109550069_林宇柔

Report HW3

Implementation

```
# Begin your code (Part 1)

def minimum(state, agentIndex, depth):
    if state.isLose() or state.isWin(): # If the game is finished(win or lose)
        return self.evaluationFunction(state) # go to the evaluation function

    if agentIndex == state.getNumAgents() - 1: # If it is the last ghost
        return min(maximum(state.getNextState(agentIndex, action), depth) for action in state.getLegalActions(agentIndex))
        # Take the minimum value of the maximum function with the depth
    else:
        return min(minimum(state.getNextState(agentIndex, action), agentIndex + 1, depth) for action in
                   state.getLegalActions(agentIndex))
        # If not, keep taking the minimum value of this function with the next agent

def maximum(state, depth):
    if state.isLose() or state.isWin() or depth == self.depth: # If the game is finished(win or lose or no more depth)
        return self.evaluationFunction(state) # go to the evaluation function

    return max(minimum(state.getNextState(0, action), 1, depth + 1) for action in state.getLegalActions(0))
    # Take the maximum value of the minimum function on the pacman and the next depth

best = max(gameState.getLegalActions(0),
           key=lambda action: minimum(gameState.getNextState(0, action), 1, 1))
# The best action is the maximum sorted by action in minimum function
return best

raise NotImplementedError("To be implemented")
# End your code (Part 1)
```

```
# Begin your code (Part 3)

def expectation(state, agentIndex, depth):
    if state.isLose() or state.isWin(): # If the game is finished(win or lose)
        return self.evaluationFunction(state) # go to the evaluation function

    probability = 1.0 / len(state.getLegalActions(agentIndex)) # the probability is one out of total num of actions
    val = 0
    for action in state.getLegalActions(agentIndex):
        if agentIndex == state.getNumAgents() - 1: # If it is the last ghost
            val += maximum(state.getNextState(agentIndex,
                                              action), depth) * probability # Add the maximum function times probability
        else:
            val += expectation(state.getNextState(agentIndex, action), agentIndex +
                               1, depth) * probability # If not, add the expectation with the next agent
    return val

def maximum(state, depth):
    if state.isLose() or state.isWin() or depth == self.depth: # If the game is finished(win or lose or no more depth)
        return self.evaluationFunction(state) # go to the evaluation function

    val = max(expectation(state.getNextState(0, action), 1, depth + 1)
              for action in state.getLegalActions(0)) # Take the maximum value of expectation on the next depth
    return val

best = max(gameState.getLegalActions(), key=lambda action: expectation(
    gameState.getNextState(0, action), 1, 1)) #
return best # The best action is the maximum sorted by action in expectation function

raise NotImplementedError("To be implemented")
# End your code (Part 3)
```

```
# Begin your code (Part 2)
def minimum(state, agentIndex, depth, a, b):
    if state.isLose() or state.isWin(): # If the game is finished(win or lose)
        return self.evaluationFunction(state) # go to the evaluation function

    v = float("inf")
    for action in state.getLegalActions(agentIndex):
        if agentIndex == state.getNumAgents() - 1: # If it is the last ghost
            newV = maximum(state.getNextState(
                agentIndex, action), depth, a, b) # record the return value of maximum function
        else: # If not
            newV = minimum(state.getNextState(
                agentIndex, action), agentIndex + 1, depth, a, b) # record the value of minimum function with the next agent

        v = min(v, newV) # update current value with minimum between current value and the new value
        if v < a: # If the value is smaller than the alpha
            return v
        b = min(b, v) # Save beta as the minimum between beta and current value
    return v
```

```
def maximum(state, depth, a, b):
    if state.isLose() or state.isWin() or depth == self.depth: # If the game is finished(win or lose or no more depth)
        return self.evaluationFunction(state) # go to the evaluation function

    v = float("-inf")
    # pruning
    if depth == 0: # If at the first layer
        best = state.getLegalActions(0) # Take the pacman's state
    for action in state.getLegalActions(0): # For the pacman's actions
        newV = minimum(state.getNextState(
            0, action), 1, depth + 1, a, b) # record the value of minimum function on the pacman and the next depth
        if newV > v: # Update the value if the new one is greater than current
            v = newV
            if depth == 0: # And if at the first layer
                best = action # Take this action
        if v > b: # Return the current value if it is greater than beta
            return v
        a = max(a, v) # Save alpha as the maximum between alpha and current value

    if depth == 0: # If at the first layer
        return best
    return v

best = maximum(gameState, 0, float("-inf"), float("inf")) # Start by taking maximum with layer 0, large alpha beta
return best

raise NotImplementedError("To be implemented")
# End your code (Part 2)
```

```
# Begin your code (Part 4)

pacman = currentGameState.getPacmanPosition()  # Get the pacman's position
foods = currentGameState.getFood().asList() # Store the position of the foods in a list
ghosts = currentGameState.getGhostStates()  # Get the position of the ghosts
closestFoodDis = min(manhattanDistance(pacman, food)
                     for food in foods) if foods else 5 # Get the distance of the closest food
closestGhostDis = min(manhattanDistance(pacman, ghost.getPosition())
                      for ghost in ghosts) # Get the distance of the closest ghost
score = currentGameState.getScore() # Get the score
evaluation = 1.0 / closestFoodDis + score - closestGhostDis
"""
1. The closer the food is, the greater the fraction is
2. add up the original score
3. Cut off the closest ghost's distance
"""
return evaluation

raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

Results & Analysis

```
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 ###
```

```
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 ###
```

```
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 ###
```

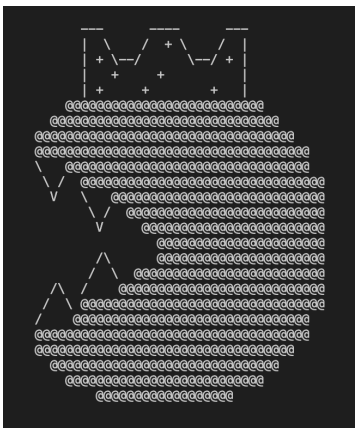
```
Question part4
==============

Pacman emerges victorious! Score: 1098
Pacman emerges victorious! Score: 1152
Pacman emerges victorious! Score: 903
Pacman emerges victorious! Score: 1098
Pacman emerges victorious! Score: 1331
Pacman emerges victorious! Score: 969
Pacman emerges victorious! Score: 1325
Pacman emerges victorious! Score: 1160
Pacman emerges victorious! Score: 1132
Pacman emerges victorious! Score: 947
Average Score: 1111.5
Scores:        1098.0, 1152.0, 903.0, 1098.0, 1331.0, 969.0, 1325.0, 1160.0, 1132.0, 947.0
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
***     1111.5 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 15:02:13

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



For the evaluation function, when the position of the ghosts are not considered, I can get greater scores than the previously showed scores, I think this is because of the way I calculate it, I tried subtraction and fractions and subtraction worked better, but I think it is important to take this into consideration because we not only have to check the position of the food so I implemented it anyway.