AI HW2

R12922029 陳姵安

1. **Show your autograder results and describe each algorithm**
    a. **autograder results**
        i. **Q1. Reflex Agent (2%)**

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

Pacman emerges victorious! Score: 1238
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1235
Pacman emerges victorious! Score: 1233
Pacman emerges victorious! Score: 1241
Pacman emerges victorious! Score: 1246
Pacman emerges victorious! Score: 1242
Pacman emerges victorious! Score: 1239
Pacman emerges victorious! Score: 1242
Average Score: 1239.9
Scores:        1238.0, 1244.0, 1239.0, 1235.0, 1233.0, 1241.0, 1246.0, 1242.0, 1239.0, 1242.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases/q1/grade-agent.test (30.0 of 30.0 points)
***     1239.9 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: 30/30 ###
```

        ii. **Q2. Minimax (2%)**

```
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
*** 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: 30/30 ###
```

iii. Q3. Alpha-Beta Pruning (2%)

```
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
*** 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: 30/30 ###
```

b. describe each algorithm

i. Q1. Reflex Agent

The Reflex Agent selects the best action from the available choices based on the current environment it perceives for its next move. **The criterion** for determining the best action is purely from Pacman's perspective, aiming to avoid contact with Ghosts to prevent the game from ending, and striving to consume as much food as possible.

The implementation is divided into two parts: ***getAction*** and ***evaluationFunction***. ***getAction*** gathers the current GameState's Successor State and the corresponding legal actions as candidates to input to ***evaluationFunction***. The ***evaluationFunction*** then chooses the relatively best few actions based on the maze environment. Afterwards, ***getAction*** returns by randomly selecting one of the best actions for the next step.

ii. Q2. Minimax

The Minimax Agent also selects the best action from the available choices for the next step, but **the criteria** for determining the best action are a bit more complex than that of the Reflex Agent. In addition to choosing the action that maximizes Pacman's score, it also considers the next move of the Ghosts, theoretically "the move that could result in the maximum possible loss of points for Pacman."

Therefore, it employs a "minimize the maximum loss" strategy to make the Ghosts choose the action that results in the lowest score.

The implementation includes three functions: ***minmax(), maxPoint(), and minPoint()***.

minmax() traverses all possible states of the tree (across all depths) to determine if the current state is an end state. If the game has not ended, it then proceeds based on the ***agentIndex*** to determine the agent's role, and uses different action evaluation methods depending on the role. For Pacman (where ***agentIndex*** == 0), it calculates the highest scoring action for the next step using ***maxPoint()***. For ghosts (where ***agentIndex*** > 0), it calculates the lowest scoring action for the next step using ***minPoint()***.

maxPoint() selects the action with the highest score. Therefore, ***maxPoint*** is initially set to negative infinity. It then iterates over all action candidates of the current agent, compares them to find the action with the highest score, and returns it.

minPoint() selects the action with the lowest score. Therefore, ***minPoint*** is initially set to positive infinity. It then iterates over all action candidates of the current agent, compares them to find the action with the lowest score, and returns it.

iii. Q3. Alpha-Beta Pruning

The Alpha-Beta Agent is an optimized version of the Minmax Agent, sharing almost identical evaluation logic with the Minmax Agent. However, while the Minmax Agent recursively calculates scores for every node, significantly reducing traversal efficiency with a large number of nodes, the Alpha-Beta Agent employs Alpha-Beta Pruning to avoid traversing unnecessary nodes, thereby enhancing the efficiency of the agent's search process.

The implementation primarily involves three functions: ***alphabeta(), maxPoint()***, and ***minPoint()***.

alphabeta() is broadly similar to the Minmax Agent's ***minmax()*** function but includes two additional parameters, alpha and beta, which serve as references for whether pruning should occur.

Alpha is the lowest score that can be obtained from the action candidates in the current state by ***maxPoint()***;
Beta is the highest score that can be obtained from the action candidates in the current state by ***minPoint()***.

Since ***maxPoint()*** aims to select the action with the highest score, if during its traversal, it encounters a score higher than the historically highest beta value, it can skip (prune) this node. Even without pruning this node, it would not affect the decision outcome of the next layer's ***minPoint()***, saving time that would otherwise be spent traversing this node.

Conversely, ***minPoint()*** aims to select the action with the lowest score, so if during its traversal, it encounters a score lower than the historically lowest alpha value, it can also skip (prune) this node. This is because even without pruning this node, it would not impact the decision outcome of the next layer's ***maxPoint()***, thus saving time.

2. Describe the idea of your design about evaluation function in Q1. (2%)

a. Brief description

The factors influencing Pacman's choice of action mainly include two aspects and two modes. The two aspects are ***Food*** and ***Ghost***, and the two modes are respectively "***chasing food while avoiding ghosts***" and "***chasing both food and ghosts***". Therefore, I have set the parameter ***haunt_ghost*** as a switch to toggle between these two modes. I also defined two other parameters to control Pacman's chasing and avoiding behaviors. The ***haunt_ghost*** parameter is set to 1 by default, which activates the ***haunt_ghost*** mode. Before Pacman acts, it will first iterate through the ***newScaredTimes*** list. If any Ghost is not in a ***"scared"*** state, the ***haunt_ghost*** mode will be deactivated.

b. About "haunt_ghost" mode

When the ***haunt_ghost*** mode is activated, both food and ghosts are targets for Pacman to chase. Therefore, the evaluation function will return the score of the original game state plus individual rewards for ***Food*** and ***Ghosts (successorGameState.getScore() + 1.0/ClosestFoodDistance + 10.0/ClosestGhostDistance)***. The ***Food reward*** is set as the reciprocal of the distance to the closest food because the closer to the food, the higher chance for Pacman to get points. The ***Ghost reward*** is ten times the reciprocal of the distance to the closest ghost because being closer to ghosts is preferred. Moreover, capturing a ghost provides an extra reward, hence the tenfold increase.

When the ***haunt_ghost*** mode is deactivated, Pacman aims to consume food while avoiding ghosts. Therefore, I have introduced an additional parameter called ***"danger".*** If Pacman's distance to a ghost during a particular action is less than a ***predefined danger distance***, the danger parameter switches from its default value of 0 to negative infinity (-float('inf')), preventing actions that would bring Pacman too close to a ghost. This mechanism serves to effectively avoid ghosts.

c. About the “danger” parameter

When the ***haunt_ghost*** mode is deactivated, I set a parameter called ***"danger".*** If Pacman's distance to a ghost during that action is less than the danger distance, the danger value switches from its default of 0 to negative infinity (-float('inf')).

For this ***predefined danger distance***, I tested numbers between 1 to 6. The resulting data from the game showed that the larger the distance, the lower Pacman's sensitivity to avoiding ghosts. However, this also increases the probability of consuming a power pellet, making the ghosts scared and achieving a higher score.

Therefore, if the goal is to score high, a distance of 5 appears to be the optimal danger distance for achieving a high score; if the goal is to win the game, a distance of 1 is the best danger distance for winning the game.

Dangerous Distance	Execution Time (sec)	Average Score	Win Rate	Number of nodes that ***haunt_ghost*** == 1 (the ghost-haunting mode being activated)
1	1.54	1241.5	10/10 (1.00)	0
2	1.55	1239.9	10/10 (1.00)	0
3	1.87	1229.1	10/10 (1.00)	1
4	1.72	1225.3	10/10 (1.00)	1
5	1.84	1275.5	10/10 (1.00)	136
6	2.09	1131.0	9/10 (0.90)	37

3. Demonstrate the speed up after the implementation of pruning. (2%)

a. Theoretically

The searching algorithm logic of the Alpha-Beta Agent is largely similar to that of the Minimax Agent, but the Alpha-Beta Agent employs the method of Alpha-Beta Pruning to avoid traversing unnecessary nodes. This enhances the efficiency of finding the best action and, in theory, should result in faster execution speed compared to the Minimax Agent.

b. Experimentally

I added a time-recording function to the ***grade()*** function in ***grading.py*** to compare the elapsed times for questions 2 and 3. Additionally, I compared the test cases for both questions in the ***AI2024-hw2/test_cases/*** folder and found that they are identical. Therefore, I compared the evaluation time for the entire test process (across all test cases) between the two questions to demonstrate that the Alpha-Beta Agent can finish the game more efficiently than the Minimax Agent, and that the pruning technique indeed speeds up the Alpha-Beta Agent.

i. The elapsed time of Q2 evaluation:

```
=======Evaluating q2 took 0.76 seconds to complete.=======
```

ii. The elapsed time of Q3 evaluation:

```
=======Evaluating q3 took 0.64 seconds to complete.=======
```