**Date – 5th October, 2018**

Project 2: The Multi-Agent Pac-Man

# Sunny Shah - 112044068

## Question 1: Reflex Agent

***>python pacman.py -p ReflexAgent -l testClassic***

Pacman emerges victorious! Score: 563

Average Score: 563.0

Scores: 563.0

Win Rate: 1/1 (1.00)

Record: Win

***>python pacman.py --frameTime 0 -p ReflexAgent -k 1***

Pacman emerges victorious! Score: 1181

Average Score: 1181.0

Scores: 1181.0

Win Rate: 1/1 (1.00)

Record: Win

***>python pacman.py --frameTime 0 -p ReflexAgent -k 2***

Pacman emerges victorious! Score: 1525

Average Score: 1525.0

Scores: 1525.0

Win Rate: 1/1 (1.00)

Record: Win

# Logic used by my evaluation function:

My evaluation function grades the game state in 2 parts:

Part 1:

Find the minimum distance to the closest food point. I have used the manhattan distance to calculate the minimum distance.

Part 2:

Even if we can eat the food, we need to check if we are safe from ghost. Following are the conditions on which we measure the score:

1. If the manhattanDistance to ghost is 1, and ghost is not scared, then we should penalize heavily
2. If the manhattanDistance to ghost is 2, and ghost is not scared, then we should penalize heavily
3. (Manhattan distance + scaredTimer for the ghost) is zero, we penalize heavily, since step 4 below will fail
4. For all the other options, we can see that the chance of getting killed by ghost is inversely proportional to (scareTimer of the ghost and manhattanDistance of the ghost)

Now, our score metrics above is mixture of minDistance to closest food and relative chance of getting killed from ghost. Thus, score will be low for a good action. But we need to output a big value for good actions. We do this by negating the score and then returning the negated value.

# Autograder results:

***>python autograder.py -q q1***

Starting on 10-5 at 23:27:26

Question q1

===========

Pacman emerges victorious! Score: 1230

Pacman emerges victorious! Score: 1236

Pacman emerges victorious! Score: 1233

Pacman emerges victorious! Score: 1235

Pacman emerges victorious! Score: 1246

Pacman emerges victorious! Score: 1230

Pacman emerges victorious! Score: 1253

Pacman emerges victorious! Score: 1253

Pacman emerges victorious! Score: 1238

Pacman emerges victorious! Score: 1231

Average Score: 1238.5

Scores: 1230.0, 1236.0, 1233.0, 1235.0, 1246.0, 1230.0, 1253.0, 1253.0, 1238.0, 1231.0

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)

\*\*\* 1238.5 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

\*\*\* >= 5: 1 points

\*\*\* >= 10: 2 points

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

Finished at 23:27:30

Provisional grades

==================

Question q1: 4/4

------------------

Total: 4/4

Your grades are NOT yet registered. To register your grades, make sure

to follow your instructor's guidelines to receive credit on your project.

## Question 2: Minimax

***>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

***>python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3***

Pacman died! Score: -501

Average Score: -501.0

Scores: -501.0

Win Rate: 0/1 (0.00)

Record: Loss

# Logic used for Minimax agent:

* The minimax agent can be modelled using 2 functions:
  + play\_max
    - This method tries to find the max score a MAX player can make.  
      It is based on the following logic

1. If the currDepth is the maximum depth according to the self.depth, then directly calculate the score for the state using the self.evaluationFunction and the return the score.
2. Calculate the max score based on the scores of the MIN players for every action taken by the MAX player.
3. If we cannot find any optimal max score (this may be probably because of state either being a win or a lose position), we directly send the score of that state using the state.getScore() method
4. In case, we found the max score, than we update the self.nextActionToTake property, which will be used by our getAction method.
   * play\_min
     + This method tries to find the min score a MIN player (ghost agent) can make.
5. It is based on the following logic:
6. Calculate the min score recursively for each action. A action taken by agent should consult the next agent for its score and similarly till we traverse all the MIN agents.
7. If we cannot find any optimal min score (this may be probably because of state either being a win or a lose position) we directly send the score of that state using the state.getScore() method
8. In case, we found the min score, we return the value. This will be the minimum score the ghosts(adversaries) will try to make to compete against pacman.

# Autograder results:

***>python autograder.py -q q2***

Starting on 10-6 at 0:32:03

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 0 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 0:32:04

Provisional grades

==================

Question q2: 5/5

------------------

Total: 5/5

Your grades are NOT yet registered. To register your grades, make sure

to follow your instructor's guidelines to receive credit on your project.

Minimax Analysis:

1. Reflex agent performs better than minimax agent, due to large state space calculations involved in the minimax calculations.

## Question 3: Alpha-Beta Pruning

***>pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic***

Pacman emerges victorious! Score: 1708

Average Score: 1708.0

Scores: 1708.0

Win Rate: 1/1 (1.00)

Record: Win

# Logic used for Alpha-Beta Pruning:

* The alpha-beta pruning agent can be modelled using 2 functions:
  + play\_max
    - This method tries to find the max score a MAX player can make.  
      It is based on the following logic

1. If the currDepth is the maximum depth according to the self.depth, then directly calculate the score for the state using the self.evaluationFunction and the return the score.
2. Calculate the max score based on the scores of the MIN players for every action taken by the MAX player.
   1. Check if the max score so far has crossed the beta value. If yes, then return, since the MIN player will not be using this value.
3. If we cannot find any optimal max score (this may be probably because of state either being a win or a lose position), we directly send the score of that state using the state.getScore() method
4. In case, we found the max score, than we update the self.nextActionToTake property, which will be used by our getAction method.
   * play\_min
     + This method tries to find the min score a MIN player (ghost agent) can make.
5. It is based on the following logic:
6. Calculate the min score recursively for each action. A action taken by agent should consult the next agent for its score and similarly till we traverse all the MIN agents.
   1. Check if the min score so far has gone below the alpha value. If yes, then return, since the MAX player will not be using this value.
7. If we cannot find any optimal min score (this may be probably because of state either being a win or a lose position) we directly send the score of that state using the state.getScore() method

In case, we found the min score, we return the value. This will be the minimum score the ghosts(adversaries) will try to make to compete against pacman.

# Autograder results:

***>python autograder.py -q q3***

Starting on 10-6 at 1:50:13

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 0 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 1:50:14

Provisional grades

==================

Question q3: 5/5

------------------

Total: 5/5

Your grades are NOT yet registered. To register your grades, make sure

to follow your instructor's guidelines to receive credit on your project.

Alpha-Beta Pruning Analysis:

## Question 4: Expectimax

***>pacman.py -p ExpectimaxAgent -l minimaxClassic -a depth=3***

Pacman emerges victorious! Score: 515

Average Score: 515.0

Scores: 515.0

Win Rate: 1/1 (1.00)

Record: Win

***> python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10***

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Pacman died! Score: -501

Average Score: -501.0

Scores: -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0

Win Rate: 0/10 (0.00)

Record: Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss

***>python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10***

Pacman emerges victorious! Score: 532

Pacman died! Score: -502

Pacman died! Score: -502

Pacman died! Score: -502

Pacman emerges victorious! Score: 532

Pacman died! Score: -502

Pacman emerges victorious! Score: 532

Pacman emerges victorious! Score: 532

Pacman died! Score: -502

Pacman emerges victorious! Score: 532

Average Score: 15.0

Scores: 532.0, -502.0, -502.0, -502.0, 532.0, -502.0, 532.0, 532.0, -502.0, 532.0

Win Rate: 5/10 (0.50)

Record: Win, Loss, Loss, Loss, Win, Loss, Win, Win, Loss, Win

# Logic used for Expectimax Pruning:

* The alpha-beta pruning agent can be modelled using 2 functions:
  + play\_max
    - This method tries to find the max score a MAX player can make.  
      It is based on the following logic

1. If the currDepth is the maximum depth according to the self.depth, then directly calculate the score for the state using the self.evaluationFunction and the return the score.
2. Calculate the max score based on the scores of the MIN players for every action taken by the MAX player.
3. If we cannot find any optimal max score (this may be probably because of state either being a win or a lose position), we directly send the score of that state using the state.getScore() method
4. In case, we found the max score, than we update the self.nextActionToTake property, which will be used by our getAction method.
   * play\_min
     + This method tries to find the min score a MIN player (ghost agent) can make.
5. It is based on the following logic:
6. Calculate the min score recursively for each action. A action taken by agent should consult the next agent for its score and similarly till we traverse all the MIN agents.
7. Store the score from each action in a list.
8. If we cannot find any optimal min score (this may be probably because of state either being a win or a lose position), we directly send the score of that state using the state.getScore() method
9. Now since the ghosts play randomly, we can find the expected score as the sum of scores divided by the number of actions

# Autograder results:

***>python autograder.py -q q4***

1. Starting on 10-6 at 2:00:04
2. Question q4
3. ===========
4. \*\*\* PASS: test\_cases\q4\0-expectimax1.test
5. \*\*\* PASS: test\_cases\q4\1-expectimax2.test
6. \*\*\* PASS: test\_cases\q4\2-one-ghost-3level.test
7. \*\*\* PASS: test\_cases\q4\3-one-ghost-4level.test
8. \*\*\* PASS: test\_cases\q4\4-two-ghosts-3level.test
9. \*\*\* PASS: test\_cases\q4\5-two-ghosts-4level.test
10. \*\*\* PASS: test\_cases\q4\6-1a-check-depth-one-ghost.test
11. \*\*\* PASS: test\_cases\q4\6-1b-check-depth-one-ghost.test
12. \*\*\* PASS: test\_cases\q4\6-1c-check-depth-one-ghost.test
13. \*\*\* PASS: test\_cases\q4\6-2a-check-depth-two-ghosts.test
14. \*\*\* PASS: test\_cases\q4\6-2b-check-depth-two-ghosts.test
15. \*\*\* PASS: test\_cases\q4\6-2c-check-depth-two-ghosts.test
16. \*\*\* Running ExpectimaxAgent on smallClassic 1 time(s).
17. Pacman died! Score: 84
18. Average Score: 84.0
19. Scores: 84.0
20. Win Rate: 0/1 (0.00)
21. Record: Loss
22. \*\*\* Finished running ExpectimaxAgent on smallClassic after 0 seconds.
23. \*\*\* Won 0 out of 1 games. Average score: 84.000000 \*\*\*
24. \*\*\* PASS: test\_cases\q4\7-pacman-game.test
25. ### Question q4: 5/5 ###
26. Finished at 2:00:05
27. Provisional grades
28. ==================
29. Question q4: 5/5
30. ------------------
31. Total: 5/5
32. Your grades are NOT yet registered. To register your grades, make sure
33. to follow your instructor's guidelines to receive credit on your project.

Expectimax Analysis: