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. For example,**

Part1:

```python
        # Begin your code (Part 1)
        value,action=self.minimax(gameState,0,0)#call the minimax function
        print(action)
        return action
        raise NotImplementedError("To be implemented")


    def minimax(self,gameState,depth,agentIndex):
        if agentIndex>=gameState.getNumAgents():#if every agent in the game return the value and action, then we finish one round
            agentIndex=0#set agentIndex=0 means that it's time to player's turn
            depth+=1#depth represent of many round we going to do
        if gameState.isWin() or gameState.isLose() or (depth>=self.depth):#(case1: if player win or lose/ case2:if finish the expected number of round)
            return self.evaluationFunction(gameState), None#in this case, we evaluate the extent of nextState if it is a good choice
        if agentIndex==0:#is player's turn
            return self.max_agent(gameState,depth,agentIndex)#player should return the max value
        else:#is ghost turn
            return self.min_agent(gameState,depth,agentIndex)#ghost should return the min value

    def max_agent(self,gameState,depth,agentIndex):#for player
        bestValue=-float('inf')#to do the comparison, initial the value equal to -inf
        NextAction = None#variable use for record the best next action
        for action in gameState.getLegalActions(agentIndex):#travel all the possible legal actions
            NextState=gameState.getNextState(agentIndex,action)#get the nextstate correspond to the legal action
            value,_=self.minimax(NextState,depth,agentIndex+1)#we should get the low level of value first then we can do the comparison
            if value>bestValue:#to find the max value
                bestValue=value#record the max value
                NextAction=action#and record the next action
        return bestValue,NextAction

    def min_agent(self,gameState,depth,agentIndex):#for ghost
        bestValue=float('inf')#to do the comparison, initial the value equal to inf
        NextAction = None
        for action in gameState.getLegalActions(agentIndex):
            NextState=gameState.getNextState(agentIndex,action)
            value,_=self.minimax(NextState,depth,agentIndex+1)
            if value<bestValue:#to find the min value
                bestValue=value#record the min value
                NextAction=action#and return next action
        return bestValue,NextAction
        # End your code (Part 1)
```

Part2:

```python
        # Begin your code (Part 2)
        """
        the difference between part2 and part 1 is the addition variable alpha and beta
        the initial value of alpha equal to -inf
        the initial value of beta equal to inf
        """
        value,action=self.minimax(gameState,0,0,float('-inf'),float('inf'))#call minimax function
        print(action)
        return action
        raise NotImplementedError("To be implemented")


    def minimax(self,gameState,depth,agentIndex,alpha,beta):
        if agentIndex>=gameState.getNumAgents():
            agentIndex=0
            depth+=1
        if gameState.isWin() or gameState.isLose() or (depth>=self.depth):
            return self.evaluationFunction(gameState), None
        if agentIndex==0:
            return self.max_agent(gameState,depth,agentIndex,alpha,beta)#add two addition variable
        else:
            return self.min_agent(gameState,depth,agentIndex,alpha,beta)

    def max_agent(self,gameState,depth,agentIndex,alpha,beta):
        bestValue=-float('inf')
        NextAction = None
        for action in gameState.getLegalActions(agentIndex):
            NextState=gameState.getNextState(agentIndex,action)
            value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
            if value>bestValue:
                bestValue=value
                NextAction=action
            """
            add the jdgmental of alpha and beta
            """
            if bestValue>beta:
                break
            if bestValue>alpha:
                alpha=max(alpha,bestValue)
        return bestValue,NextAction
```

```python
    def min_agent(self,gameState,depth,agentIndex,alpha,beta):
        bestValue=float('inf')
        NextAction = None
        for action in gameState.getLegalActions(agentIndex):
            NextState=gameState.getNextState(agentIndex,action)
            value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
            if value<bestValue:
                bestValue=value
                NextAction=action
            if bestValue<alpha:
                break
            if bestValue<beta:
                beta=min(beta,bestValue)
        return bestValue,NextAction
        # End your code (Part 2)
```

Part3:

```python
        # Begin your code (Part 3)
        """
        change the min_agent function in part 3
        the algorithm should change the outcomes to average-case, not worst-case
        """
        value,action=self.minimax(gameState,0,0,float('-inf'),float('inf'))
        print(action)
        return action
        raise NotImplementedError("To be implemented")


    def minimax(self,gameState,depth,agentIndex,alpha,beta):
        if agentIndex>=gameState.getNumAgents():
            agentIndex=0
            depth+=1
        if gameState.isWin() or gameState.isLose() or (depth>=self.depth):
            return self.evaluationFunction(gameState), None
        if agentIndex==0:
            return self.max_agent(gameState,depth,agentIndex,alpha,beta)
        else:
            return self.min_agent(gameState,depth,agentIndex,alpha,beta)

    def max_agent(self,gameState,depth,agentIndex,alpha,beta):
        bestValue=-float('inf')
        NextAction = None
        for action in gameState.getLegalActions(agentIndex):
            NextState=gameState.getNextState(agentIndex,action)
            value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
            if value>bestValue:
                bestValue=value
                NextAction=action
            if bestValue>beta:
                break
            if bestValue>alpha:
                alpha=max(alpha,bestValue)
        return bestValue,NextAction
```

```python
    def min_agent(self,gameState,depth,agentIndex,alpha,beta):
        total_value=0
        actionNum=0
        NextAction = None
        for action in gameState.getLegalActions(agentIndex):
            NextState=gameState.getNextState(agentIndex,action)
            value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
            total_value+=value
            actionNum+=1
        bestValue=total_value/actionNum#calculate the average-case value
        return bestValue,NextAction
        # End your code (Part 3)
```

Part4:

```python
# Begin your code (Part 4)
GhostStates = currentGameState.getGhostStates() #all the ghost states
Pacman_Pos = currentGameState.getPacmanPosition()
food_list = (currentGameState.getFood()).asList() #get all the food as list.
capsule_list = currentGameState.getCapsules() #get all the capsules.
no_food = len(food_list) #check that if there still have food or not
no_capsule = len(capsule_list) #chenck if there still have capsule or not

state_score=0 #variable for record the final score
if currentGameState.getNumAgents() > 1:#>1 represent that there are ghost exist
    ghost_dis = min( [manhattanDistance(Pacman_Pos, ghost.getPosition()) for ghost in GhostStates])# search the nearest ghost distance
    if (ghost_dis <= 1):
        return -10000 #if nearest ghost distance is small than 1, this is the worst case so return -10000
    state_score -= 1.0/ghost_dis #minus the score with 1/(distance between player and ghost)
#Feature 3 food positions
current_food = Pacman_Pos
for food in food_list: #calculate all the food distance and update the score
    closestFood = min(food_list, key=lambda x: manhattanDistance(x, current_food))
    state_score += 1.0/(manhattanDistance(current_food, closestFood)) #plus the score with 1/(the distance between player and food)
    current_food = closestFood #then change the pacman position to the previous eaten food
    food_list.remove(closestFood) # remove the food already been eaten
#Feature 4 capsule positions, same idea as food position
current_capsule = Pacman_Pos
for capsule in capsule_list:
    closest_capsule = min(capsule_list, key=lambda x: manhattanDistance(x, current_capsule))
    state_score += 1.0/(manhattanDistance(current_capsule, closest_capsule))
    current_capsule = closest_capsule
    capsule_list.remove(closest_capsule)
#Feature 4 Score of the game
state_score += 8*(currentGameState.getScore())

#Feature 5: remaining food and capsule
state_score -= 6*(no_food + no_capsule)

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

Part II. Results & Analysis (5%):

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

```
==================
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     @@@@@@@@@@@@@@@@@@@@@@@@
                            @@@@@@@@@@@@@@@@@@@@@@
                    /\      @@@@@@@@@@@@@@@@@@@@@@
                   /  \  @@@@@@@@@@@@@@@@@@@@@@@@@
              /\  /    @@@@@@@@@@@@@@@@@@@@@@@@@@@
             /  \ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
            /    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
            @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
            @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
              @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                @@@@@@@@@@@@@@@@@@@@@@@@@@
                    @@@@@@@@@@@@@@@@@@
```