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,**

Part 1

```
# Begin your code (Part 1)
"""
First,call the minimax function.
minimax:If every agents have returned value the value and action,this roun should be finished and set
        agentIndex as zero which means it is time to get into player's turn,then depth pluse 1 to
        represent how many round we are going to do.
        Either player is win/lose or the expected number of round is finished,we sould evaluate the
        extent of nextState whether it is a good choice.If it is ghost's turn,return min_agent.else,return max_agent.
max_agent:This is for player.First,the value named "best" is initialized as -inf and the variable used to
          record the best next action named "NextAction" is none.Use for loop to travel all the possible actions
          which are legal and know the next state corresponds to the legal action.Also,we get the low level of value
          to do the following comparison to find max value and turn it to "best".
min_agent:This is for ghosts.First,the value named "best" is initialized as inf and the variable used to record the
          best next action named "NextAction" is none.Use for loop to travel all the possible actions which are legal
          and know the next state corresponds to the legal action.Also,we get the low level of value to do the following
          comparison to find min value and turn it to "best".
"""
```

```
        value,action=self.minimax(gameState,0,0)
        # print(action)
        return action
        raise NotImplementedError("To be implemented")


    def minimax(self,gameState,depth,agentIndex):
        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.min_agent(gameState,depth,agentIndex)
        else:
            return self.max_agent(gameState,depth,agentIndex)

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

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

Part 2

```python
# Begin your code (Part 2)
"""
First,call the minimax function.
minimax:If every agents have returned value the value and action,this roun should be finished and set agentIndex as zero
        which means it is time to get into player's turn,then depth pluse 1 to represent how many round we are going to do.
        Either player is win/lose or the expected number of round is finished,we sould evaluate the extent of nextState
        whether it is a good choice.If it is ghost's turn,return min_agent.else,return max_agent.
max_agent:This is for player.First,the value named "best" is initialized as -inf and the variable used to record the best
          next action named "NextAction" is none.Use for loop to travel all the possible actions which are legal and know the
          next state corresponds to the legal action.Also,we get the low level of value to do the following comparison to find
          max value and turn it to "best".If "best" is greater than beta,just break.If best is larger than alpha,alpha=best.
min_agent:This is for ghosts.First,the value named "best" is initialized as inf and the variable used to record the best
          next action named "NextAction" is none.Use for loop to travel all the possible actions which are legal and know the
          next state corresponds to the legal action.Also,we get the low level of value to do the following comparison to find
          min value and turn it to "best".
The difference between part 1 and part 2 is that we need to note two variables:alpha and beta.
The initial value of alpha is -inf.
The initial value of beta is inf.
"""
```

```python
        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.min_agent(gameState,depth,agentIndex,alpha,beta)
        else:
            return self.max_agent(gameState,depth,agentIndex,alpha,beta)

    def max_agent(self,gameState,depth,agentIndex,alpha,beta):
        best=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>best:
                best=value
                NextAction=action

            if best>beta:
                break
            if best>alpha:
                alpha=max(alpha,best)
        return best,NextAction
```

```python
    def min_agent(self,gameState,depth,agentIndex,alpha,beta):
        best=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<best:
                best=value
                NextAction=action
            if best<alpha:
                break
            if best<beta:
                beta=min(beta,best)
        return best,NextAction
        # End your code (Part 2)
```

Part 3

```python
# Begin your code (Part 3)
"""
First,call the minimax function.
The initial value of alpha is -inf.
The initial value of beta is inf.
minimax:If every agents have returned value the value and action,this roun should be finished and set agentIndex as zero
        which means it is time to get into player's turn,then depth pluse 1 to represent how many round we are going to do.
        Either player is win/lose or the expected number of round is finished,we sould evaluate the extent of nextState
        whether it is a good choice.If it is ghost's turn,return min_agent.else,return max_agent.
max_agent:This is for player.First,the value named "best" is initialized as -inf and the variable used to record the best
          next action named "NextAction" is none.Use for loop to travel all the possible actions which are legal and know the
          next state corresponds to the legal action.Also,we get the low level of value to do the following comparison to find
          max value and turn it to "best".If "best" is greater than beta,just break.If best is larger than alpha,alpha=best.
min_agent:This is for ghosts.Use total_Num/the number of action to replace worst-case into average case.
"""
```

```python
        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.min_agent(gameState,depth,agentIndex,alpha,beta)
        else:
            return self.max_agent(gameState,depth,agentIndex,alpha,beta)

    def max_agent(self,gameState,depth,agentIndex,alpha,beta):
        best=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>best:
                best=value
                NextAction=action
            if best>beta:
                break
            if best>alpha:
                alpha=max(alpha,best)
        return best,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
        best=total_value/actionNum
        return best,NextAction
        # End your code (Part 3)
```

Part 4

```python
# Begin your code (Part 4)
"""
Use currentGameState to get ghosts'states,foodand capsule as list.Travel the list of food to calculate the distance between food and player and refresh the list.Travel the list of capsule to calculate the distance between capsules and player and refresh the list.
Use currentGameState to update the score of the game and remain food and capsules.
"""
```

```python
    GhostStates = currentGameState.getGhostStates()
    Pacman_Pos = currentGameState.getPacmanPosition()
    food_list = (currentGameState.getFood()).asList()
    capsule_list = currentGameState.getCapsules()
    no_food = len(food_list)
    no_capsule = len(capsule_list)

    state_score=0
    if currentGameState.getNumAgents() > 1:
        ghost_dis = min( [manhattanDistance(Pacman_Pos, ghost.getPosition()) for ghost in GhostStates])
        if (ghost_dis <= 1):
            return -10000
        state_score -= 1.0/ghost_dis
    current_food = Pacman_Pos
    for food in food_list:
        closestFood = min(food_list, key=lambda x: manhattanDistance(x, current_food))
        state_score += 1.0/(manhattanDistance(current_food, closestFood))
        current_food = closestFood
        food_list.remove(closestFood)
    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)
    state_score += 10*(currentGameState.getScore())
    state_score -= 8*(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**.

```
***          < 1:  fail
***         >= 1:  1 points
***         >= 4:  2 points
***         >= 7:  3 points
***         >= 10:  4 points

### Question part4: 10/10 ###


Finished at 19:37:16

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