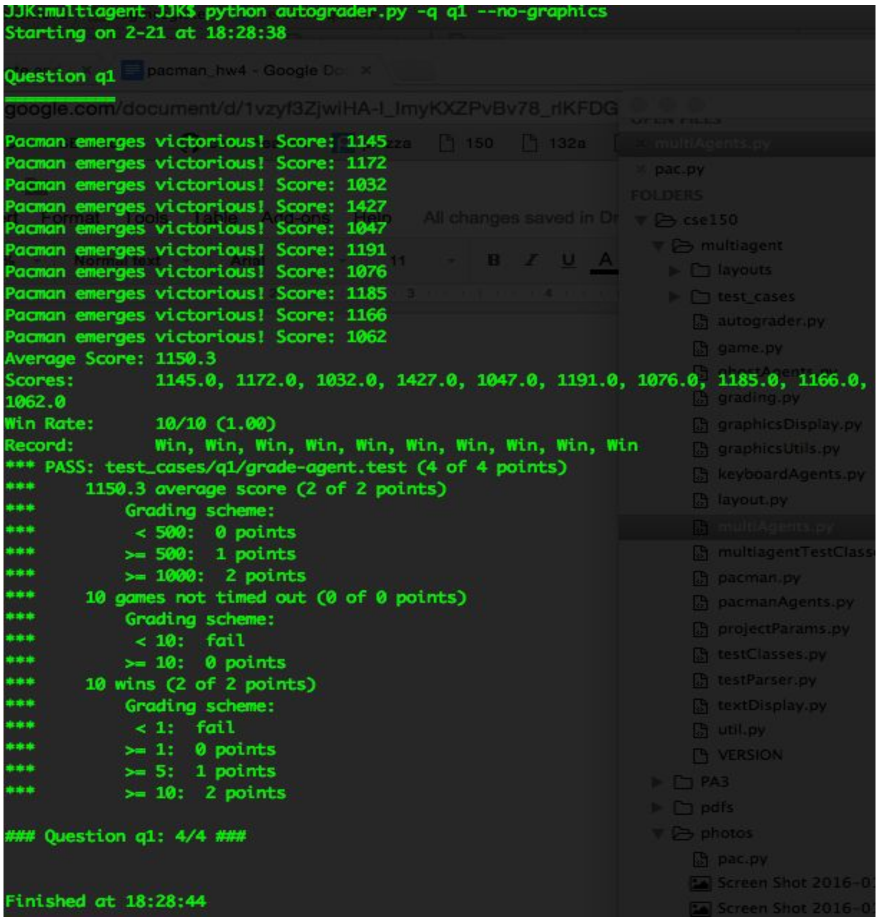
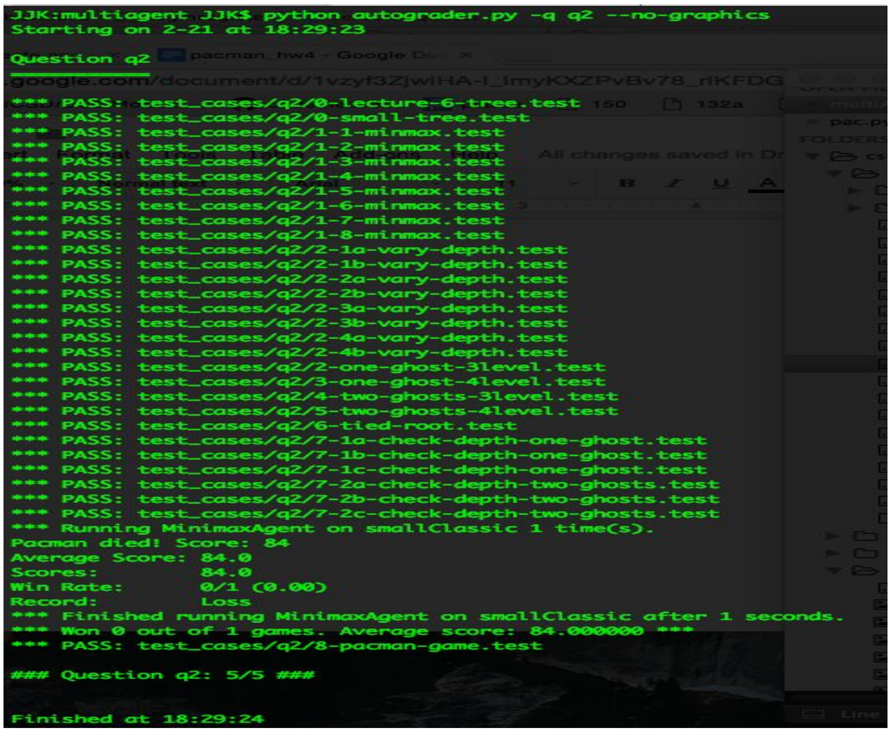
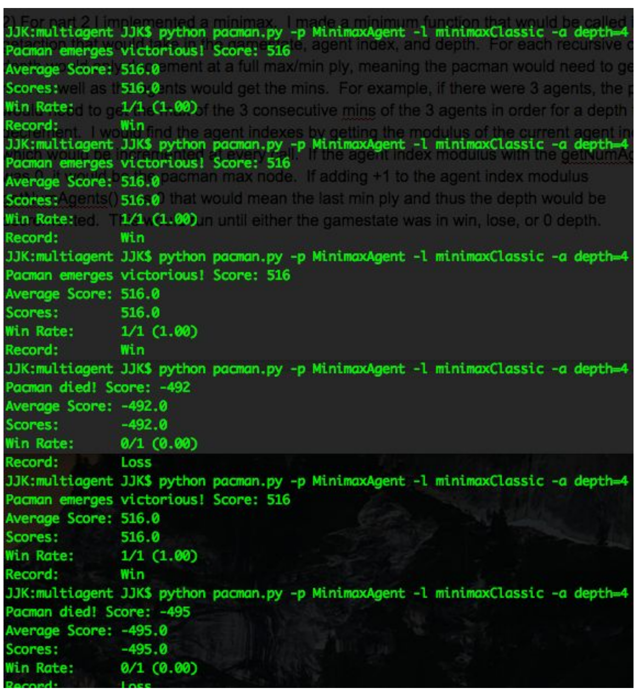
1) For part 1 I just used comparisons between the current state and the state that would come after a possible move by the pacman. I weighed the moves and found that applying weights to the distances of food and ghosts, with the ghost distance having a bit more weight since losing a game would be very bad. I also took all other factors into account, such as if moving to a place would result in eating a food, if the pacman not moving was more beneficial than moving, and if the new position would eat a capsule. It seemed to have positive results, all over 1000 points and a win. It would take 6 seconds overall. This is probably due to the time needed to check all the moves every time and make deterministic actions from each action.



2) For part 2 I implemented a minimax. I made a minimum function that would be called in getaction that would take in the gamestate, agent index, and depth. For each recursive call, the depth would only decrement at a full max/min ply, meaning the pacman would need to get the max as well as the agents would get the mins. For example, if there were 3 agents, the pacman would need to get the max of the 3 consecutive mins of the 3 agents in order for a depth to decrement. I would find the agent indexes by getting the modulus of the current agent index which would be incremented at every call. If the agent index modulus with the getNumAgents was 0, it would be the pacman max node. If adding +1 to the agent index modulus getNumAgents() was 0 that would mean the last min ply and thus the depth would be decremented. This would run until either the gamestate was in win, lose, or 0 depth. It would win ⅗ times when tested against the minimaxclassic test with a depth of 4.

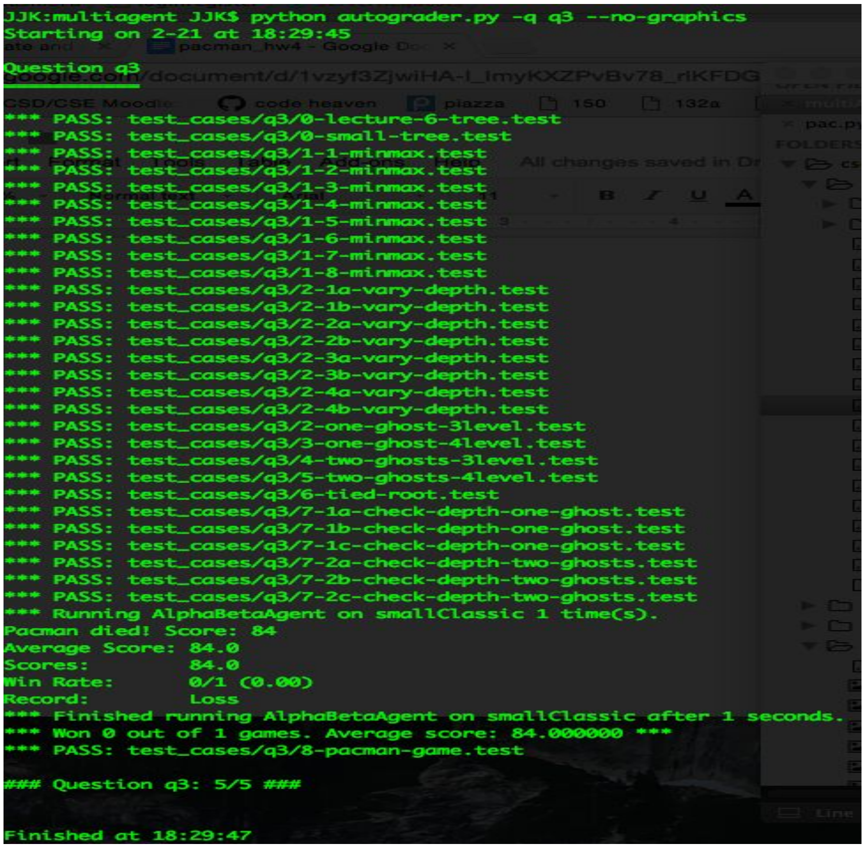


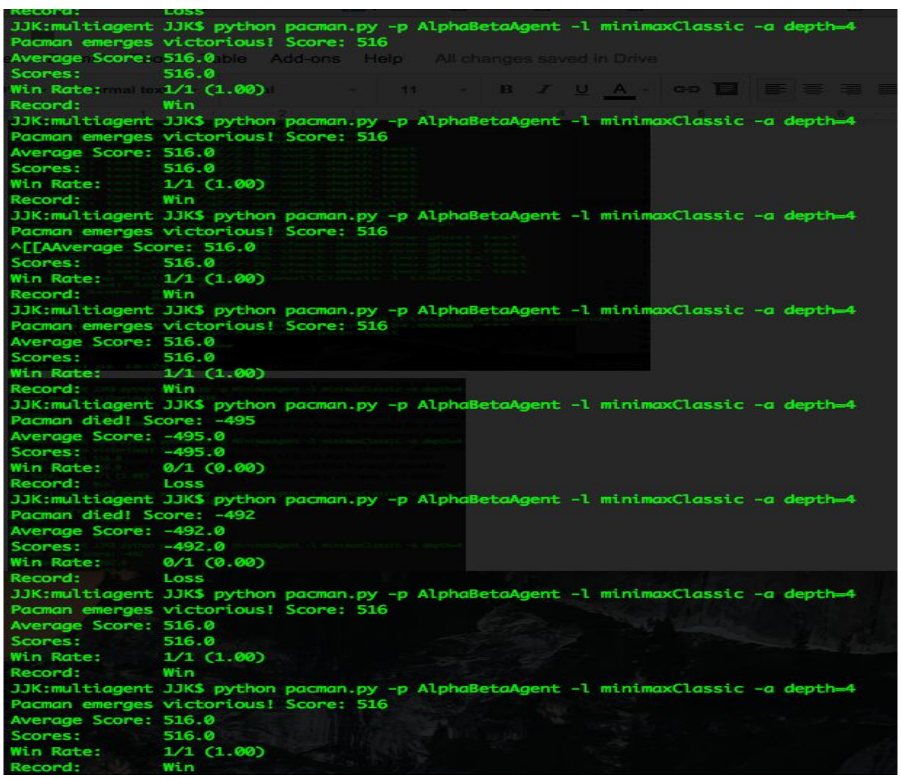
```
JJK:multiagent JJK$ python autograder.py -q q2 --no-graphics
Starting on 2-21 at 18:29:23

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

*** 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
*** PASS: test_cases/q2/4-two-ghosts-3level.test
*** PASS: test_cases/q2/5-two-ghosts-4level.test
*** PASS: test_cases/q2/6-tied-root.test
*** PASS: test_cases/q2/7-1a-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1b-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-1c-check-depth-one-ghost.test
*** PASS: test_cases/q2/7-2a-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2b-check-depth-two-ghosts.test
*** PASS: test_cases/q2/7-2c-check-depth-two-ghosts.test
*** 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 ###

Finished at 18:29:24
```



```
JJK:multiagent JJK$ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman died! Score: -492
Average Score: -492.0
Scores:        -492.0
Win Rate:      0/1 (0.00)
Record:        Loss
JJK:multiagent JJK$ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4
Pacman died! Score: -495
Average Score: -495.0
Scores:        -495.0
Win Rate:      0/1 (0.00)
Record:        Loss
```

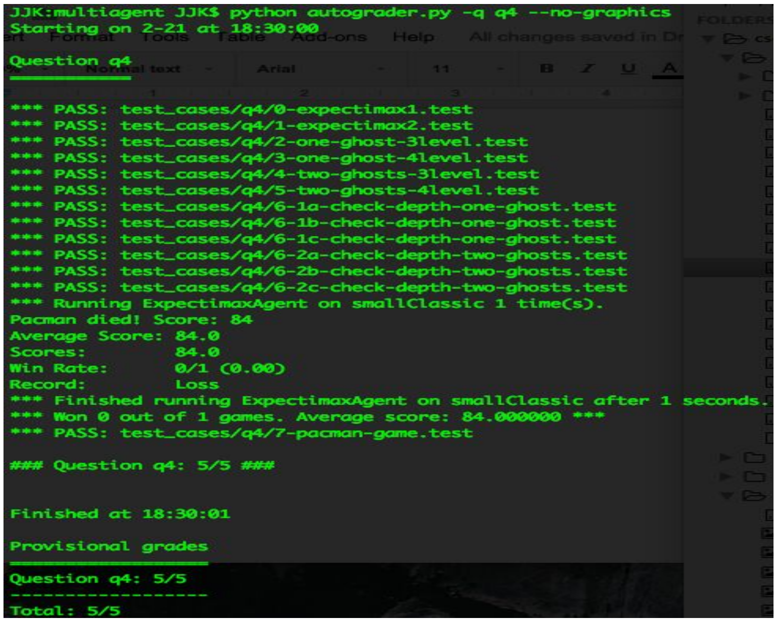
3) I did the AB pruning the exact same way as before, but I added the AB prunning check which would pass on alpha and beta values and prune possible branch moves based off of min and max values from leaf nodes. Rather than check every value, AB would reduce the number of possibilities that would need to be searched in a minimax search. It had a winning rate of 6/8, which was similar to the minimax as expected since it does the same as minimax, but just reduces search domains using the AB values.

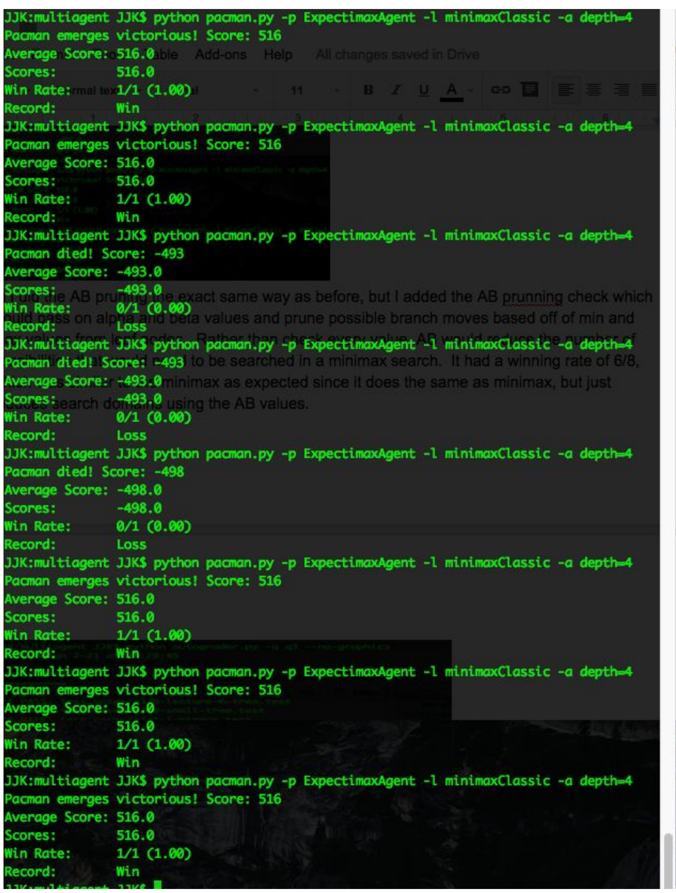

```
JJK:multiagent JJK$ python autograder.py -q q3 --no-graphics
Starting on 2-21 at 18:29:45

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

*** 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
*** 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 ###


Finished at 18:29:47
```


```
Record:        Loss
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
^[[AAverage Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman died! Score: -495
Average Score: -495.0
Scores:        -495.0
Win Rate:      0/1 (0.00)
Record:        Loss
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman died! Score: -492
Average Score: -492.0
Scores:        -492.0
Win Rate:      0/1 (0.00)
Record:        Loss
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
JJK:multiagent JJK$ python pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4
Pacman emerges victorious! Score: 516
Average Score: 516.0
Scores:        516.0
Win Rate:      1/1 (1.00)
Record:        Win
```

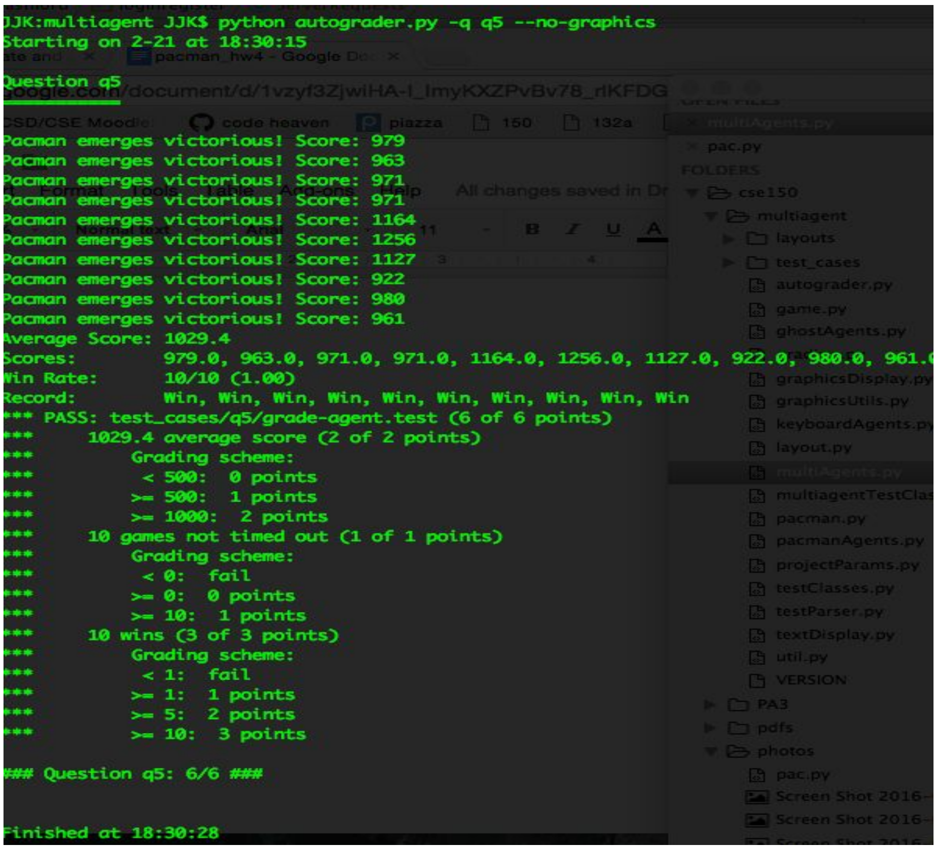
4) The implementation of the expeximax was similar to the minimax as well, but rather than get the mins, the values that were passed along would be averaged instead. This would allow for a

more optimistic performance rather than the worst case scenario of minimax. It had a winning rate of ⅝. Although it did not perform as well as the other searches, it would do better in scenarios that would have situations that were not very optimistic. The power of the expectimax comes from the averaging of max min values and rather than concede in a non optimistic game state, it would still attempt moves that were the best and win in some cases.





5) For part 5, I took a similar approach as part 1 with weighted values determining the current game state. I weighted ghost distances a bit heavier than food so that the pacman would try its best to avoid death states. The distance of the closest ghost would also affect other factors directly, by weighing food locations as less preferable if the ghost was extremely close to the food location as well. I also took into account the number of food that was on the board as well as the number of capsule locations on the board. The results were not as great as the other evaluation function, but would score extremely well on boards with strong starts, like where the agents would be on the other side of the board initially.



```
JJK:multiagent JJK$ python autograder.py -q q5 --no-graphics
Starting on 2-21 at 18:30:15

Question q5
===========

Pacman emerges victorious! Score: 979
Pacman emerges victorious! Score: 963
Pacman emerges victorious! Score: 971
Pacman emerges victorious! Score: 971
Pacman emerges victorious! Score: 1164
Pacman emerges victorious! Score: 1256
Pacman emerges victorious! Score: 1127
Pacman emerges victorious! Score: 922
Pacman emerges victorious! Score: 980
Pacman emerges victorious! Score: 961
Average Score: 1029.4
Scores:        979.0, 963.0, 971.0, 971.0, 1164.0, 1256.0, 1127.0, 922.0, 980.0, 961.0
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)
***     1029.4 average score (2 of 2 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  1 points
***         >= 1000:  2 points
***     10 games not timed out (1 of 1 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 10:  1 points
***     10 wins (3 of 3 points)
***         Grading scheme:
***          < 1:  fail
***         >= 1:  1 points
***         >= 5:  2 points
***         >= 10:  3 points

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

Finished at 18:30:28
```