Homework 3: Multi-Agent Search

109550116 楊傑宇

Part I. Implementation (5%):

```
        # Begin your code (Part 1)
        '''
        I separate my code into three parts : main, find min, find max.

        For main part, get every possible pacman move's min value and put them in temp=[]
        then choose the best(max value) to move. I use random to avoid getting stuck

        For min function, it will return the min value between different ghost moves.
        Because there is more than one ghost, min value is related to another's min value.
        Therefore, we need to use recursion to get another ghost's min value
        ex: temp.append(self.Min_Value(gameState1,i+1, d)
        However, if the ghost is the last one it have to call max function because
        it's value it related to pacman's max value,and when doing this step means depth+1 so d+1.
        ex: temp.append(self.Max_Value(gameState1, d+1)
        PS: if pacman touch ghost, there is no legalaction for ghost, so we need to
        return the score right now.

        For max function, it will return max value  between different pacman moves.
        it's value is related to ghost's min value. Therefore, we need to call min function
        to get min value.
        ex: temp.append(self.Min_Value(gameState1,1,d)
        and if d=self.depth means it arrive leaf, just return current score.

        '''

        legalMoves = gameState.getLegalActions()
        temp=[]
```

```
        for action in legalMoves:
            childGameState = gameState.getNextState(0,action)
            temp.append(self.Min_Value(childGameState,1,0))
        maxscore=max(temp)
        bestIndices = [index for index in range(len(temp)) if temp[index] == maxscore]
        chosenIndex = random.choice(bestIndices)
        return legalMoves[chosenIndex]


    def Max_Value (self, gameState, d):

        if gameState.isLose() or gameState.isWin() or d == self.depth:
            return self.evaluationFunction(gameState)

        temp=[]
        pacmoves=gameState.getLegalActions(0)
        for action in pacmoves :
            gameState1=gameState.getNextState(0,action)
            temp.append(self.Min_Value(gameState1,1, d))
        return max(temp)


    def Min_Value (self, gameState, i, d):

        if gameState.isLose() or gameState.isWin():
            return self.evaluationFunction(gameState)

```

```
        if (i < gameState.getNumAgents() - 1):
            temp=[]
            ghostmoves=gameState.getLegalActions(i)
            for action in ghostmoves :
                gameState1=gameState.getNextState(i,action)
                temp.append(self.Min_Value(gameState1,i+1, d))
            return min(temp)

        else:
            temp=[]
            ghostmoves=gameState.getLegalActions(i)
            for action in ghostmoves :
                gameState1=gameState.getNextState(i,action)
                temp.append(self.Max_Value(gameState1, d+1))
            return min(temp)

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

```
        # Begin your code (Part 2)
        '''
        Actually, part2 structure is similiar to part1. I just add puning condition.
        For main part, it will bring the best(max) value as alpha to function.

        For min function, if there is one move that make value smaller than alpha,
        we will not consider other moves, just return. Furthermore, min function will bring
        the min value as beta to let max function puning

        Fro max function, if there is one move that make value bigger than beta,
        we will not consider other moves, just return.

        Finally, the main will record the move that has biggest temp and return the move.
        '''

        a = float('-inf')
        b = float('inf')
        moves=None
        legalMoves = gameState.getLegalActions()
        for action in legalMoves:
            childGameState = gameState.getNextState(0,action)
            temp=self.Min_Value(childGameState,1,0,a,b)
            if(a<temp):
                a=temp
                moves=action
        return moves

    def Max_Value (self, gameState, d, alpha, beta):
        if gameState.isLose() or gameState.isWin() or d == self.depth:
            return self.evaluationFunction(gameState)

        maxi=float('-inf')
        pacmoves=gameState.getLegalActions(0)
        for action in pacmoves :
            gameState1=gameState.getNextState(0,action)
            temp=self.Min_Value(gameState1,1,d,alpha, beta)
```

```
            maxi=max(maxi,temp)
            if (maxi>beta):
                return maxi
            alpha = max(alpha,maxi)
        return maxi


    def Min_Value (self, gameState, i, d, alpha, beta):

        if gameState.isLose() or gameState.isWin():
            return self.evaluationFunction(gameState)

        minii=float('inf')
        if (i < gameState.getNumAgents() - 1):
```

```
            ghostmoves=gameState.getLegalActions(i)
            for action in ghostmoves :
                gameState1=gameState.getNextState(i,action)
                temp=self.Min_Value(gameState1,i+1,d,alpha, beta)
                minii=min(temp,minii)
                if minii < alpha:
                    return minii
                beta = min(beta,minii)

        else:
            ghostmoves=gameState.getLegalActions(i)
            for action in ghostmoves :
                gameState1=gameState.getNextState(i,action)
                temp=self.Max_Value(gameState1,d+1,alpha, beta)
                minii=min(temp,minii)
                if minii < alpha:
                    return minii
                beta = min(beta,minii)

        return minii

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

```
        # Begin your code (Part 3)
        '''
        Actually, part3 is similiar to part1, even easilier.

        For main part, no change

        For min function, we don't need to return min value anymore.
        instead, we return expected value.
        ex: return (exp/len(ghostmoves))

        For max function, no change
        '''
        legalMoves = gameState.getLegalActions()
        temp=[]
        for action in legalMoves:
            childGameState = gameState.getNextState(0,action)
            temp.append(self.Min_Value(childGameState,1,0))
        maxscore=max(temp)
        bestIndices = [index for index in range(len(temp)) if temp[index] == maxscore]
        chosenIndex = random.choice(bestIndices)
        return legalMoves[chosenIndex]

    def Max_Value (self, gameState, d):

        if gameState.isLose() or gameState.isWin() or d == self.depth:
            return self.evaluationFunction(gameState)

        temp=[]
        pacmoves=gameState.getLegalActions(0)
        for action in pacmoves :
            gameState1=gameState.getNextState(0,action)
            temp.append(self.Min_Value(gameState1,1, d))
        return max(temp)
```

```
344     def Min_Value (self, gameState, i, d):
345
346         if gameState.isLose() or gameState.isWin():
347             return self.evaluationFunction(gameState)
348
349         if (i < gameState.getNumAgents() - 1):
350             exp=0
351             ghostmoves=gameState.getLegalActions(i)
352             for action in ghostmoves :
353                 gameState1=gameState.getNextState(i,action)
354                 exp=exp+self.Min_Value(gameState1,i+1,d)
355             return (exp/len(ghostmoves))
356
357         else:
358             exp=0
359             ghostmoves=gameState.getLegalActions(i)
360             for action in ghostmoves :
361                 gameState1=gameState.getNextState(i,action)
362                 exp=exp+self.Max_Value(gameState1, d+1)
363             return (exp/len(ghostmoves))
364
365         #raise NotImplementedError("To be implemented")
366         # End your code (Part 3)
```

```
375     # Begin your code (Part 4)
376     '''
377     My code is easy, it can be seen as three parts.
378
379     First, cfd is closetest food. if the move can eat food cfd will=0
380     and score will not minus, otherwise it will move forward to the closest food
381     somewhat like greedy.
382
383     Second, the same method to capsule
384
385     Third, this is in order to let pacman to be close to ghost, when ghost is scared.
386     And the distance will keep in a value, which make sure pacman can chase ghost before scare ends
387
388     the coefficient of every method are setted after several tries
389     '''
390     pacman    = current.getPacmanPosition()
391     ghosts    = current.getGhostStates()
392     score     = current.getScore()
393     food_list = (current.getFood()).asList()
394     capsule_list = current.getCapsules()
395
396     cfd=-1
397     for food in food_list:
398         if (cfd<0 or manhattanDistance(pacman,food)<cfd):
399             cfd=manhattanDistance(pacman,food)
400     if(cfd<0):
401         score+=0
402     else:
403         score+=-0.25*cfd
404
405     ccd=-1
406     for capsule in capsule_list:
407         if (ccd<0 or manhattanDistance(pacman,capsule)<ccd):
408             ccd=manhattanDistance(pacman,capsule)
```

```
409
410         if(ccd<0):
411             score+=0
412         else:
413             score+=-1*ccd
414
415
416         for ghost in ghosts:
417             if ghost.scaredTimer > 0 and  manhattanDistance(ghost.getPosition(), pacman)<= ghost.scaredTimer:
418                 score += 200
419             elif pacman == ghost.getPosition():
420                 score -= 500
421
422         return score
423
424         #raise NotImplementedError("To be implemented")
425         # End your code (Part 4)
426
427     # Abbreviation
428     better = betterEvaluationFunction
```

Part II. Results & Analysis (5%):

```
AI_HW3 — -zsh — 99×52
***          < 1:  fail
***         >= 1:  1 points
***         >= 4:  2 points
***         >= 7:  3 points
***         >= 10:  4 points

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


Finished at 21:22:59

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     @@@@@@@@@@@@@@@@@@@@@@@@
                               @@@@@@@@@@@@@@@@@@@@@@@
                        /\     @@@@@@@@@@@@@@@@@@@@@@@
                       /  \  @@@@@@@@@@@@@@@@@@@@@@@@
                  /\  /    @@@@@@@@@@@@@@@@@@@@@@@@@
                 /  \ @@@@@@@@@@@@@@@@@@@@@@@@@@@@@
               /    @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
               @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
               @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@
                   @@@@@@@@@@@@@@@@@@@@@@@@@@
                       @@@@@@@@@@@@@@@@@@@

(base) jerrys_macbook_pro@jerrys-MacBook-Pro AI_HW3 %
```

Observation:

At the beginning, I supposed that if I decrease the coefficient of ccd, which can let pacman eat capsule preferentially. However, it will not eat the first capsule because ccd will become so big after eating the first capsule. Therefore, every number need to be considered carefully. With every factor affecting modestly, pacman can run in a high-score strategy.