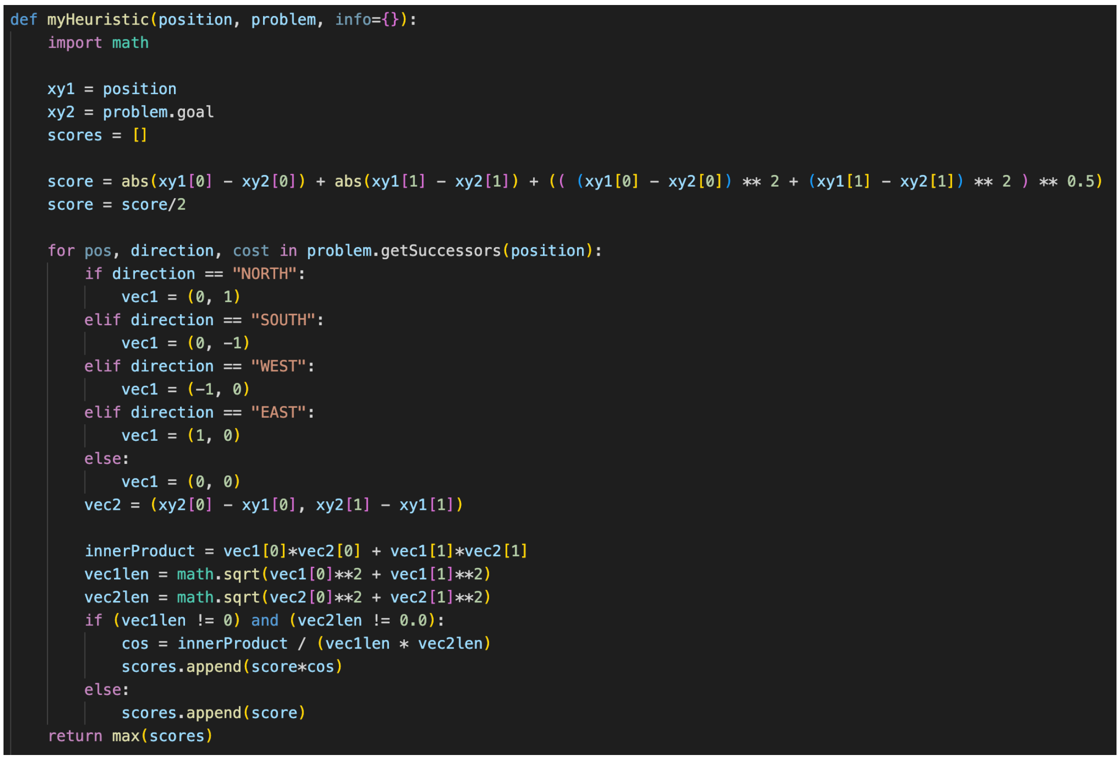
```python
def myHeuristic(position, problem, info={}):

    xy1 = position
    xy2 = problem.goal

    score = abs(xy1[0] - xy2[0]) + abs(xy1[1] - xy2[1]) + (( (xy1[0] - xy2[0]) ** 2 + (xy1[1] - xy2[1]) ** 2 ) ** 0.5)
    score = score/2

    return score
```

With above myHeuristic.(f1)



```
(pacman) wlsgur@wlsgur-MacBookAir search % python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=myHeuristic
[SearchAgent] using function astar and heuristic myHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 1110
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

→another result



```python
def myHeuristic(position, problem, info={}):
    import math

    xy1 = position
    xy2 = problem.goal
    scores = []

    score = abs(xy1[0] - xy2[0]) + abs(xy1[1] - xy2[1]) + (( (xy1[0] - xy2[0]) ** 2 + (xy1[1] - xy2[1]) ** 2 ) ** 0.5)
    score = score/2

    for pos, direction, cost in problem.getSuccessors(position):
        if direction == "NORTH":
            vec1 = (0, 1)
        elif direction == "SOUTH":
            vec1 = (0, -1)
        elif direction == "WEST":
            vec1 = (-1, 0)
        elif direction == "EAST":
            vec1 = (1, 0)
        else:
            vec1 = (0, 0)
        vec2 = (xy2[0] - xy1[0], xy2[1] - xy1[1])

        innerProduct = vec1[0]*vec2[0] + vec1[1]*vec2[1]
        vec1len = math.sqrt(vec1[0]**2 + vec1[1]**2)
        vec2len = math.sqrt(vec2[0]**2 + vec2[1]**2)
        if (vec1len != 0) and (vec2len != 0.0):
            cos = innerProduct / (vec1len * vec2len)
            scores.append(score*cos)
        else:
            scores.append(score)
    return max(scores)
```

With above one(f2).

F2 is not efficient, while f1 is not bad.