Homework 3: Multi-Agent Search

110550168

Part I. Implementation (5%):

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

Part 1: Minimax Search (20%)

```
        # Begin your code (Part 1)
        def minimax(depth,agentID,state): # define a function to do recursive
            if state.isWin() | state.isLose(): # current state is already win lose
                return self.evaluationFunction(state) # return current evaluation value
            if depth == self.depth: # reach the required depth
                return self.evaluationFunction(state) # return current evaluation value
            action = state.getLegalActions(agentID) # get the current state action with agent ID
            if agentID == 0: # agent is 0 => then is pacman
                tmp_score = float("-INF") # set a max value with the min num
                for acts in action: # go through all action
                    next_state = state.getNextState(agentID,acts) # get the action's next state
                    tmp_score = max(minimax(depth,agentID + 1,next_state),tmp_score)
                    # if the minimax return value is larger than tmp_score, change tmp_score
                return tmp_score # return the max value in this action
            else: # agent is not 0 => then is ghost
                min_score = float("INF") # set a min value with the max num
                for acts in action: # go through all action
                    next_state = state.getNextState(agentID,acts) # get the action's next state
                    if agentID + 1 >= state.getNumAgents(): # all agents choose action
                        min_score = min(minimax(depth + 1,0,next_state),min_score)
                        # if the minimax return value is smaller than min_score, change min_score
                        # since all agents has moves, depth + 1
                    else:
                        min_score = min(minimax(depth,agentID + 1,next_state),min_score)
                        # if the minimax return value is smaller than min_score, change min_score
                return min_score # return the min value in this action
```

```

        best = float("-INF") # set a max value with the min num
        move = None # define a variable to store action
        if gameState.isWin() | gameState.isLose() | self.depth == 0:
            return None # if already win/lose or reach required depth, do nothing
        for act in gameState.getLegalActions(0): # go through all actions
            next = gameState.getNextState(0,act) # get the action's next state
            score = minimax(0,1,next) # find out it's minimax value
            if score > best: # choose the max one
                best = score # store its value
                move = act # store its action
        return move # return the action found
        # End your code (Part 1)
```

I define a function, so that I can do recursive. At first, I do the depth = 0, agentID = 0 outside, so that I can get the specific action, and then the recursive function can just return the value. At every time the agentID + 1 reaches the total number of the agents, depth needs to add 1.

Part 2: Alpha-Beta Pruning (25%)

```
186         # Begin your code (Part 2)
187         def alphabeta(depth,agentID,state,alpha,beta): # define a function to do recursive
188             if state.isWin() | state.isLose(): # current state is already win lose
189                 return self.evaluationFunction(state) # return current evaluation value
190             if depth == self.depth: # reach the required depth
191                 return self.evaluationFunction(state) # return current evaluation value
192             action = state.getLegalActions(agentID) # get the current state action with agent ID
193             if agentID == 0: # agent is 0 => then is pacman
194                 tmp_score = float("-INF") # set a max value with the min num
195                 for acts in action: # go through all action
196                     next_state = state.getNextState(agentID,acts) # get the action's next state
197                     tmp_score = max(alphabeta(depth,agentID + 1,next_state,alpha,beta),tmp_score)
198                     # if the minimax return value is larger than tmp_score, change tmp_score
199                     if tmp_score > beta:
200                         return tmp_score # return if current value is larger than beta
201                     alpha = max(tmp_score,alpha) # update alpha with the max value in this action
202                 return tmp_score # return the max value in this action
203             else: # agent is not 0 => then is ghost
204                 min_score = float("INF") # set a min value with the max num
205                 for acts in action: # go through all action
206                     next_state = state.getNextState(agentID,acts) # get the action's next state
207                     if agentID + 1 >= state.getNumAgents(): # all agents choose action
208                         min_score = min(alphabeta(depth + 1,0,next_state,alpha,beta),min_score)
209                         # if the minimax return value is smaller than min_score, change min_score
210                         # since all agents has moves, depth + 1
211                     else:
212                         min_score = min(alphabeta(depth,agentID + 1,next_state,alpha,beta),min_score)
213                         # if the minimax return value is smaller than min_score, change min_score
214                     if min_score < alpha:
215                         return min_score # return if current value is smaller than alpha
216                     beta = min(min_score,beta) # update beta with the min value in this action
217                 return min_score # return the min value in this action
```

```
218
219         best = float("-INF") # set a max value with the min num
220         alpha = float("-INF") # set alpha to the minimal, then later update its value
221         beta = float("INF") # set beta to the maximal, then later update its value
222         move = None # define a variable to store action
223         if gameState.isWin() | gameState.isLose() | self.depth == 0:
224             return None # if already win/lose or reach required depth, do nothing
225         for act in gameState.getLegalActions(0): # go through all actions
226             next = gameState.getNextState(0,act) # get the action's next state
227             score = alphabeta(0,1,next,alpha,beta) # find out it's minimax value
228             if score > best: # choose the max one
229                 best = score # store its value
230                 move = act # store its action
231             alpha = max(score,alpha) # in here, need to update alpha
232         return move # return the action found
233         # End your code (Part 2)
```

I define a function, so that I can do recursive. At first, I do the depth = 0, agentID = 0 outside, so that I can get the specific action, and then the recursive function can just return the value. Because of this reason, I need to update alpha at that part, which is line 231. Also, the parameters in the recursive function need to contain alpha and beta, when doing the algorithm, update alpha and beta, then pass down. At every time the agentID + 1 reaches the total number of the agents, depth needs to add 1.

Part 3: Expectimax Search (25%)

```python
        # Begin your code (Part 3)
        def expectimax(depth,agentID,state): # define a function to do recursive
            if state.isWin() | state.isLose(): # current state is already win lose
                return self.evaluationFunction(state) # return current evaluation value
            if depth == self.depth: # reach the required depth
                return self.evaluationFunction(state) # return current evaluation value
            action = state.getLegalActions(agentID) # get the current state action with agent ID
            if agentID == 0: # agent is 0 => then is pacman
                tmp_score = float("-INF") # set a max value with the min num
                for acts in action: # go through all action
                    next_state = state.getNextState(agentID,acts) # get the action's next state
                    tmp_score = max(expectimax(depth,agentID + 1,next_state),tmp_score)
                    # if the minimax return value is larger than tmp_score, change tmp_score
                return tmp_score # return the max value in this action
            else: # agent is not 0 => then is ghost
                val = 0. # variable to sum up all action value
                count = 0 # counting the numbers of the value
                for acts in action: # go through all action
                    count = count + 1 # the number of the value add 1
                    next_state = state.getNextState(agentID,acts) # get the action's next state
                    if agentID + 1 >= state.getNumAgents(): # all agents choose action
                        val += expectimax(depth + 1,0,next_state)
                        # sum up the action value
                        # since all agents has moves, depth + 1
                    else:
                        val += expectimax(depth,agentID + 1,next_state) # sum up the action value
                return val/count # return the expection

        best = float("-INF") # set a max value with the min num
        move = None # define a variable to store action
        if gameState.isWin() | gameState.isLose() | self.depth == 0:
            return None # if already win/lose or reach required depth, do nothing
        for act in gameState.getLegalActions(0): # go through all actions
            next = gameState.getNextState(0,act) # get the action's next state
            score = expectimax(0,1,next) # find out it's minimax value
            if score > best: # choose the max one
                best = score # store its value
                move = act # store its action
        return move # return the action found
        # End your code (Part 3)
```

I define a function, so that I can do recursive. At first, I do the depth = 0, agentID = 0 outside, so that I can get the specific action, and then the recursive function can just return the value. At every time the agentID + 1 reaches the total number of the agents, depth needs to add 1. The special part of this is that I don’t choose the min value for ghost, but the expectation, so I use a variable to store the sum, and then take average when I return value.

Part 4: Evaluation Function (20%)

```python
        # Begin your code (Part 4)
        pac_pos = currentGameState.getPacmanPosition() # get pacman's position
        food_list = currentGameState.getFood().asList() # get the food list
        score = currentGameState.getScore() # get current score
        ghost_state = currentGameState.getGhostStates() # get the ghosts' state

        remain_cap = len(currentGameState.getCapsules()) # calculate remain capsules
        remian_food = len(food_list) # calculate remain food
        score -= 15 * remian_food # remain more, score is less
        score -= 30 * remain_cap # remain more, score is less

        for food in food_list: # go through all food
            dis = manhattanDistance(pac_pos,food) # calculate the distance
            if dis < 3: # according the the distance to decide the weight
                score -= 1 * dis
            else:
                score -= 0.5 * dis

        for ghost in ghost_state: # go through all ghost
            dis = manhattanDistance(pac_pos,ghost.getPosition())
            # calculate the distance
            # since the ghost is in state type, need to use getPosition() to get the position
            if dis < 3: # according the the distance to decide the weight
                score -= 20 * dis
            else:
                score -= 10 * dis

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

I maintain the score by the distance between the pacman and the food/ghosts. Then, I give it a reasonable value to reach the goal.

Part II. Results & Analysis (5%):

Please screenshot the results.

```
Finished at 18:40:25

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

```
Pacman emerges victorious! Score: 1090
Pacman emerges victorious! Score: 1371
Pacman emerges victorious! Score: 1171
Pacman emerges victorious! Score: 1166
Pacman emerges victorious! Score: 1129
Pacman emerges victorious! Score: 1149
Pacman emerges victorious! Score: 1155
Pacman emerges victorious! Score: 1126
Pacman emerges victorious! Score: 1216
Pacman emerges victorious! Score: 943
Average Score: 1151.6
Scores:        1090.0, 1371.0, 1171.0, 1166.0, 1129.0, 1149.0, 1155.0, 1126.0, 1216.0, 943.0
Win Rate:      10/10 (1.00)
```