Homework 3: Multi-Agent Search

110550108 施柏江

Part I. Implementation

Part 1

```
136         # Begin your code (Part 1)
137         """
138         The "minimax" function first checks if the game is over. If so, it returns the
139         evaluation of the current state. If the game is not over, the function gets the
140         legal actions available to the agent in the current state. For each legal action,
141         the function gets the next game state that would result from taking that action.
142         The function then recursively calls "minimax" for the next agent or next depth.
143         Depending on whether the current agent is Pacman or a ghost, the function chooses
144         the maximum or minimum score. If the current depth is 0, the function returns
145         the action that resulted in the maximum score. Otherwise, it returns the best score.
146         """
147         def minimax(agentIndex, depth, gameState):
148             if gameState.isWin() or gameState.isLose() or depth == self.depth:
149                 return self.evaluationFunction(gameState)
150 
151             scores = []
152             actions = gameState.getLegalActions(agentIndex)
153             for action in actions:
154                 nextGameState = gameState.getNextState(agentIndex, action)
155                 if agentIndex < gameState.getNumAgents() - 1:
156                     score = minimax(agentIndex+1, depth, nextGameState)
157                 else:
158                     score = minimax(0, depth+1, nextGameState)
159                 scores.append(score)
160 
161             if agentIndex > 0:
162                 score = min(scores)
163             elif depth > 0:
164                 score = max(scores)
165             else:
166                 return actions[scores.index(max(scores))]
167             return score
168 
169         return minimax(0, 0, gameState)
170         # End your code (Part 1)
```

Part 2

```
        # Begin your code (Part 2)
        """
        The "alpha_beta" function first checks if the game is over. If so, it returns the
        evaluation of the current state. If the game is not over, the function gets the
        legal actions available to the agent in the current state. For each legal action,
        the function gets the next game state that would result from taking that action.
        The function then recursively calls "alpha_beta" for the next agent or next depth,
        passing in the updated alpha and beta values. It then performs alpha-beta pruning
        by updating the alpha or beta value if the score is lower or higher than the current
        alpha or beta value, respectively. Depending on whether the current agent is
        Pacman or a ghost, the function chooses the maximum or minimum score. If the current
        depth is 0, the function returns the action that resulted in the maximum score.
        Otherwise, it returns the best score.
        """
        def alpha_beta(agentIndex, depth, gameState, alpha, beta):
            if gameState.isWin() or gameState.isLose() or depth == self.depth:
                return self.evaluationFunction(gameState)
            scores = []
            actions = gameState.getLegalActions(agentIndex)
            for action in actions:
                nextGameState = gameState.getNextState(agentIndex, action)
                if agentIndex < gameState.getNumAgents() - 1:
                    score = alpha_beta(agentIndex+1, depth, nextGameState, alpha, beta)
                else:
                    score = alpha_beta(0, depth+1, nextGameState, alpha, beta)
                scores.append(score)

                if agentIndex > 0:
                    if score < alpha:
                        return score
                    beta = min(beta, score)
                else:
                    if score > beta:
                        return score
                    alpha = max(alpha, score)
            if agentIndex > 0:
                score = min(scores)
            elif depth > 0:
                score = max(scores)
            else:
                return actions[scores.index(max(scores))]
            return score

        return alpha_beta(0, 0, gameState, float('-inf'), float('inf'))
        # End your code (Part 2)
```

Part 3

```python
        # Begin your code (Part 3)
        """
        The "expectimax" function first checks if the game is over. If so, it returns the
        evaluation of the current state. If the game is not over, the function gets the
        legal actions available to the agent in the current state. For each legal action,
        the function gets the next game state that would result from taking that action.
        The function then recursively calls "expectimax" for the next agent or next depth.
        Next, we calculate the expected score based on the scores obtained from all
        possible actions, assuming the ghosts choose their actions randomly based on
        some probability distribution. Depending on whether the current agent is Pacman or
        a ghost, the function chooses the maximum or expected score. If the current depth
        is 0, the function returns the action that resulted in the maximum score.
        Otherwise, it returns the score.
        """
        def expectimax(agentIndex, depth, gameState):
            if gameState.isWin() or gameState.isLose() or depth == self.depth:
                return self.evaluationFunction(gameState)
            scores = []
            actions = gameState.getLegalActions(agentIndex)
            for action in actions:
                nextGameState = gameState.getNextState(agentIndex, action)
                if agentIndex < gameState.getNumAgents() - 1:
                    score = expectimax(agentIndex+1, depth, nextGameState)
                else:
                    score = expectimax(0, depth+1, nextGameState)
                scores.append(score)
            if agentIndex > 0:
                score = sum(scores) / len(scores)
            elif depth > 0:
                score = max(scores)
            else:
                return actions[scores.index(max(scores))]
            return score
        return expectimax(0, 0, gameState)
        # End your code (Part 3)
```

Part 4

```
    # Begin your code (Part 4)
    """
    The function starts by getting Pacman's position and the current score, and then
    sets some constants for the values of different game elements. Next, we calculates
    the distances between Pacman and each ghost, food, and capsule on the game board
    using the manhattanDistance function. For each ghost, we check if it is scared or not.
    For the food and capsule, we find the nearest one to Pacman.
    Finally, the function returns the updated score for the current game state.
    """
    pos = currentGameState.getPacmanPosition()
    score = currentGameState.getScore()

    GHOST = -5
    SCARED_GHOST = 200
    FOOD = 10
    CAPSULE = 15
    ghosts = currentGameState.getGhostStates()
    distance = [manhattanDistance(pos, ghost.getPosition()) for ghost in ghosts]
    for dis, ghost in zip(distance, ghosts):
        if dis:
            if ghost.scaredTimer:
                score += SCARED_GHOST / dis
            elif dis < 7:
                score += GHOST / dis

    foods = currentGameState.getFood()
    dis = [manhattanDistance(pos, food) for food in foods.asList()]
    if dis:
        nearestFoodDistance = min(dis)
        score += FOOD / nearestFoodDistance
    else:
        score += FOOD
    capsules = currentGameState.getCapsules()
    dis = [manhattanDistance(pos, capsule) for capsule in capsules]
    if dis:
        nearestCapsuleDistance = min(dis)
        score += CAPSULE / nearestCapsuleDistance
    else:
        score += CAPSULE

    return score
    # End your code (Part 4)
```

Part II. Results & Analysis

Part 1

```
Question part1
==============

*** PASS: test_cases\part1\0-eval-function-lose-states-1.test
*** PASS: test_cases\part1\0-eval-function-lose-states-2.test
*** PASS: test_cases\part1\0-eval-function-win-states-1.test
*** PASS: test_cases\part1\0-eval-function-win-states-2.test
*** PASS: test_cases\part1\0-lecture-6-tree.test
*** PASS: test_cases\part1\0-small-tree.test
*** PASS: test_cases\part1\1-1-minmax.test
*** PASS: test_cases\part1\1-2-minmax.test
*** PASS: test_cases\part1\1-3-minmax.test
*** PASS: test_cases\part1\1-4-minmax.test
*** PASS: test_cases\part1\1-5-minmax.test
*** PASS: test_cases\part1\1-6-minmax.test
*** PASS: test_cases\part1\1-7-minmax.test
*** PASS: test_cases\part1\1-8-minmax.test
*** PASS: test_cases\part1\2-1a-vary-depth.test
*** PASS: test_cases\part1\2-1b-vary-depth.test
*** PASS: test_cases\part1\2-2a-vary-depth.test
*** PASS: test_cases\part1\2-2b-vary-depth.test
*** PASS: test_cases\part1\2-3a-vary-depth.test
*** PASS: test_cases\part1\2-3b-vary-depth.test
*** PASS: test_cases\part1\2-4a-vary-depth.test
*** PASS: test_cases\part1\2-4b-vary-depth.test
*** PASS: test_cases\part1\2-one-ghost-3level.test
*** PASS: test_cases\part1\3-one-ghost-4level.test
*** PASS: test_cases\part1\4-two-ghosts-3level.test
*** PASS: test_cases\part1\5-two-ghosts-4level.test
*** PASS: test_cases\part1\6-tied-root.test
*** PASS: test_cases\part1\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part1\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part1\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\part1\8-pacman-game.test

### Question part1: 20/20 ###
```

Part 2

```
Question part2
==============

*** PASS: test_cases\part2\0-eval-function-lose-states-1.test
*** PASS: test_cases\part2\0-eval-function-lose-states-2.test
*** PASS: test_cases\part2\0-eval-function-win-states-1.test
*** PASS: test_cases\part2\0-eval-function-win-states-2.test
*** PASS: test_cases\part2\0-lecture-6-tree.test
*** PASS: test_cases\part2\0-small-tree.test
*** PASS: test_cases\part2\1-1-minmax.test
*** PASS: test_cases\part2\1-2-minmax.test
*** PASS: test_cases\part2\1-3-minmax.test
*** PASS: test_cases\part2\1-4-minmax.test
*** PASS: test_cases\part2\1-5-minmax.test
*** PASS: test_cases\part2\1-6-minmax.test
*** PASS: test_cases\part2\1-7-minmax.test
*** PASS: test_cases\part2\1-8-minmax.test
*** PASS: test_cases\part2\2-1a-vary-depth.test
*** PASS: test_cases\part2\2-1b-vary-depth.test
*** PASS: test_cases\part2\2-2a-vary-depth.test
*** PASS: test_cases\part2\2-2b-vary-depth.test
*** PASS: test_cases\part2\2-3a-vary-depth.test
*** PASS: test_cases\part2\2-3b-vary-depth.test
*** PASS: test_cases\part2\2-4a-vary-depth.test
*** PASS: test_cases\part2\2-4b-vary-depth.test
*** PASS: test_cases\part2\2-one-ghost-3level.test
*** PASS: test_cases\part2\3-one-ghost-4level.test
*** PASS: test_cases\part2\4-two-ghosts-3level.test
*** PASS: test_cases\part2\5-two-ghosts-4level.test
*** PASS: test_cases\part2\6-tied-root.test
*** PASS: test_cases\part2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part2\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\part2\8-pacman-game.test

### Question part2: 25/25 ###
```

Part 3

```
Question part3
==============

*** PASS: test_cases\part3\0-eval-function-lose-states-1.test
*** PASS: test_cases\part3\0-eval-function-lose-states-2.test
*** PASS: test_cases\part3\0-eval-function-win-states-1.test
*** PASS: test_cases\part3\0-eval-function-win-states-2.test
*** PASS: test_cases\part3\0-expectimax1.test
*** PASS: test_cases\part3\1-expectimax2.test
*** PASS: test_cases\part3\2-one-ghost-3level.test
*** PASS: test_cases\part3\3-one-ghost-4level.test
*** PASS: test_cases\part3\4-two-ghosts-3level.test
*** PASS: test_cases\part3\5-two-ghosts-4level.test
*** PASS: test_cases\part3\6-1a-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1b-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1c-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part3\6-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part3\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\part3\7-pacman-game.test

### Question part3: 25/25 ###
```

Part 4

```
Question part4
==============

Pacman emerges victorious! Score: 1366
Pacman emerges victorious! Score: 1353
Pacman emerges victorious! Score: 1328
Pacman emerges victorious! Score: 1355
Pacman emerges victorious! Score: 1372
Pacman emerges victorious! Score: 1367
Pacman emerges victorious! Score: 1379
Pacman emerges victorious! Score: 1370
Pacman emerges victorious! Score: 1350
Pacman emerges victorious! Score: 1370
Average Score: 1361.0
Scores:        1366.0, 1353.0, 1328.0, 1355.0, 1372.0, 1367.0, 1379.0, 1370.0, 1350.0, 1370.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\part4\grade-agent.test (8 of 8 points)
*** EXTRA CREDIT: 2 points
***     1361.0 average score (4 of 4 points)
***         Grading scheme:
***          < 500:  0 points
***         >= 500:  2 points
***         >= 1000:  4 points
***     10 games not timed out (2 of 2 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 5:  1 points
***         >= 10:  2 points
 ***     10 wins (4 of 4 points)
 ***         Grading scheme:
 ***          < 1:  fail
 ***         >= 1:  1 points
 ***         >= 4:  2 points
 ***         >= 7:  3 points
 ***         >= 10:  4 points

 ### Question part4: 10/10 ###


 Finished at 22:47:36

 Provisional grades
 ==================
 Question part1: 20/20
 Question part2: 25/25
 Question part3: 25/25
 Question part4: 10/10
 ------------------
 Total: 80/80
```

Analysis:

In order to achieve a higher score, it is necessary to eat all the ghosts, so I adjusted the weight of scared ghosts to be particularly high. Furthermore, since capsules allow Pacman to gain the ability to eat ghosts, the weight of capsules is higher than that of food.
When the distance to the ghosts is closer, in order to survive, it is necessary to move away from the ghosts, so the weight of ghosts is negative.