Homework 3: Multi-Agent Search

Please keep the title of each section and delete examples.

Part I. Implementation (5%):

Part1

```python
# Begin your code (Part 1)
"""
Using recursion to compute the minimax values of each state. The minimax function takes state, depth, agent index as arguments,
and returns minimax value of the state. If the state is a terminal state or the maximum depth has been reached, the function
returns the evaluation of the state. Otherwise, the function computes the minimax value for the current agent by either
maximizing or minimizing the values of the next agent's actions, depending on the current agent is Pacman or a ghost.
Finally, the function returns the computed minimax value.
"""
def minimax(state, depth, agentIndex):
    if state.isWin() or state.isLose() or depth == self.depth:
        return self.evaluationFunction(state)
    if agentIndex == 0:  # maximize for pacman
        v = float('-inf')
        for action in state.getLegalActions(agentIndex):
            v = max(v, minimax(state.getNextState(agentIndex, action), depth, agentIndex + 1))
        return v
    else:  # minimize for ghost
        v = float('inf')
        for action in state.getLegalActions(agentIndex):
            if agentIndex == state.getNumAgents() - 1:  # next agent is pacman, depth + 1
                v = min(v, minimax(state.getNextState(agentIndex, action), depth + 1, 0))
            else:  #move to next ghost
                v = min(v, minimax(state.getNextState(agentIndex, action), depth, agentIndex + 1))
        return v
legalMoves = gameState.getLegalActions()  # get all legal actions
# get the score for each legal action by running minimax on the resulting state
scores = [minimax(gameState.getNextState(0, action), 0, 1) for action in legalMoves]
# find the best score and choose a random action with the best score
bestScore = max(scores)
bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
chosenIndex = random.choice(bestIndices)
return legalMoves[chosenIndex]
# End your code (Part 1)
```

Part2:

```python
        # Begin your code (Part 2)
        """
        maxValue function returns the maximum value achievable by the current agent in the current state,
        while minValue function returns the minimum value achievable by the next agent.
        The main loop iterates through all legal actions of the current agent and calculates the value
        of each action using the minValue function, and updates the best action accordingly. It uses
        alpha-beta pruning to avoid exploring paths that will not lead to a better outcome.
        Finally, return the best action.
        """
        def maxValue(state, agentIndex, depth, alpha, beta):
            v = float("-inf")
            legalActions = state.getLegalActions(agentIndex)
            if not legalActions or depth == self.depth:
                return self.evaluationFunction(state)
            for action in legalActions:
                nextState = state.getNextState(agentIndex, action)
                v = max(v, minValue(nextState, agentIndex + 1, depth, alpha, beta))
                if v > beta:
                    return v
                alpha = max(alpha, v)
            return v

        def minValue(state, agentIndex, depth, alpha, beta):
            v = float("inf")
            legalActions = state.getLegalActions(agentIndex)
            if not legalActions or depth == self.depth:
                return self.evaluationFunction(state)
            for action in legalActions:
                nextState = state.getNextState(agentIndex, action)
                if agentIndex == state.getNumAgents() - 1:
                    v = min(v, maxValue(nextState, 0, depth + 1, alpha, beta))
                else:
                    v = min(v, minValue(nextState, agentIndex + 1, depth, alpha, beta))
                if v < alpha:
                    return v
                beta = min(beta, v)
            return v

        legalActions = gameState.getLegalActions()
        bestAction = None
        v = float("-inf")
        alpha = float("-inf")
        beta = float("inf")
        for action in legalActions:
            nextState = gameState.getNextState(0, action)
            nextValue = minValue(nextState, 1, 0, alpha, beta)
            if nextValue > v:
                v = nextValue
                bestAction = action
            if v > beta:
                return bestAction
            alpha = max(alpha, v)
        return bestAction
        # End your code (Part 2)
```

Part3:

```python
        # Begin your code (Part 3)
        """
        maxValue function recursively calculates the maximum value of a given state by iterating over
        all legal actions of the pacman player and calling the expectValue function on the resulting next state.
        The expectValue function calculates the expected value of a given state by iterating over all legal
        actions of the current agent and recursively calling either maxValue or expectValue on the resulting next state.
        And returns the average of the resulting values, weighted by probability of each action.
        Finally, the agent selects the action with the highest expected score and returns.
        """
        def maxValue(state, depth):
            if state.isWin() or state.isLose() or depth == self.depth:
                return self.evaluationFunction(state)
            v = float('-inf')
            for action in state.getLegalActions(0):
                v = max(v, expectValue(state.getNextState(0, action), depth, 1))
            return v

        def expectValue(state, depth, agentIndex):
            if state.isWin() or state.isLose() or depth == self.depth:
                return self.evaluationFunction(state)
            v = 0
            legalActions = state.getLegalActions(agentIndex)
            p = 1.0 / len(legalActions)
            for action in legalActions:
                if agentIndex == state.getNumAgents() - 1:
                    v += p * maxValue(state.getNextState(agentIndex, action), depth + 1)
                else:
                    v += p * expectValue(state.getNextState(agentIndex, action), depth, agentIndex + 1)
            return v
        legalMoves = gameState.getLegalActions()
        scores = []
        for action in legalMoves:
            score = expectValue(gameState.getNextState(0, action), 0, 1)
            scores.append(score)

        bestScore = max(scores)
        bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
        chosenIndex = random.choice(bestIndices)
        return legalMoves[chosenIndex]
        # End your code (Part 3)
```

Part4:

```python
    # Begin your code (Part 4)
    """
    The function calculates a score for a pacman game state based on the distance and proximity of ghosts to pacman,
    the distance to the closest remaining food, and the number of remaining capsules.
    The function prioritizes getting closer to food while avoiding ghosts, while also
    taking into account the remaining capsules on the board.
    """
    pPos = currentGameState.getPacmanPosition()
    gDist = 0
    gProximity = 0
    for gState in currentGameState.getGhostPositions():
        dist = util.manhattanDistance(pPos, gState)
        gDist += dist
        if dist <= 1:
            gProximity += 1

    foodList = currentGameState.getFood().asList()
    minfoodList = min([util.manhattanDistance(pPos, foodPos) for foodPos in foodList], default=0)

    capsuleNum = len(currentGameState.getCapsules())
    return currentGameState.getScore() - (1 / (gDist+1)) - gProximity + (1 / (minfoodList+1)) - capsuleNum
    # End your code (Part 4)
```

Part II. Results & Analysis (5%):

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


           ALL HAIL GRANDPAC.
    LONG LIVE THE GHOSTBUSTING KING.

          ---      ----      ---
          |  \    /  + \    /  |
          | + \--/      \--/ + |
          |   +     +          |
          | +     +        +   |
      @@@@@@@@@@@@@@@@@@@@@@@@@@
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  \   @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   \ /  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    V   \   @@@@@@@@@@@@@@@@@@@@@@@@@@@@
         \ /  @@@@@@@@@@@@@@@@@@@@@@@@@@
          V     @@@@@@@@@@@@@@@@@@@@@@@@
                  @@@@@@@@@@@@@@@@@@@@@@
          /\      @@@@@@@@@@@@@@@@@@@@@@
         /  \  @@@@@@@@@@@@@@@@@@@@@@@@@
    /\  /    @@@@@@@@@@@@@@@@@@@@@@@@@@@
   /  \ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  /    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
      @@@@@@@@@@@@@@@@@@@@@@@@@@
          @@@@@@@@@@@@@@@@@@
```

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

Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1159
Pacman emerges victorious! Score: 1158
Pacman emerges victorious! Score: 1174
Pacman emerges victorious! Score: 1371
Pacman emerges victorious! Score: 849
Pacman emerges victorious! Score: 1170
Pacman emerges victorious! Score: 926
Pacman emerges victorious! Score: 1155
Pacman emerges victorious! Score: 1160
Average Score: 1149.5
Scores:        1373.0, 1159.0, 1158.0, 1174.0, 1371.0, 849.0, 1170.0, 926.0, 1155.0, 1160.0
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
***     1149.5 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 ###
```

During the evaluation of the pacman game state, the distance between pacman and ghosts, food, the proximity of ghosts around pacman, and the number of capsules are taken into consideration. Among these factors, the distance between pacman and food has the most significant impact on the score. This makes sense since the ultimate objective of the game is to gain more points, and getting closer to food is the key to achieving this objective. However, the other factors such as the proximity of ghosts and the number of capsules also influence the outcome since they are essential for survival and successful completion of the game.