Homework 3: Multi-Agent Search

Please keep the title of each section and delete examples.

Part I. Implementation (5%):

- Please screenshot your code snippets of Part 1 ~ Part 4 and explain your implementation.

Part 1 (Minimax Agent)

```
    # Begin your code (Part 1)
    result = self.value(gameState, 0, 0)
    return result[1]                                    # returns 'action' from result

def value(self, gameState, index, depth):               # returns value: pair of [score, action]
    if len(gameState.getLegalActions(index)) == 0 or depth == self.depth:   # for terminal states
        action = ""
        score = self.evaluationFunction(gameState)
        return score, action

    if index == 0: return self.max_value(gameState, index, depth)          # if Pacman (index = 0), use max-agent
    else:          return self.min_value(gameState, index, depth)          # else (index > 0 => Ghost), use min-agent

def max_value(self, gameState, index, depth):           # returns the max utility value-action for max-agent
    legal_actions = gameState.getLegalActions(index)                       # get all possible actions for all agents
    max_v = float("-inf")                                                  # initialize v as -infinity
    max_action = ""

    for action in legal_actions:                                           # for each action in the possible actions

        #          [0,1,2,3,4]
        #            /      \
        # [0,1,2,3,4]       [0,1,2,3,4]

        nextstate = gameState.getNextState(index, action)                  # get the next state after the agent take an action

        if index + 1 == gameState.getNumAgents():                          # if last agent for that depth, go to next depth (4+1 = 5)
            nextstate_index = 0                                            # starting from index 0 (pacman)
            nextstate_depth = depth + 1                                    # increment depth
        else:                                                              # else continue going through the agents
            nextstate_index = index + 1                                    # by adding 1 to the index
            nextstate_depth = depth                                        # remain in the same depth

        curr_v = self.value(nextstate, nextstate_index, nextstate_depth)[0] # current value from value of the next state

        if curr_v > max_v:                                                 # if current value bigger than maximum value
            max_v = curr_v                                                 # update max_v as curr_v and use the action in loop
            max_action = action

    return max_v, max_action
```

```
def min_value(self, gameState, index, depth):           # returns the min utility value-action for min-agent
    legal_actions = gameState.getLegalActions(index)                       # get all possible actions for all agents
    min_v = float("inf")                                                   # initialize v as +infinity
    min_action = ""

    for action in legal_actions:                                           # for loop similar to max_value
        nextstate = gameState.getNextState(index, action)

        if index + 1 == gameState.getNumAgents():
            nextstate_index = 0
            nextstate_depth = depth + 1
        else:
            nextstate_index = index + 1
            nextstate_depth = depth

        curr_v = self.value(nextstate, nextstate_index, nextstate_depth)[0]

        if curr_v < min_v:                                                 # if current value smaller than minimum value, update the min_v and the action
            min_v = curr_v
            min_action = action

    return min_v, min_action

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

Part 2 (Alpha-Beta Agent)

```
    # Begin your code (Part 2)
    alpha = float("-inf")                                   # initialize alpha as -infinity
    beta = float("inf")                                     # initialize beta as +infinity
    result = self.value(gameState, 0, 0, alpha, beta)
    return result[1]                                        # returns 'action' from the result

def value(self, gameState, index, depth, alpha, beta):      # returns value: pair of [score, action]
    if len(gameState.getLegalActions(index)) == 0 or depth == self.depth:
        action = ""
        score = self.evaluationFunction(gameState)
        return score, action

    if index == 0: return self.max_value(gameState, index, depth, alpha, beta)
    else:          return self.min_value(gameState, index, depth, alpha, beta)

def max_value(self, gameState, index, depth, alpha, beta):  # returns the max utility value-action for max-agent
    legal_actions = gameState.getLegalActions(index)
    max_v = float("-inf")
    max_action = ""

    for action in legal_actions:                            # similar to minimax except the alpha-beta (including for min_value too)
        nextstate = gameState.getNextState(index, action)

        if index + 1 == gameState.getNumAgents():
            nextstate_index = 0
            nextstate_depth = depth + 1
        else:
            nextstate_index = index + 1
            nextstate_depth = depth

        curr_v = self.value(nextstate, nextstate_index, nextstate_depth, alpha, beta)[0]

        if curr_v > max_v:
            max_v = curr_v
            max_action = action

        alpha = max(alpha, max_v)                           # update alpha for this function max_value
        if max_v > beta:                                    # Optimisation: pruning to stop evaluating game state
            return max_v, max_action

    return max_v, max_action
```

```
def min_value(self, gameState, index, depth, alpha, beta):  # returns the min utility value-action for min-agent
    legal_actions = gameState.getLegalActions(index)
    min_v = float("inf")
    min_action = ""

    for action in legal_actions:
        nextstate = gameState.getNextState(index, action)

        if index + 1 == gameState.getNumAgents():
            nextstate_index = 0
            nextstate_depth = depth + 1
        else:
            nextstate_index = index + 1
            nextstate_depth = depth

        curr_v = self.value(nextstate, nextstate_index, nextstate_depth, alpha, beta)[0]

        if curr_v < min_v:                                  # if current value smaller than minimum value, update the min_v and the action
            min_v = curr_v
            min_action = action

        beta = min(beta, min_v)                             # update beta for this function min_value
        if min_v < alpha:                                   # Optimisation: pruning to stop evaluating game state
            return min_v, min_action

    return min_v, min_action

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

Part 3 (Expectimax Agent)

```
    # Begin your code (Part 3)
    score, action = self.value(gameState, 0, 0)
    return action

def value(self, gameState, index, depth):           # returns value: pair of [score, action]
    if len(gameState.getLegalActions(index)) == 0 or depth == self.depth:
        score = self.evaluationFunction(gameState)
        action = ""
        return score, action

    if index == 0: return self.max_value(gameState, index, depth)
    else:          return self.expected_value(gameState, index, depth)      # if index > 0 (ghost), take expected value

def max_value(self, gameState, index, depth):       # returns the max utility value-action for max-agent
    legal_actions = gameState.getLegalActions(index)
    max_v = float("-inf")
    max_action = ""

    for action in legal_actions:
        nextstate = gameState.getNextState(index, action)

        if index + 1 == gameState.getNumAgents():
            nextstate_index = 0
            nextstate_depth = depth + 1
        else:
            nextstate_index = index + 1
            nextstate_depth = depth

        curr_v = self.value(nextstate, nextstate_index, nextstate_depth)[0]

        if curr_v > max_v:
            max_v = curr_v
            max_action = action

    return max_v, max_action
```

```
def expected_value(self, gameState, index, depth):
    legal_actions = gameState.getLegalActions(index)
    exp_v = 0                                                   # initialize the expected value as 0
    exp_action = ""


    nextstate_p = 1.0 / len(legal_actions)                      # next state's probability or weight by uniform distribution

    for action in legal_actions:
        nextstate = gameState.getNextState(index, action)

        if index + 1 == gameState.getNumAgents():
            nextstate_index = 0
            nextstate_depth = depth + 1
        else:
            nextstate_index = index + 1
            nextstate_depth = depth

        curr_v = self.value(nextstate, nextstate_index, nextstate_depth)    # current value from value of the next state

        # Update expected_value with the current_value and successor_probability
        exp_v += nextstate_p * curr_v                           # get the expected value from their weights times current value

    return exp_v, exp_action

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

Part 4 (Better Evaluation Function)

```python
def betterEvaluationFunction(currentGameState):
    """
    Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
    evaluation function (Part 4).
    """
    # Begin your code (Part 4)

    # get positions of the agents
    pos = currentGameState.getPacmanPosition()
    ghost_pos_list = currentGameState.getGhostPositions()

    # get the current score
    curr_score = currentGameState.getScore()

    # get food distances
    foods = currentGameState.getFood().asList()
    food_distances = [manhattanDistance(pos, food_position) for food_position in foods]

    nearest_food_dist = 1
    if len(foods) > 0:                                                  # if there's still food
        nearest_food_dist = min(food_distances)                         # find the nearest food distance from the food_distances

    min_ghost_dist = min(manhattanDistance(pos, ghost_pos) for ghost_pos in ghost_pos_list) # the closest ghost distance
    if min_ghost_dist < 2:                                              # if the ghost is one step away
        nearest_food_dist = 10000000        # let the nearest food distance be really big and it will prioritize of running away instead of eating the nearest food

    features = [curr_score, 1.0 / nearest_food_dist, len(foods), len(currentGameState.getCapsules())]   # factors that affects the pacman
    weights = [10, 1, -1, -1]                                                                           # weights are tuned by experiment

    return sum([feature * weight for feature, weight in zip(features, weights)])            # sum of all factors times by their weights

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

# Abbreviation
better = betterEvaluationFunction
```

Part II. Results & Analysis (5%):

- Please screenshot the results. For instance, the result of the autograder and any observation of your evaluation function.

Part 1 & Part 2

```
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 ###
```

For part 1 and part 2, I think the implementation is quite similar, except the part that Alpha-Beta Pruning use alpha and beta to prune or to stop evaluating further states. I tried running Alpha-Beta on small classic with depth=3 and win with score of 1349 and duration 63 seconds. Then, I tried running Minimax Agent on small classic with depth=2 and lose with score of 42 and duration of 66 seconds. After several attempts, I found that alpha-beta pruning is the faster way of minimax agent because it does not visit all the nodes unless it's needed.

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 ###
```

The difference of expectimax search and the previous two searches is that this search use the function max-value and expected-value, instead of using max and min-value functions. This search uses weight or probability to make the choices.

```
PS C:\Users\Cindy\Downloads\AI\HW\AI_HW3\AI_HW3\AI_HW3> python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Pacman died! Score: -501
Average Score: -501.0
Scores:        -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0, -501.0
Win Rate:      0/10 (0.00)
Record:        Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss, Loss
```

```
PS C:\Users\Cindy\Downloads\AI\HW\AI_HW3\AI_HW3\AI_HW3> python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Pacman died! Score: -502
Average Score: -88.4
Scores:        -502.0, 532.0, 532.0, -502.0, 532.0, 532.0, -502.0, -502.0, -502.0, -502.0
Win Rate:      4/10 (0.40)
Record:        Loss, Win, Win, Loss, Win, Win, Loss, Loss, Loss, Loss
```

We can see that expectimax agent win rate is higher than that of alpha-beta agent, and alpha-beta agent always loses. Compared to minimax agent, expectimax agent reflect average-case and minimax agent reflect worst-case outcomes.

Part 4

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

Pacman emerges victorious! Score: 1344
Pacman emerges victorious! Score: 1244
Pacman emerges victorious! Score: 1049
Pacman emerges victorious! Score: 883
Pacman emerges victorious! Score: 826
Pacman emerges victorious! Score: 1284
Pacman emerges victorious! Score: 1088
Pacman emerges victorious! Score: 964
Pacman emerges victorious! Score: 794
Pacman emerges victorious! Score: 1269
Average Score: 1074.5
Scores:        1344.0, 1244.0, 1049.0, 883.0, 826.0, 1284.0, 1088.0, 964.0, 794.0, 1269.0
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
***     1074.5 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 ###
```

After trying several attempts on deciding the weight for the features to get a better evaluation function result, I came out with the numbers and managed to pass the test cases with score more than one thousand. It is also able to run in a reasonable duration of time and get a decent result.

Overall result:

```
Finished at 11:27:40

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

--- 李嘉玲 109550186