HW 3: Multi-Agent Search

Student ID: 111550100 Name: 邱振源

Part I. Implementation

Part 1:

```
# Begin your code (Part 1)
"""
#Minimax Search

End of Recursion
1. If the depth equals to the given depth equals to the given depth 'self.depth', or the state is the
   winning state or losing state. The recursion should stop by returning the
   'self.evaluationFunction(state)', which can compute the score of state.

Each Recursion
1. Create the list 'Value', which record the score of each state in each recursion.
2. Use for loop to calculate the score of each legal action, the record the score in 'Value'.

Return of The Recursion
1. If it is the root the the decision tree(agent is Pacman and depth == 0), return the index of the
   Pacman's legal actions which has the highest score.
2. If the agent is Pacman, return the maximum value in 'Value'.
3. If the agent is ghost, return the minimum value in 'Value'.
"""

def minimax(state, depth, agentIndex):
    if depth == self.depth or state.isWin() or state.isLose():
        return self.evaluationFunction(state)

    Value = []

    for action in state.getLegalActions(agentIndex):
        nextState = state.getNextState(agentIndex, action)

        if agentIndex == gameState.getNumAgents() - 1:
            Value.append(minimax(nextState, depth + 1, 0))
        else:
            nextagent = agentIndex + 1
            Value.append(minimax(nextState, depth, nextagent))

    if (agentIndex == 0 and depth == 0):
        maxValue = max(Value)
        idx = Value.index(maxValue)
        return idx

    if (agentIndex == 0):
        return max(Value)
    else:
        return min(Value)

return gameState.getLegalActions(0)[minimax(gameState, 0, 0)]
# End your code (Part 1)
```

Part 2:

```
# Begin your code (Part 2)
"""
#Alpha-Beta Pruning

End of Recursion
1. If the depth equals to the given depth equals to the given depth 'self.depth', or the state is the
   winning state or losing state. The recursion should stop by returning the
   'self.evaluationFunction(state)', which can compute the score of state.

Each Recursion
1. If the current agent is Pacman
   a. Initial the 'value' to negative infinity
   b. Use a for loop to do run the recursion for each legal action.
   c. Use 'value' to rememeber the maximum score of 'alpha_beta_pruning()'.
   d. If the value is bigger than 'beta', return value.
   e. Update 'alpha' if 'Value' is larger than 'alpha'.
2. If the current agent is ghost
   a. Initial the 'value' to positive infinity.
   b. Use a for loop to do run the recursion for each legal action.
   c. Use 'value' to rememeber the minimum score of 'alpha_beta_pruning()'.
   d. If the value is smaller than 'alpha', return value.
   e. Update 'beta' if 'Value' is samller than 'beta'.
```

```
#Outside the function

Define the variables
1. Use 'legal_actions' to record the legal action of initial Pacman.
2. Initial 'best_action' to None for return the best action.
3. Initial 'best_value' and 'alpha' to negative infinity, and 'beta' to
   positive infinity.

Recursion Part
1. Use 'value' to record the socre return by 'alpha_beta_pruning'.
2. If the score is larger than 'best_value', record the score and update
   'best_action' to the action of this score.
3. Update 'alpha' if 'Value' is larger than 'alpha'.
"""
```

```python
def alpha_beta_pruning(state, depth, agentIndex, alpha, beta):
    if depth == self.depth or state.isWin() or state.isLose():
        return self.evaluationFunction(state)

    if agentIndex == 0:
        value = float("-inf")
        for action in state.getLegalActions(agentIndex):
            next_state = state.getNextState(agentIndex, action)
            value = max(value, alpha_beta_pruning(next_state, depth, agentIndex + 1, alpha, beta))
            if value > beta:
                return value
            alpha = max(alpha, value)
        return value
    else:
        value = float("inf")
        for action in state.getLegalActions(agentIndex):
            next_state = state.getNextState(agentIndex, action)
            if agentIndex == gameState.getNumAgents() - 1:
                value = min(value, alpha_beta_pruning(next_state, depth + 1, 0, alpha, beta))
            else:
                value = min(value, alpha_beta_pruning(next_state, depth, agentIndex + 1, alpha, beta))
            if value < alpha:
                return value
            beta = min(beta, value)
        return value

legal_actions = gameState.getLegalActions(0)
best_action = None
best_value = float("-inf")
alpha = float("-inf")
beta = float("inf")

for action in legal_actions:
    next_state = gameState.getNextState(0, action)
    value = alpha_beta_pruning(next_state, 0, 1, alpha, beta)
    if value > best_value:
        best_value = value
        best_action = action
    alpha = max(alpha, best_value)

return best_action
# End your code (Part 2)
```

Part 3:

```
# Begin your code (Part 3)
"""
#Expectimax Search

End of Recursion
1. If the depth equals to the given depth equals to the given depth 'self.depth', or the state is the
   winning state or losing state. The recursion should stop by returning the
   'self.evaluationFunction(state)', which can compute the score of state.

Each Recursion
1. Create the list 'Value', which record the score of each state in each recursion.
2. Use for loop to calculate the score of each legal action, the record the score in 'Value'.

Return of The Recursion
1. If it is the root the the decision tree(agent is Pacman and depth == 0), return the index of the
   Pacman's legal actions which has the highest score.
2. If the agent is Pacman, return the maximum value in 'Value'.
3. If the agent is ghost, return the mean value of 'Value' since the ghosts' action is random.
"""
def expectimax(state, depth, agentIndex):
    if depth == self.depth or state.isWin() or state.isLose():
        return self.evaluationFunction(state)

    Value = []

    for action in state.getLegalActions(agentIndex):
        nextState = state.getNextState(agentIndex, action)

        if agentIndex == gameState.getNumAgents() - 1:
            Value.append(expectimax(nextState, depth + 1, 0))
        else:
            nextagent = agentIndex + 1
            Value.append(expectimax(nextState, depth, nextagent))

    if (agentIndex == 0 and depth == 0):
        maxValue = max(Value)
        idx = Value.index(maxValue)
        return idx

    if (agentIndex == 0):
        return max(Value)
    else:
        return sum(Value)/len(Value)

return gameState.getLegalActions(0)[expectimax(gameState, 0, 0)]
# End your code (Part 3)
```

Part 4:

```
# Begin your code (Part 4)
"""
Define Variables
1. 'score' for current score, 'currentPosition' for current position, 'list_of_food' is
   a list record current foods, 'list_of_capsule' record current capsules, 'state_of_ghost'
   record current ghosts
2. 'number_of_food' record the number of food, initial 'min_food_distance', 'min_capsule_distance',
   and 'min_scare_ghost' as negative infinity.

Calculate Variables
1. 'min_food_distance' would be the minimum distance of food.
2. 'min_capsule_distance' would be the minimum distance of capsule.
3. If the Pacman was eaten by the ghost the score should be negative infinity.
4. 'min_scare_ghost' would be the minimum distance of scared ghost.

Evaluate Score
1. Consider current score.
2. If the distance of food, capsule, or scared ghost is shorter, the score will be higher.
3. If the number of food is more, the score should be lower.
"""
score = currentGameState.getScore()
currentPosition = currentGameState.getPacmanPosition()
list_of_food = currentGameState.getFood().asList()
list_of_capsule = currentGameState.getCapsules()
state_of_ghost = currentGameState.getGhostStates()

number_of_food = len(list_of_food)
min_food_distance = float('inf')
min_capsule_distance = float('inf')
min_scare_ghost = float('inf')

for food in list_of_food:
    min_food_distance = min(min_food_distance, manhattanDistance(currentPosition, food))
for capsule in list_of_capsule:
    min_capsule_distance = min(min_capsule_distance, manhattanDistance(currentPosition, capsule))
for ghost in state_of_ghost:
    if manhattanDistance(currentPosition, ghost.getPosition()) == 0 and ghost.scaredTimer == 0:
        return float("-inf")
    if ghost.scaredTimer > 0:
        min_scare_ghost = min(min_scare_ghost, manhattanDistance(currentPosition, ghost.getPosition()))

food_score = 10 / min_food_distance
capsule_score = 50 / min_capsule_distance
ghost_score = 200 / min_scare_ghost

return score + food_score + capsule_score + ghost_score - number_of_food
# 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: 15/15 ###
```

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: 20/20 ###
```

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: 20/20 ###
```

Part 4:

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

Pacman emerges victorious! Score: 1263
Pacman emerges victorious! Score: 1293
Pacman emerges victorious! Score: 1305
Pacman emerges victorious! Score: 1356
Pacman emerges victorious! Score: 1246
Pacman emerges victorious! Score: 1313
Pacman emerges victorious! Score: 1262
Pacman emerges victorious! Score: 1318
Pacman emerges victorious! Score: 1331
Pacman emerges victorious! Score: 1236
Average Score: 1292.3
Scores:        1263.0, 1293.0, 1305.0, 1356.0, 1246.0, 1313.0, 1262.0, 1318.0, 1331.0, 1236.0
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
***     1292.3 average score (4 of 4 points)
***         Grading scheme:
***          < 600:  0 points
***         >= 600:  2 points
***         >= 1200:  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 3:52:50

Provisional grades
==================
Question part1: 15/15
Question part2: 20/20
Question part3: 20/20
Question part4: 10/10
------------------
Total: 65/65
```

Discussion

In my scoring formula, because being eaten by a ghost would lead to an immediate loss, I set the score for the case where the distance between Pacman and the ghost is 0 to negative infinity and return it directly. Furthermore, since we aim to maximize the

game score, I first consider the current score in the formula. The game score is a parameter that continuously decreases, so we should consider other parameters to make the score of each state different. Under normal circumstances, we can divide Pacman's actions into: eating food, eating capsules, eating ghosts. Eating food and eating ghosts can earn 10 points and 200 points respectively. Therefore, I define the food score as 10 divided by the distance from Pacman to the closest food, and the ghost score as 200 divided by the distance from Pacman to the closest scared ghost. Additionally, because I want Pacman to be more motivated to eat capsules in order to eat ghosts and earn a higher score, I temporarily define the capsule score as 50 divided by the distance to the closest capsule. So, my initial formula can be set as:

$$\begin{cases} -\infty, & where \min ghost\ dost = 0 \\ \text{score} + \dfrac{10}{\min food\ dist} + \dfrac{50}{\min capsule\ dist} + \dfrac{200}{\min ghost\ dist}, & otherwose \end{cases}$$

After defining the formula, I once attempted to change the parameters, such as modifying the numerator of the capsule score to 20 or 70, or changing the portion of eating ghost score to 150. However, these adjustments sometimes caused Pacman to remain stationary for extended periods, resulting in excessively long completion times for scoring the game. Therefore, I decided to leave the constants as 10, 50, and 200 respectively without further modification.

However, since Pacman tends to stop moving a bit too often, which is not common during normal Pacman gameplay, I sought to incorporate more parameters in the formula that would decrease over time, encouraging Pacman to move instead of staying still and prompting it to eat more food. Therefore, I decided to use the quantity of food as a factor. After testing various multiplicative factors, I selected the result with the highest average score, which turned out to be multiplying by 1. Finally, we can define the formula as follows:

$$\text{score} + \frac{10}{\min food\ dist} + \frac{50}{\min capsule\ dist} + \frac{200}{\min ghost\ dist} - number\ of\ food$$

Finally, I'd like to share an interesting observation. When using my evaluate function, you may notice that in the window generated during execution, when Pacman approaches a capsule to a certain distance, he will stop moving and wait for the ghost to approach. Once he ensures that he can eat the capsule and then catch up to the ghost, he will consume the capsule and eliminate the ghost. I find this behavior of waiting intriguing.