Pac-Man Final Report

Mahmoud Gamal 201601022

Moustafa Abada 201601212

Eslam Zaher 201600241

Introduction

In our project we will use the Pac-Man project developed for CS 188, one of Berkeley courses. This project supports us with required GUI for implementing the learned algorithms through the course. We will implement search algorithms like BFS,DFS,A* and UCF. In addition to adversarial search algorithms like minmax , alpha-beta pruning, and expectimax.

Metric :

The autograder is used to evaluate our algorithms. The autograder checks the code to determine whether it explores the correct number of game states. So, it is picky about how many times you call GameState.generateSuccessor.

Approach :

1. Project 1

In this project, we will examine different types of search algorithms such as BFS,DFS,A* and UCF. Also, we will define our cost function including heuristic functions and use these algorithms to reach specific goals such as Eating All Dots in less time and shortest path or Eating the closest dots.
The algorithms that we used are:

- BFS:
- DFS:
- UCF: Uniform Cost Function is implemented using a priority queue to give higher priority for a given object over the other. For example, areas where the ghost agents are located will be given a lower priority than areas with free ghosts.
- A*: A star algorithm is implemented using a priority queue also and uses a heuristic function to calculate the cost to the nearest goal.

The heuristic function takes search states and returns numbers that are used to calculate the cost of the path to the shortest goal. The heuristic function should be admissible meaning that it never overestimates the true cost to a nearest goal and consistent meaning that if an action or step has cost X, taking or going to that step, we should never overestimates the actual step cost.

These search algorithms will be used in these actions:

- Eating all dots: We will be using a for loop to iterate over the places of all dots and reach them using our search algorithms.

- Eating the closet dot: We won't use a for loop, instead, the pac-man agent will use the A* algorithm only and use a heuristic function to estimate the cost of reaching the nearest goal which in this case reaching for the closet dot.

Tasks that may also use the search algorithms:

- Finding four dots in the four corners: the algorithm BFS will be used by expanding the four possible paths and reaching for them in a shortest way- not necessarily starting with the closest point.

2. Project 2

In this project, the mission was to design agents suitable for the classic version of Pacman, including ghosts. Along the way, the project can be divided into two sub steps :

- The adversarial search algorithms used for the pacman to escape from the ghosts and collect all food.
- Improving the evaluation function used by adversarial search algorithms to get better solutions.

Adversarial search algorithms:

Minmax :

The minimax algorithm used in this project is more general than the usual minimax as it should work with any number of ghosts.
The modifications over the normal minimax:

1. The minimax tree has multiple min layers (one for each ghost) for every max layer.
2. The evaluation function -its contribution will be discussed at length in the second step.
3. The depth of the tree - cutoff function.

Alpha-beta pruning:

This algorithm is similar to the minimax algorithms with a small modification by adding the alpha-beta checking in max_value() and min_value() functions

Expectimax :

Here the agent, unlike minimax algorithms and alpha-beta pruning suppose that the ghosts are not behaving optimally so we use probabilistic behavior of agents who may make suboptimal choices. Monte carlo is usually used , but instead we used uniformly distributed probability to choose the coming actions of the ghost.

Improving evaluation function:

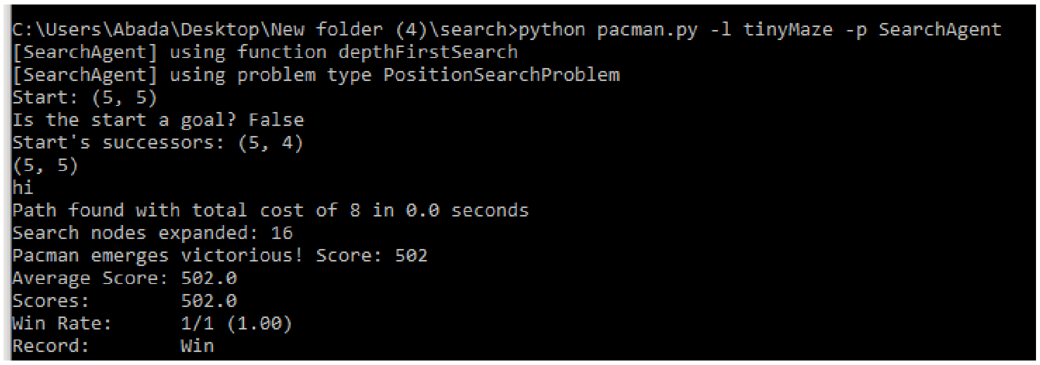
In our design of the reflex agent and the adversarial search algorithm, we used a conventional evaluation function that took into consideration two heuristic functions, the first one was the total distance to all ghosts, the second one was the distance to the nearest pellet of food. The evaluation function is designed with the reciprocal of these heuristic functions such that the nearer is the distance to the food the greater is the contribution to the score produced by the evaluation function, and the smaller the total distance to all ghosts, the more the score is reduced. In the design of an improved evaluation function, we included other terms such as the proximity to ghosts and the distance to large capsules(that cause the ghosts to scream and flee); the proximity heuristic acts as an alarm that although the total distance to all ghosts might be large, one ghost of them might be very close to the Pacman and hence there is a danger. We also put different weights on the heuristic functions to give for example large focus on escaping the ghosts and considering the capsules.

Findings:

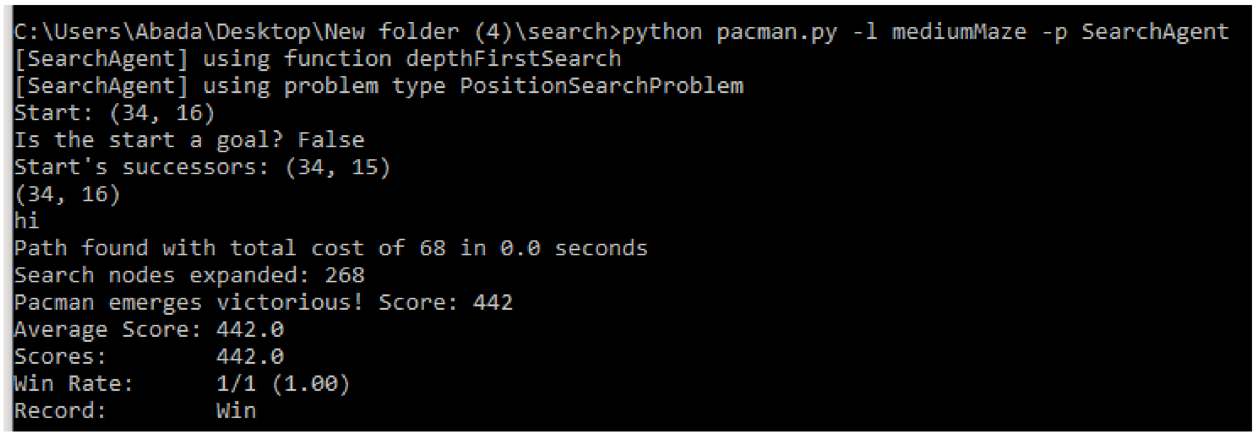
1. Project 1

For the task Eating all dots implementing using a for loop to iterate over the places of all dots is not efficient, instead, we should use the A* algorithm and the heuristic function that takes into consideration the closet dot only when estimating the cost of the shortest path. This method gave an optimal and complete solution with a shortest path. For the task of Eating dots at the corners: The algorithm DFS expands the possible paths and gives the optimal and complete solution.

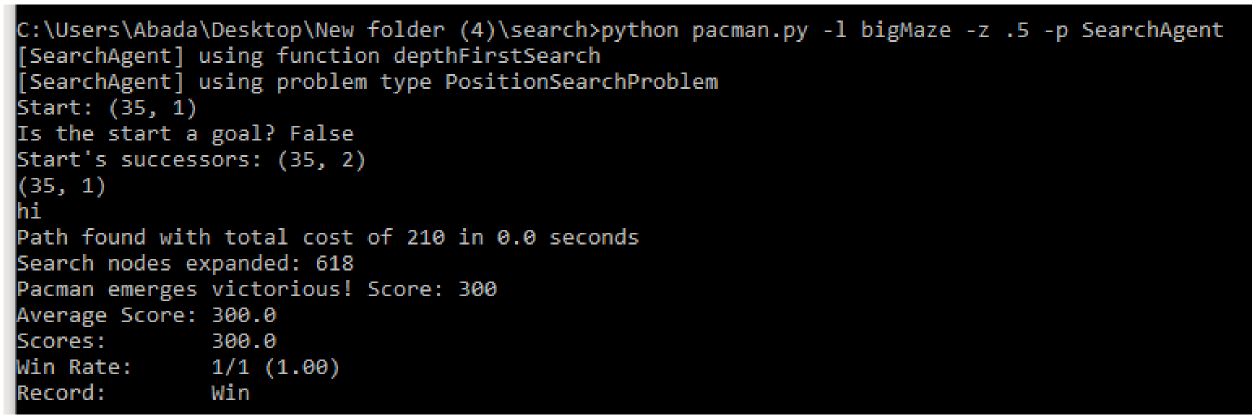
Questions 1 Autograder:



```
C:\Users\Abada\Desktop\New folder (4)\search>python pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Start: (5, 5)
Is the start a goal? False
Start's successors: (5, 4)
(5, 5)
hi
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 16
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```



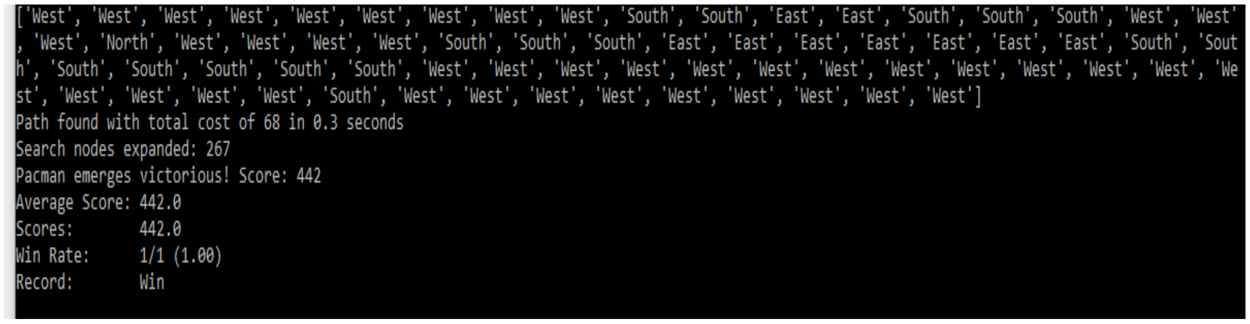
```
C:\Users\Abada\Desktop\New folder (4)\search>python pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Start: (34, 16)
Is the start a goal? False
Start's successors: (34, 15)
(34, 16)
hi
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 268
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```



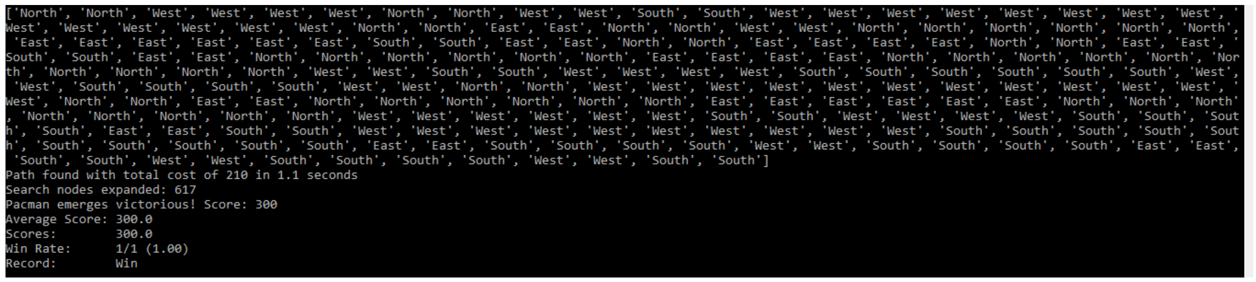
```
C:\Users\Abada\Desktop\New folder (4)\search>python pacman.py -l bigMaze -z .5 -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Start: (35, 1)
Is the start a goal? False
Start's successors: (35, 2)
(35, 1)
hi
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 618
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

DFS succeeded to solve the maze and reach the goal and expand any visited nodes.

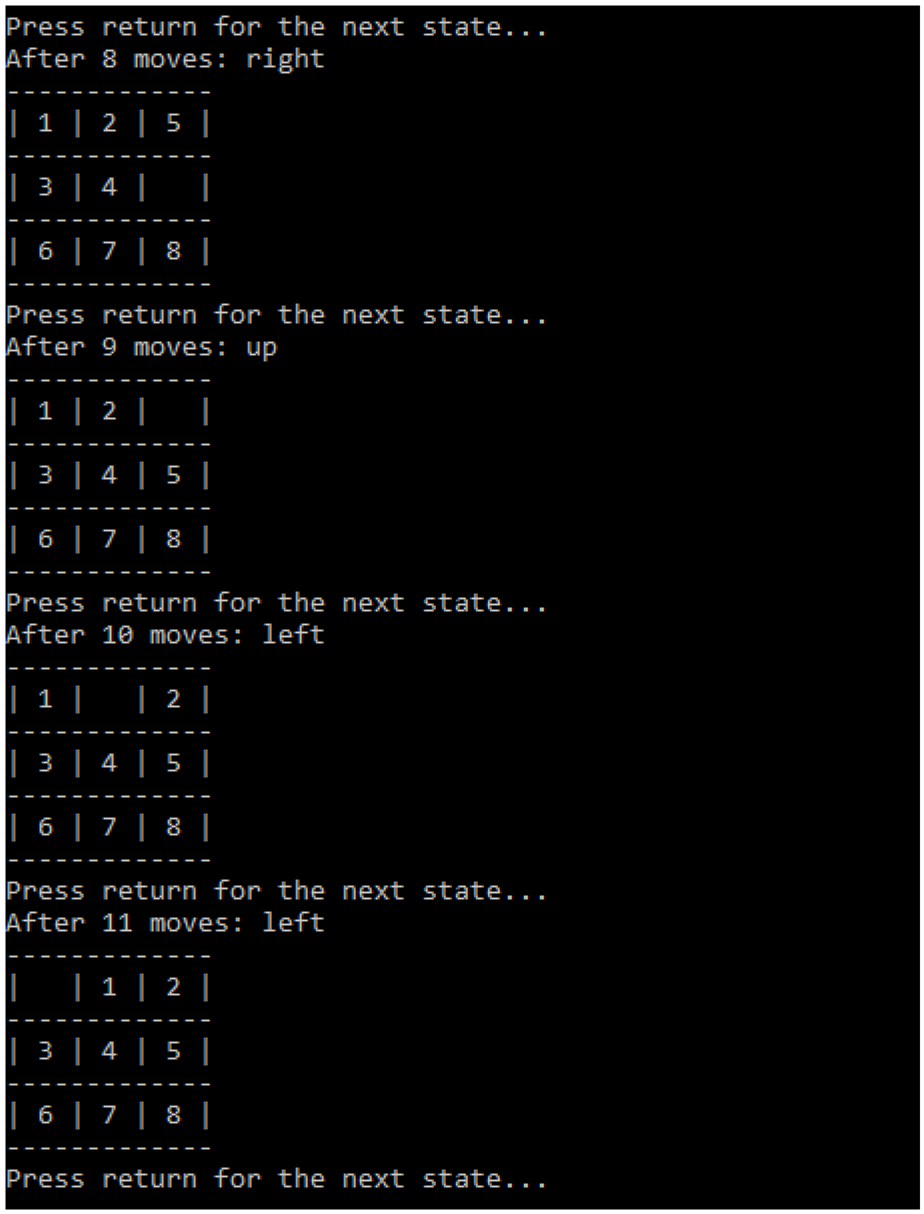
Questions 2 Autograder:



```
['West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'South', 'South', 'East', 'East', 'South', 'South', 'South', 'West', 'West'
, 'West', 'North', 'West', 'West', 'West', 'West', 'South', 'South', 'South', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'Sout
h', 'South', 'South', 'South', 'South', 'South', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'We
st', 'West', 'West', 'West', 'West', 'South', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West']
Path found with total cost of 68 in 0.3 seconds
Search nodes expanded: 267
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```



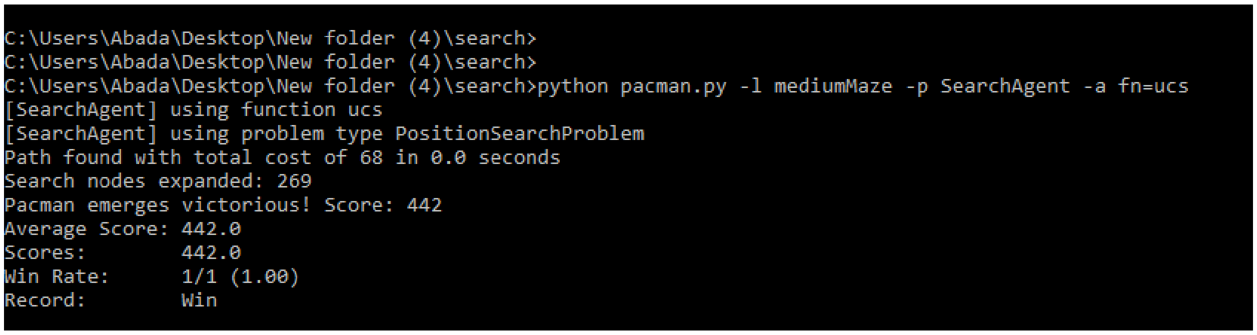
```
['North', 'North', 'West', 'West', 'West', 'West', 'North', 'North', 'West', 'West', 'South', 'South', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', '
West', 'West', 'West', 'West', 'West', 'West', 'North', 'North', 'East', 'East', 'North', 'North', 'West', 'West', 'North', 'North', 'North', 'North', 'North', 'North',
 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'East', 'East', 'North', 'North', 'East', 'East', 'East', 'East', 'North', 'North', 'East', 'East', '
South', 'South', 'East', 'East', 'North', 'North', 'North', 'North', 'North', 'North', 'East', 'East', 'East', 'East', 'North', 'North', 'North', 'North', 'North', 'Nor
th', 'North', 'North', 'North', 'North', 'West', 'West', 'South', 'South', 'West', 'West', 'West', 'West', 'South', 'South', 'South', 'South', 'South', 'South', 'West',
 'West', 'South', 'South', 'South', 'South', 'West', 'West', 'North', 'North', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', '
West', 'North', 'North', 'East', 'East', 'North', 'North', 'North', 'North', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East', 'North', 'North', 'North'
, 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'West', 'West', 'West', 'West', 'South', 'South', 'West', 'West', 'West', 'West', 'South', 'South', 'Sout
h', 'South', 'East', 'East', 'South', 'South', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'West', 'South', 'South', 'South', 'South', 'Sout
h', 'South', 'South', 'South', 'South', 'South', 'East', 'East', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'South', 'East', 'East',
 'South', 'South', 'West', 'West', 'South', 'South', 'South', 'South', 'West', 'West', 'South', 'South']
Path found with total cost of 210 in 1.1 seconds
Search nodes expanded: 617
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```



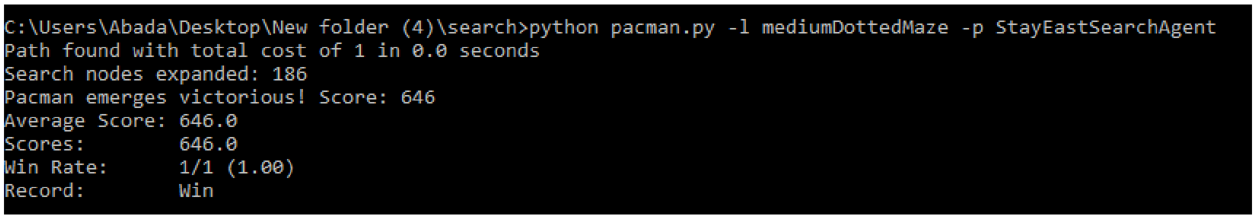
```
Press return for the next state...
After 8 moves: right
-------------
| 1 | 2 | 5 |
-------------
| 3 | 4 |   |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
After 9 moves: up
-------------
| 1 | 2 |   |
-------------
| 3 | 4 | 5 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
After 10 moves: left
-------------
| 1 |   | 2 |
-------------
| 3 | 4 | 5 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
After 11 moves: left
-------------
|   | 1 | 2 |
-------------
| 3 | 4 | 5 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
```

BFS succeeded to solve the eight-puzzle problem (a test step in this project) and reached the far left corner(the given goal) in the same number of nodes expanded as DFS.

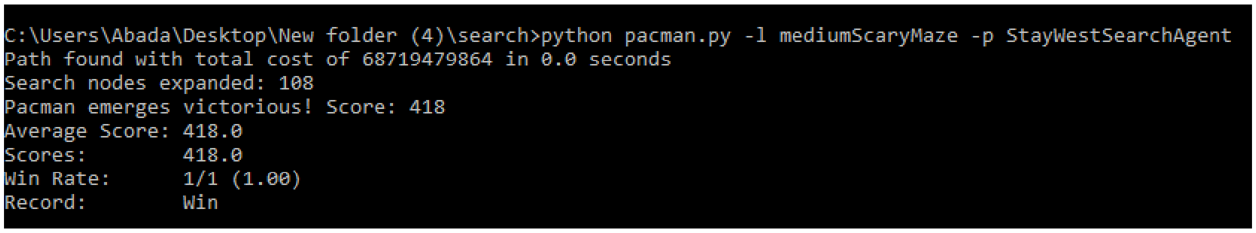
Questions 3 Autograder:



```
C:\Users\Abada\Desktop\New folder (4)\search>
C:\Users\Abada\Desktop\New folder (4)\search>
C:\Users\Abada\Desktop\New folder (4)\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 0.0 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
```



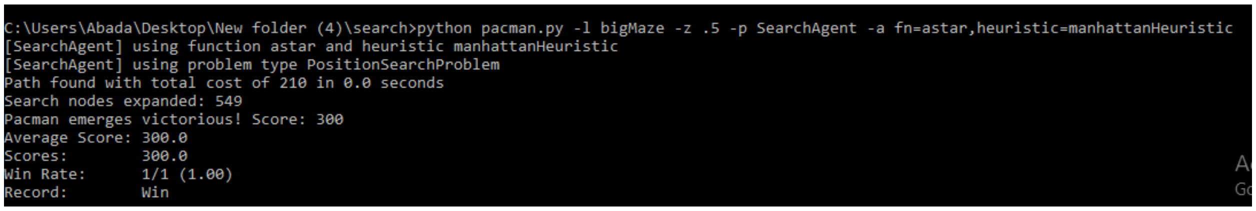
```
C:\Users\Abada\Desktop\New folder (4)\search>python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```



```
C:\Users\Abada\Desktop\New folder (4)\search>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
```

All dots are eaten using the UCF due to putting higher cost to areas with food and lower cost for areas with ghosts.

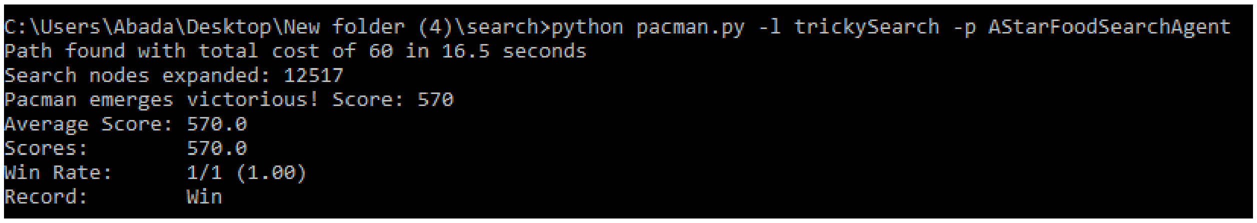
Questions 4 Autograder:



```
C:\Users\Abada\Desktop\New folder (4)\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 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
```

A* found the optimal solution in a shorter path than uniform cost search by using a heuristic function that gives a heuristic cost for the closest dots.

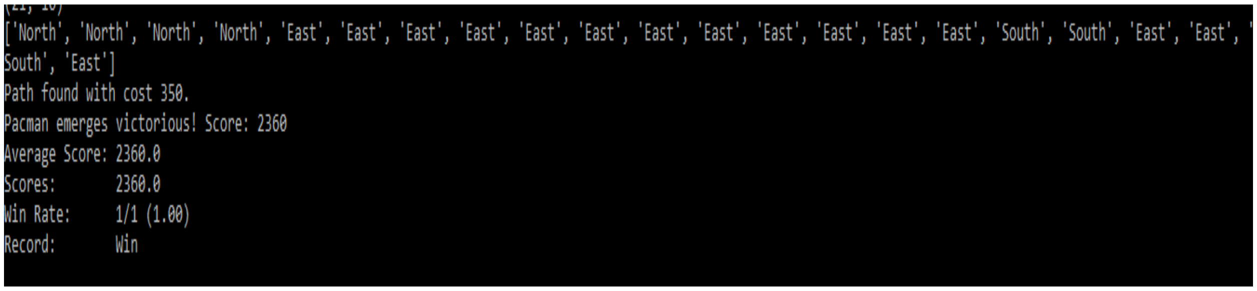
Questions 7 Autograder:



```
C:\Users\Abada\Desktop\New folder (4)\search>python pacman.py -l trickySearch -p AStarFoodSearchAgent
Path found with total cost of 60 in 16.5 seconds
Search nodes expanded: 12517
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```

A* Eated all dots, however, it expanded a very large number of nodes due to iterating over all distances between dots and the pacman agent.

Questions 8 Autograder:

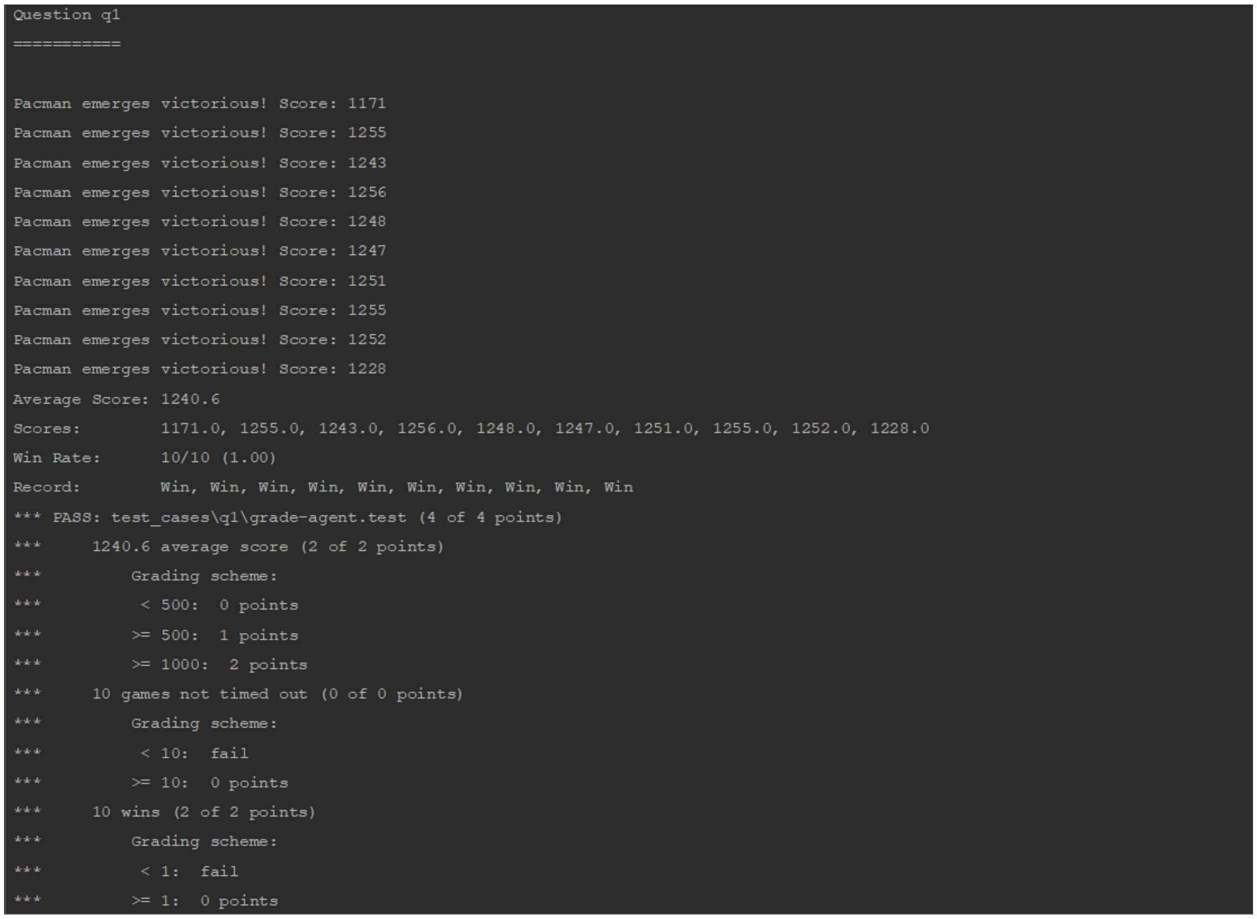


```
(21, 10)
['North', 'North', 'North', 'North', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'South', 'South', 'East', 'East', '
South', 'East']
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:        2360.0
Win Rate:      1/1 (1.00)
Record:        Win
```

This search has eaten all the Pacman food in fewer steps than Question 7 because it doesn't expand the distances between the pacman agent and all dots, instead, it finds the distance of the closest dot.

2. Project 2:

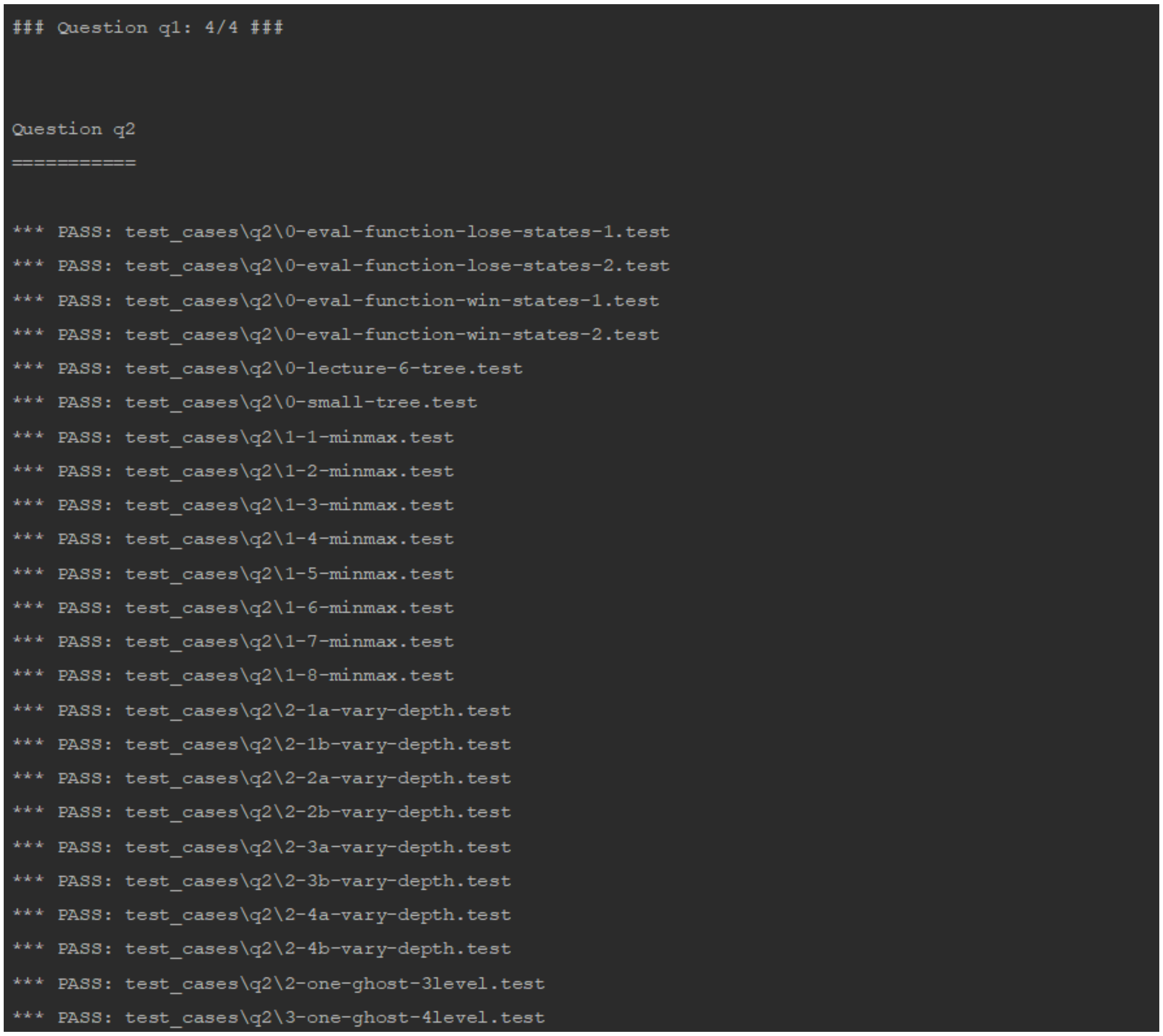
Comparing between Expctimax and the previous algorithms , a more cavalier approach in close quarters with ghosts is observed. In particular, if Pacman perceives that he could be trapped but might escape to grab a few more pieces of food, he'll at least try. Another observation , Expectimax agent wins about half the time, while the Alpha-beta pruning agent always loses. As Minimax and Alpha-beta pruning always assume the worst scenario it might die early to reduce the cost of living , in contrast the expectimax always tries. Figures 1,2,3,4,5 show the autograder results for all tasks in this project.



```
Question q1
===========

Pacman emerges victorious! Score: 1171
Pacman emerges victorious! Score: 1255
Pacman emerges victorious! Score: 1243
Pacman emerges victorious! Score: 1256
Pacman emerges victorious! Score: 1248
Pacman emerges victorious! Score: 1247
Pacman emerges victorious! Score: 1251
Pacman emerges victorious! Score: 1255
Pacman emerges victorious! Score: 1252
Pacman emerges victorious! Score: 1228
Average Score: 1240.6
Scores:        1171.0, 1255.0, 1243.0, 1256.0, 1248.0, 1247.0, 1251.0, 1255.0, 1252.0, 1228.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q1\grade-agent.test (4 of 4 points)
***     1240.6 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (0 of 0 points)
***         Grading scheme:
***          < 10:  fail
***         >= 10:  0 points
***     10 wins (2 of 2 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  0 points
```

Figure 1: autograder results for question 1 (reflex agent)

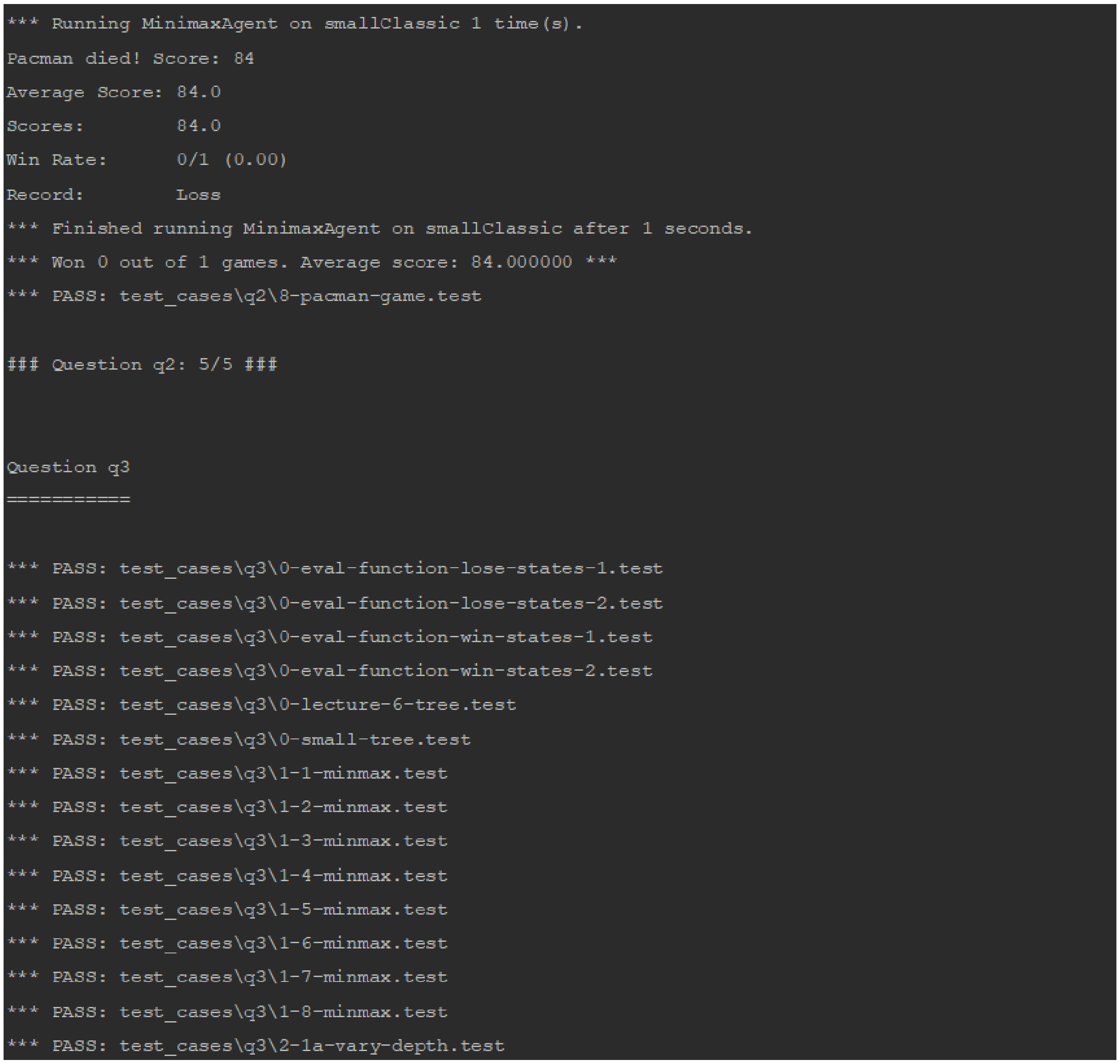


```
### Question q1: 4/4 ###


Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
```

Figure 2: autograder results for question 2(Minimax agent)



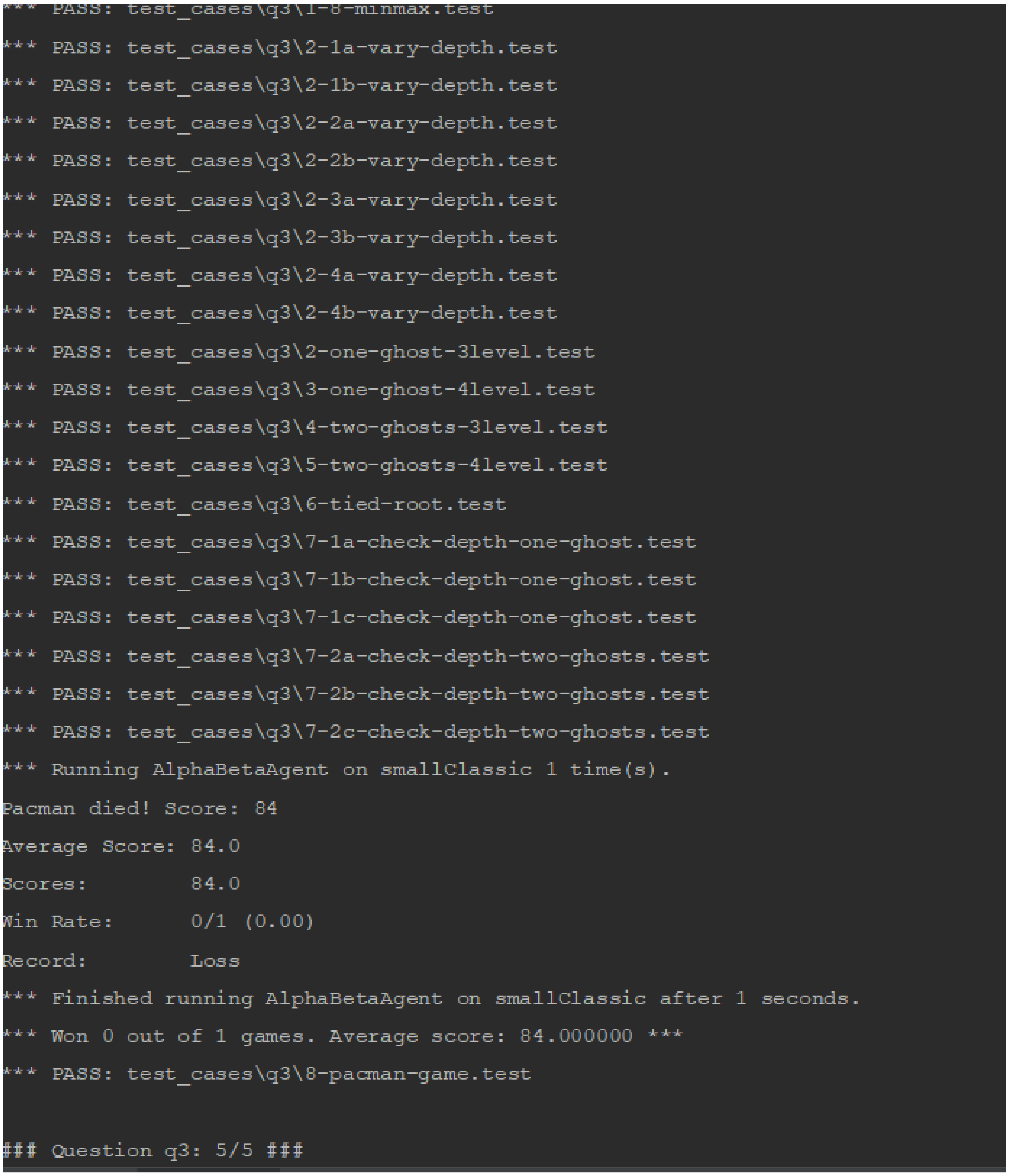
```
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###


Question q3
===========

*** PASS: test_cases\q3\0-eval-function-lose-states-1.test
*** PASS: test_cases\q3\0-eval-function-lose-states-2.test
*** PASS: test_cases\q3\0-eval-function-win-states-1.test
*** PASS: test_cases\q3\0-eval-function-win-states-2.test
*** PASS: test_cases\q3\0-lecture-6-tree.test
*** PASS: test_cases\q3\0-small-tree.test
*** PASS: test_cases\q3\1-1-minmax.test
*** PASS: test_cases\q3\1-2-minmax.test
*** PASS: test_cases\q3\1-3-minmax.test
*** PASS: test_cases\q3\1-4-minmax.test
*** PASS: test_cases\q3\1-5-minmax.test
*** PASS: test_cases\q3\1-6-minmax.test
*** PASS: test_cases\q3\1-7-minmax.test
*** PASS: test_cases\q3\1-8-minmax.test
*** PASS: test_cases\q3\2-1a-vary-depth.test
```

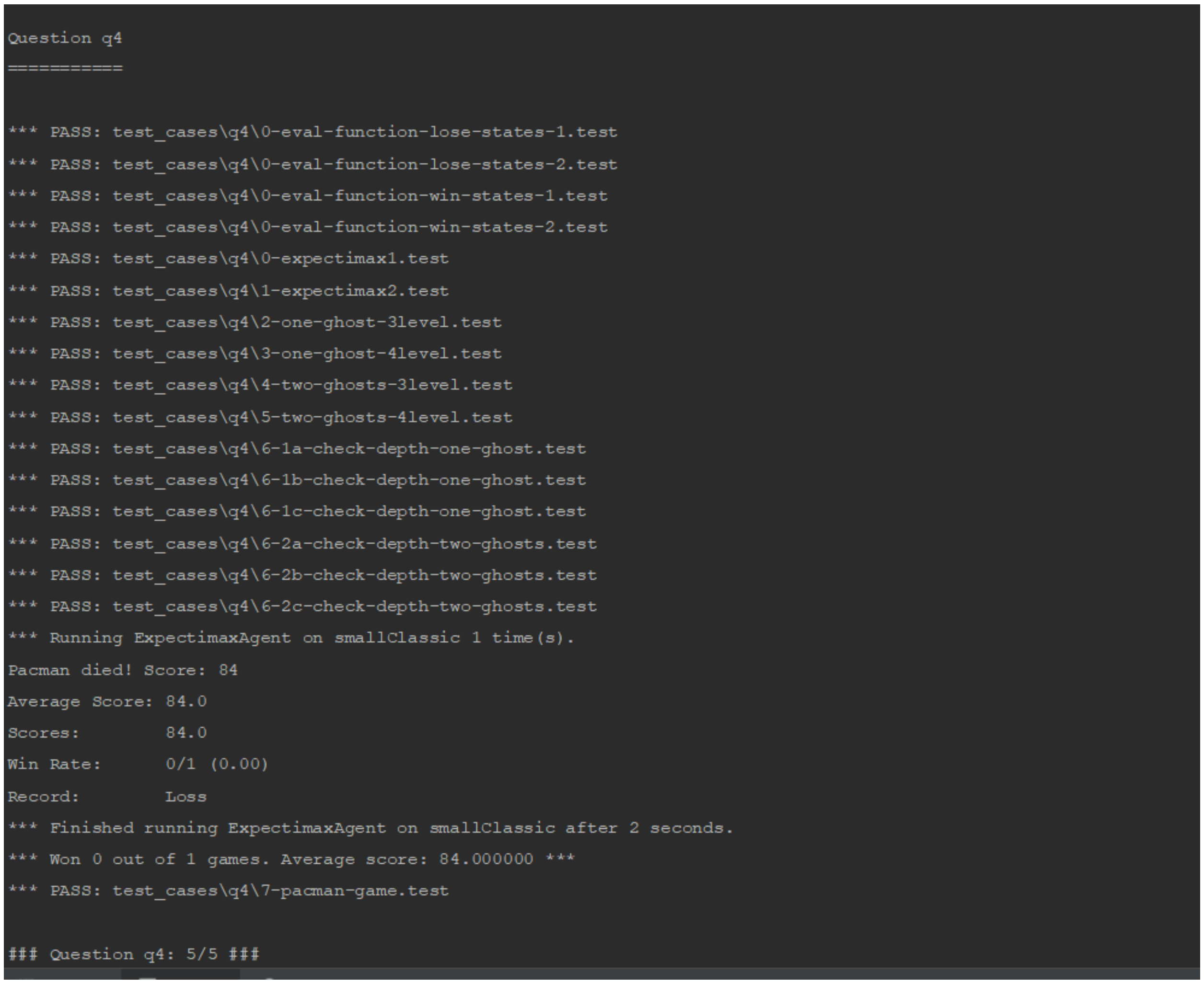
Figure 3 : autograder results for question 2 (Minimax agent)



```
*** PASS: test_cases\q3\1-8-minmax.test
*** PASS: test_cases\q3\2-1a-vary-depth.test
*** PASS: test_cases\q3\2-1b-vary-depth.test
*** PASS: test_cases\q3\2-2a-vary-depth.test
*** PASS: test_cases\q3\2-2b-vary-depth.test
*** PASS: test_cases\q3\2-3a-vary-depth.test
*** PASS: test_cases\q3\2-3b-vary-depth.test
*** PASS: test_cases\q3\2-4a-vary-depth.test
*** PASS: test_cases\q3\2-4b-vary-depth.test
*** PASS: test_cases\q3\2-one-ghost-3level.test
*** PASS: test_cases\q3\3-one-ghost-4level.test
*** PASS: test_cases\q3\4-two-ghosts-3level.test
*** PASS: test_cases\q3\5-two-ghosts-4level.test
*** PASS: test_cases\q3\6-tied-root.test
*** PASS: test_cases\q3\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q3\8-pacman-game.test

### Question q3: 5/5 ###
```

Figure 4: autograder results for question 3 (Alpha-beta pruning)

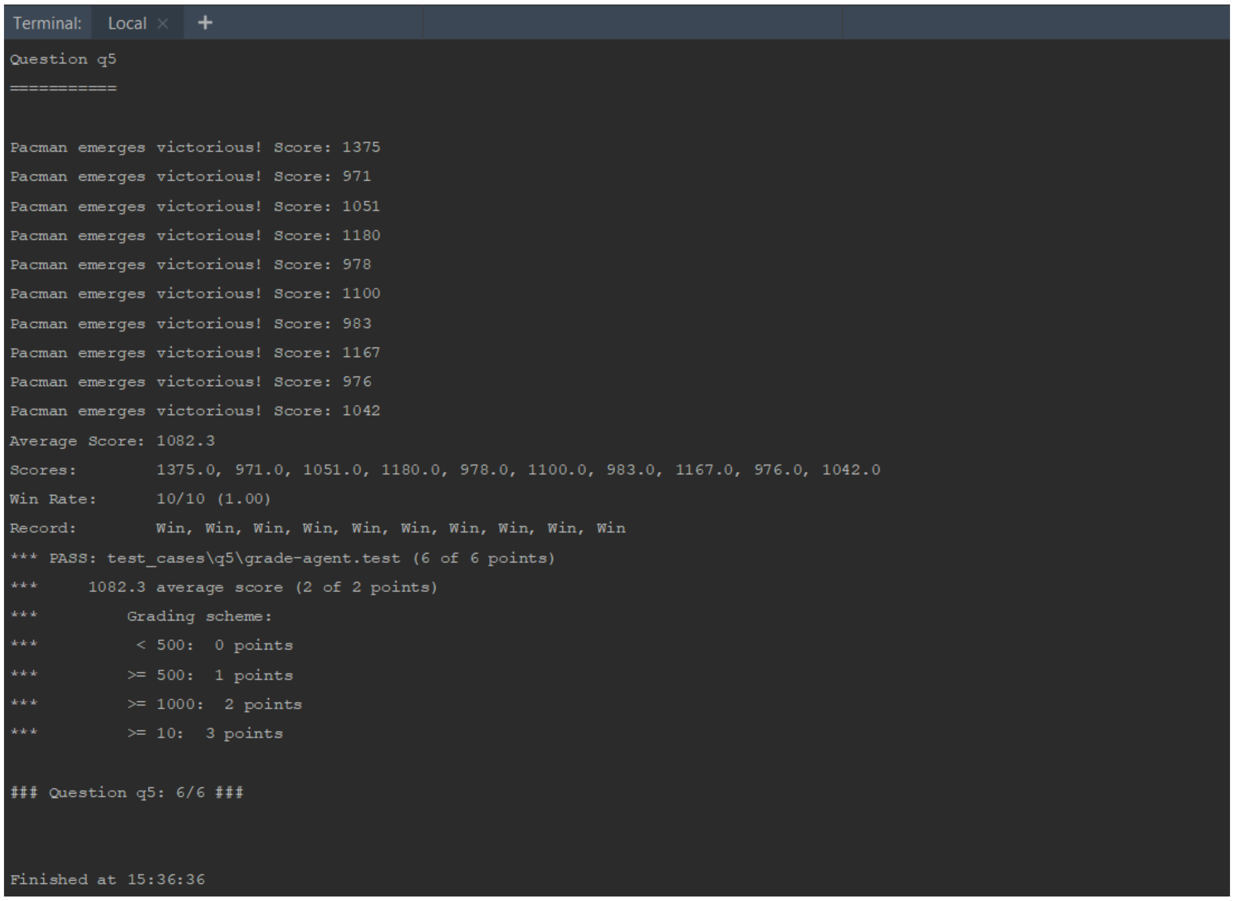

```
Question q4
==========

*** PASS: test_cases\q4\0-eval-function-lose-states-1.test
*** PASS: test_cases\q4\0-eval-function-lose-states-2.test
*** PASS: test_cases\q4\0-eval-function-win-states-1.test
*** PASS: test_cases\q4\0-eval-function-win-states-2.test
*** PASS: test_cases\q4\0-expectimax1.test
*** PASS: test_cases\q4\1-expectimax2.test
*** PASS: test_cases\q4\2-one-ghost-3level.test
*** PASS: test_cases\q4\3-one-ghost-4level.test
*** PASS: test_cases\q4\4-two-ghosts-3level.test
*** PASS: test_cases\q4\5-two-ghosts-4level.test
*** PASS: test_cases\q4\6-1a-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-1b-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-1c-check-depth-one-ghost.test
*** PASS: test_cases\q4\6-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q4\6-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q4\6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running ExpectimaxAgent on smallClassic after 2 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q4\7-pacman-game.test

### Question q4: 5/5 ###
```

Figure5: autograder results for question 4(Expectimax agent)

Redesigning a better evaluation function with more weighted heuristics, the Expectimax agent always wins and runs at a suitable rate as indicated by the autograder results in figure 6.



```
Question q5
==========

Pacman emerges victorious! Score: 1375
Pacman emerges victorious! Score: 971
Pacman emerges victorious! Score: 1051
Pacman emerges victorious! Score: 1180
Pacman emerges victorious! Score: 978
Pacman emerges victorious! Score: 1100
Pacman emerges victorious! Score: 983
Pacman emerges victorious! Score: 1167
Pacman emerges victorious! Score: 976
Pacman emerges victorious! Score: 1042
Average Score: 1082.3
Scores:        1375.0, 971.0, 1051.0, 1180.0, 978.0, 1100.0, 983.0, 1167.0, 976.0, 1042.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q5\grade-agent.test (6 of 6 points)
***     1082.3 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***         >= 10:  3 points

### Question q5: 6/6 ###


Finished at 15:36:36
```

Figure 6: autograder results for the Expectimax agent with the *better evaluation function* (question5)

As shown in the results, upgrading the search algorithms results in better performance in terms of win rate. A simple reflex agent plays respectably. It easily clears the *testClassic* layout(game board), however it dies with two ghosts on board with the conventional heuristic. Recognizing the probabilistic behavior of the ghost agents, the suitable agent to handle the problem in this project is the Expectimax agent as it relaxes the assumption of optimal behavior. Our results show that the Expectimax agent wins half the times with the conventional evaluation function. With the better evaluation function. The Expectimax agent wins all the times with an average score +1000 and runs at a suitable rate.

Contributions :

Mahmoud Gamal :

Project 1 :

1. DFS algorithm
2. BFS algorithms

Project 2 :

1. Minimax algorithm
2. Alpha-beta pruning algorithms
3. Expectimax algorithms

Eslam Zaher :

Project 1 :

1. Varying the Cost Function
2. Eating All The Dots

Project 2 :

1. Reflex agent
2. Evaluation function

Moustafa Abada :

Project 1 :

1. A* algorithm
2. Suboptimal Search

Project 2 :

1. Reflex agent