Date – 7th 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

```
totalNodesExpandedTillNow =  5916
totalNodesExpandedTillNow =  5972
totalNodesExpandedTillNow =  6110
totalNodesExpandedTillNow =  6813
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

```
totalNodesExpandedTillNow =  77
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 3 functions:
    - Value
        - The main purpose of this method is to decide which function to call i.e. whether it is MAX's chance or MIN's chance. There can be 2 possible actions:
            1. If we have went through all the ghosts, then we will receive agent value as numberOfAgent. This means that, we have finished expanding the plies for MIN and we should now call max_value. In doing so, we should also increase the depth, as completing the MIN for all ghosts, means one total move.
            2. Agent is between 1 and numberOfAgents-1, indicating we are still in the same move, where we are evaluating the actions for each ghost.
    - max_value
        - 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.

- min_value
  - This method tries to find the min score a MIN player (ghost agent) can make. It is based on the following logic:
    1. 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.
    2. 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
    3. 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:

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

Question 3: Alpha-Beta Pruning

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

```
..
..
..
..
totalNodesExpandedTillNow =  142006
totalNodesExpandedTillNow =  142230
totalNodesExpandedTillNow =  142433
Pacman emerges victorious! Score: 1231
Average Score: 1231.0
Scores:        1231.0
Win Rate:      1/1 (1.00)
Record:        Win
```

NOTE: Below is the same layout with a minimax agent. We see that the number of nodes expanded is more than that of alpha-beta agent

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

```
..
..
..
..
totalNodesExpandedTillNow =  231522
totalNodesExpandedTillNow =  231665
Pacman emerges victorious! Score: 1233
Average Score: 1233.0
Scores:        1233.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Logic used for Alpha-Beta Pruning:

- The alpha-beta pruning agent can be modelled using 3 functions:
    - Value
        - The main purpose of this method is to decide which function to call i.e. whether it is MAX's chance or MIN's chance. There can be 2 possible actions:
            1. If we have went through all the ghosts, then we will receive agent value as numberOfAgent. This means that, we have finished expanding the plies for MIN and we should now call max_value. In doing so, we should also increase the depth, as completing the MIN for all ghosts, means one total move.
            2. Agent is between 1 and numberOfAgents-1, indicating we are still in the same move, where we are evaluating the actions for each ghost.
    - max_value

- 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.
     a. 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.

- min_value
  - This method tries to find the min score a MIN player (ghost agent) can make.
    It is based on the following logic:
    1. 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.
       a. 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.
    2. 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:

- Alpha -Beta pruning run on the same layout seems to perform faster than minimax algorithm.
- Examining the complete state space tree forces the minimax algorithm to consider the action of STOP if the ghosts are far away.
- Running the same layout on minimax and alpha-beta agent shows that minimax being idle in most of the moves, which this happened comparatively less with alpha-beta agent.

Question 4: Expectimax

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

```
totalNodesExpandedTillNow =  1160
totalNodesExpandedTillNow =  1855
totalNodesExpandedTillNow =  3143
totalNodesExpandedTillNow =  3431
totalNodesExpandedTillNow =  3647
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 3 functions:
    - Value
        - The main purpose of this method is to decide which function to call i.e. whether it is MAX's chance or MIN's chance. There can be 2 possible actions:
            1. If we have went through all the ghosts, then we will receive agent value as numberOfAgents. This means that, we have finished expanding the plies for CHANCE nodes and we should now call max_value. In doing so, we should also increase the depth, as completing the CHANCE for all ghosts, means one total move.
            2. 2Agent is between 1 and numberOfAgents-1, indicating we are still in the same move, where we are evaluating the actions for each ghost, which will be a chance_value for ghost.
    - max_value
        - This method tries to find the max score a MAX player can make.
          It is based on the following logic
            3. 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.
            4. Calculate the max score based on the scores of the MIN players for every action taken by the MAX player.
            5. 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
            6. In case, we found the max score, than we update the self.nextActionToTake property, which will be used by our getAction method.

    - min_value
        - This method tries to find the min score a MIN player (ghost agent) can make.
          It is based on the following logic:
            1. 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.
            2. Store the score from each action in a list.
            3. 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
            4. 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

```
Starting on 10-6 at 2:00:04

Question q4
==========

*** 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 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q4\7-pacman-game.test

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


Finished at 2:00:05

Provisional grades
==================
Question q4: 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.
```

Expectimax Analysis:

- We can't apply alpha-beta pruning technique to expectimax agent as we cannot guarantee the expected outcome of any subtree.
- Thus, expectimax has to evaluate till the depth of the tree, unless we find some linear evaluation function.