109550182 莊婕妤

AI Homework3 Report

Part I. Implementation (5%)

Part 1 Minimax Agent

```
# Begin your code (Part 1)

# the recursion function implementing the minimax agent
def implementMinimax(depth, agent, state):

    # return the evaluation score when Pacman comes to the end state or overceeding the depth limit
    if state.isWin() or state.isLose() or depth > self.depth:
        return self.evaluationFunction(state)

    # getting all legal actions
    actions = state.getLegalActions(agent)
    # storing the score of every possible action in a list
    actionScores = []
    for action in actions:
        nextState = state.getNextState(agent, action)
        # all agents have moved -> back to Pacman's turn and start another round
        if (agent + 1) == state.getNumAgents():
            actionScores.append(implementMinimax(depth + 1, 0, nextState))
        # agent taking turns
        else: actionScores.append(implementMinimax(depth, agent + 1, nextState))

    # performing the minimax procedure
    # 1. Pacman : return the maximum action score
    if agent == 0:
        if depth == 1: # return the next action when it comes back to the root
            for i in range(len(actionScores)):
                if actionScores[i] == max(actionScores): return actions[i]
        else: actionScore = max(actionScores)
    # 2. Ghosts : return the minimum action score
    else: actionScore = min(actionScores)
    return actionScore

# implement minimax agent
return implementMinimax (1, 0, gameState)
# End your code (Part 1)
```

109550182 莊婕妤

Part 2 Alpha-Beta Pruning

```
# Begin your code (Part 2)
def implementAlphaBeta(depth, agent, state, alpha, beta):

    # return the evaluation score when Pacman comes to the end state or overceeding the depth limit
    if (state.isWin() or state.isLose() or depth > self.depth):
        return self.evaluationFunction(state)

    # getting all legal actions
    actions = state.getLegalActions(agent)
    # storing the score of every possible action in a list
    actionScores = []
    for action in actions:
        nextState = state.getNextState(agent, action)
        # all agents have moved -> back to Pacman's turn and start another round
        if (agent + 1) == state.getNumAgents():
            actionScore = implementAlphaBeta(depth + 1, 0, nextState, alpha, beta)
            actionScores.append(actionScore)
        # agent taking turns
        else:
            actionScore = implementAlphaBeta(depth, agent + 1, nextState, alpha, beta)
            actionScores.append(actionScore)

        # pruning the branches
        # 1. for Pacman / the maximizer
        if agent == 0:
            if actionScore > beta: return actionScore
            alpha = max(alpha, actionScore)
        # 2. for the ghosts / the minimizer
        else:
            if actionScore < alpha: return actionScore
            beta = min(beta, actionScore)
```

```
    # performing the minimax procedure
    # 1. Pacman : return the maximum action score
    if agent == 0:
        if depth == 1: # return the next action when it comes back to the root
            for i in range(len(actionScores)):
                if actionScores[i] == max(actionScores): return actions[i]
        else: actionScore = max(actionScores)
    # 2. Ghosts : return the minimum action score
    else: actionScore = min(actionScores)
    return actionScore

# initialize alpha & beta
alpha = -99999
beta = 99999
# implement alpha-beta pruning
return implementAlphaBeta(1, 0, gameState, alpha, beta)
# End your code (Part 2)
```

109550182 莊婕妤

Part 3 Expectimax Search

```
# Begin your code (Part 3)
def implementExpectimax(depth, agent, state):

    # return the evaluation score when Pacman comes to the end state or overceeding the depth limit
    if (state.isWin() or state.isLose() or depth > self.depth):
        return self.evaluationFunction(state)

    # getting all legal actions
    actions = state.getLegalActions(agent)
    # storing the score of every possible action in a list
    actionScores = []
    for action in actions:
        nextState = state.getNextState(agent, action)
        # all agents have moved -> back to Pacman's turn and start another round
        if (agent + 1) == state.getNumAgents():
            actionScores.append(implementExpectimax(depth + 1, 0, nextState))
        # agent taking turns
        else: actionScores.append(implementExpectimax(depth, agent + 1, nextState))


    # performing the expectimax procedure
    # 1. Pacman : return the maximum action score
    if agent == 0:
        if depth == 1: # return the next action when it comes back to the root
            for i in range(len(actionScores)):
                if actionScores[i] == max(actionScores): return actions[i]
        else: actionScore = max(actionScores)
    # 2. Ghosts : choose legal action uniformly at random
    #    -> now returning the average case instead of the minimum
    else: actionScore = float(sum(actionScores) / len(actionScores))
    return actionScore

# implement the expectimax agent
return implementExpectimax(1, 0, gameState)
# End your code (Part 3)
```

109550182 莊婕妤

Part 4 Evaluation Function

```
# Begin your code (Part 4)

# Accessing useful information for my evaluation function :
score = currentGameState.getScore() # current score
position = currentGameState.getPacmanPosition() # Pacman's current position
food = currentGameState.getFood() # list of foods
capsules = currentGameState.getCapsules() # list of capsules
ghostStates = currentGameState.getGhostStates() # ghosts' states

# Giving different weights to some particular states
# -> to control Pacman's action / strategy
WEIGHT_FOOD = 10.0
WEIGHT_CAPSULE = 25.0
WEIGHT_GHOST = -10.0
WEIGHT_SCARED_GHOST = 300.0
```

```
# set a higher score while approaching capsules
capsuleDistances = [manhattanDistance(position, capsulePosition) for capsulePosition in capsules]
if len(capsuleDistances): score += WEIGHT_CAPSULE / min(capsuleDistances)
else: score += WEIGHT_FOOD

# set a higher score while approaching food
foodDistances = [manhattanDistance(position, foodPosition) for foodPosition in food.asList()]
if len(foodDistances): score += WEIGHT_FOOD / min(foodDistances)
else: score += WEIGHT_FOOD

# interactions with the ghosts
for ghost in ghostStates:
    distance = manhattanDistance(position, ghost.getPosition())
    if distance > 0:
        # set a higher score if approaching scared ghosts
        if ghost.scaredTimer > 0: score += WEIGHT_SCARED_GHOST / distance
        # lower the score when the ghost is close
        else: score += WEIGHT_GHOST / distance

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

Part II. Results & Analysis (5%)

Part 1. Minimax Agent

```
*** 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. Alpha-Beta Pruning

```
*** 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. Expectimax Agent

```
*** 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 ###
```

Analysis for Part 1 to 3

Since all these agents use the same evaluation criteria, their results are quite similar. Although their processing time may be slightly different. For example, the Alpha-Beta Pruning may take less time compared to the others.

Part 4. Better Evaluation Function

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

Pacman emerges victorious! Score: 1366
Pacman emerges victorious! Score: 1364
Pacman emerges victorious! Score: 1362
Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1358
Pacman emerges victorious! Score: 1358
Pacman emerges victorious! Score: 1319
Pacman emerges victorious! Score: 1340
Pacman emerges victorious! Score: 1339
Pacman emerges victorious! Score: 1369
Average Score: 1354.8
Scores:        1366.0, 1364.0, 1362.0, 1373.0, 1358.0, 1358.0, 1319.0, 1340.0, 1339.0, 1369.0
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
***     1354.8 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 11:37:55

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

Observation and Analysis

I added different weights to different states in order to control Pacman's actions. My strategy is to let Pacman go for the capsules first and then eat up the scared ghosts quickly by assigning larger weights to WEIGHT_CAPSULE and WEIGHT_SCARED_GHOST. Also, I set higher scores for states that are closer to food and states that are far from not scared ghosts.

From the result, we can see that this way of evaluating states makes Pacman more intelligent and wins the game with a high probability. I think the difference between my evaluation function and the original one is that I handled the case of scared ghosts which leads to a higher score.