Homework 3: Multi-Agent Search

By 109550093 黃得誠

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:

```
109  class MinimaxAgent(MultiAgentSearchAgent):
110      '''
111      Your minimax agent (Part 1)
112      '''
113      def getAction(self, gameState):
114          """...
136          # Begin your code (Part 1)
137
138          def countmax(depth,gameState,index_agent):...
214
215          # Take the best move from function countmax(recursively)
216          bestScore,bestmove = countmax(self.depth,gameState,0)
217
218          return bestmove
219          # End your code (Part 1)
```

details of the countmax function

```
141      def countmax(depth,gameState,index_agent):
142          '''
143          depth means the depth this state in.
144          gameState means the state that is going to be execute this time.
145          index_agent means the index of the agent,
146          pacman is 0, ghosts are 1~num_agents-1
147          countmax will return the bestscore and bestmove
148          bestscore is either min/max score in ghost/pacman.
149          bestmove is only matters when it comes to pacman(index_agent=0)
150          '''
151          num_agents = gameState.getNumAgents() #number of agents
152          legalMoves = gameState.getLegalActions(index_agent)#legal moves of the executing state
153          if len(legalMoves)==0:#done with game at the state(either win or lose)
154              return self.evaluationFunction(gameState),"0" #return the points
155
```

```
            #In each condition, scores means all the scores of childstates
            #Take all posibile childstates from legalmove and count the min/max of it.
            if depth ==1 and index_agent == num_agents-1:
                '''
                depth =1 and index = num_agents-1 means it is the terminal state.
                It is also the leaf process of the recursion function call.
                take scores from self.evaluationFunction easily, since it's the terminal state.
                return bestScore, which is the minimum score of all posible scores.
                '''
                scores = []
                for action in legalMoves:
                    GameState = gameState.getNextState(index_agent,action)
                    scores.append(self.evaluationFunction(GameState))
                bestScore=min(scores)
                bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
                chosenIndex = random.choice(bestIndices)
                return bestScore,legalMoves[chosenIndex]
```

```
            elif index_agent==0:
                '''
                index_agent=0 means it is the time when pacman(max_player)'s time.
                countmax(depth,Gamestate,1) is used because I start from the ghost with index 1.
                return bestScore(max score of scores)
                and
                bestMove(which matters because the original get_action function relies on it)
                '''
                scores = []
                for action in legalMoves:
                    GameState = gameState.getNextState(0,action)
                    scores.append(countmax(depth,GameState,1))
                bestScore = max(scores)[0]
                bestIndices = [index for index in range(len(scores)) if scores[index][0] == bestScore]
                chosenIndex = random.choice(bestIndices)
                return bestScore,legalMoves[chosenIndex]
            elif index_agent == num_agents-1:
                '''
                index_agent=num_agents-1 but depth not equal to 1 means the childstate is pacman in the next depth.
                Therefore, countmax(depth-1,Gamestate,0) is used instead of countmax(depth,Gamestate,index_agent+1)
                return bestScore, which is the minimum of scores.
                '''
                scores=[]
                for action in legalMoves:
                    GameState = gameState.getNextState(index_agent,action)
                    scores.append(countmax(depth-1,GameState,0))
                bestScore = min(scores)[0]
                bestIndices = [index for index in range(len(scores)) if scores[index][0] == bestScore]
                chosenIndex = random.choice(bestIndices)
                return bestScore,legalMoves[chosenIndex]
```

Part 2.
The “IF,ELSE” condition means the same as Part1, so I will not explain it again.
In addition, I will explain the implementation of alpha-beta pruning.

```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
    Your minimax agent with alpha-beta pruning (Part 2)
    """


    def getAction(self, gameState):
        """
        Returns the minimax action using self.depth and self.evaluationFunction
        """
        # Begin your code (Part 2)
        def countmax(depth,gameState,index_agent,alpha_beta):···
        #alpha_beta is send as a parameter, which is new from minimax
        bestScore,bestmove = countmax(self.depth,gameState,0,[-INF,INF])
        return bestmove
        # End your code (Part 2)
```

details of the countmax function.

```python
    def countmax(depth,gameState,index_agent,alpha_beta):
        #alpha_beta[0] means alpha
        #alpha_beta[1] means beta
        num_agents = gameState.getNumAgents()
        legalMoves = gameState.getLegalActions(index_agent)
        if len(legalMoves)==0:
            return self.evaluationFunction(gameState),"0"
        if depth ==1 and index_agent == num_agents-1:
            '''
            this is a min_player, so update beta and prune if bestScore(v) is less than alpha
            the updated alpha_beta list will be changed in the function that call this function,
            because list is mutable.
            '''
            scores = []
            bestScore = INF
            for action in legalMoves:
                GameState = gameState.getNextState(index_agent,action)
                thisScore = self.evaluationFunction(GameState)
                scores.append(thisScore)
                bestScore = min(bestScore,thisScore)
                if bestScore<alpha_beta[0]:
                    return bestScore, action
                alpha_beta[1] = min(alpha_beta[1],bestScore)
            bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
            chosenIndex = bestIndices[-1]
            return bestScore,legalMoves[chosenIndex]
```

```
        elif index_agent==0:
            '''
            this is a max_player, so update alpha and prune if bestScore is larger than beta.
            the updated alpha_beta list will be changed in the function that call this function,
            because list is mutable.
            alpha_beta1 is used in order not to chanage the beta's value.
            (since this is a max_player, value of beta shouldn't be changed)
            '''
            scores = []
            bestScore = -INF

            for action in legalMoves:
                GameState = gameState.getNextState(0,action)
                alpha_beta1 = [alpha_beta[0],alpha_beta[1]]
                thisScore = countmax(depth,GameState,1,alpha_beta1)
                scores.append(thisScore)
                bestScore = max(bestScore,thisScore[0])
                if bestScore>alpha_beta[1]:
                    return bestScore, action
                alpha_beta1[0] = max(alpha_beta1[0],bestScore)
                alpha_beta[0] = alpha_beta1[0]
            bestIndices = [index for index in range(len(scores)) if scores[index][0] == bestScore]
            chosenIndex = bestIndices[-1]
            return bestScore,legalMoves[chosenIndex]
    elif index_agent == num_agents-1:
        '''
        this is a min_player, so update beta and prune if bestScore(v) is less than alpha
        the updated alpha_beta list will be changed in the function that call this function,
        because list is mutable.
        '''
        scores=[]
        bestScore = INF
        for action in legalMoves:
            alpha_beta1 = [alpha_beta[0],alpha_beta[1]]
            GameState = gameState.getNextState(index_agent,action)

            thisScore = countmax(depth-1,GameState,0,alpha_beta1)
            scores.append(thisScore)
            bestScore = min(bestScore,thisScore[0])
            if bestScore<alpha_beta[0]:
                return bestScore, action
            alpha_beta1[1] = min(alpha_beta1[1],bestScore)
            alpha_beta[1] = alpha_beta1[1]

        bestIndices = [index for index in range(len(scores)) if scores[index][0] == bestScore]
        chosenIndex = bestIndices[-1]

        return bestScore,legalMoves[chosenIndex]
    else:
        '''
        this is a min_player, so update beta and prune if bestScore(v) is less than alpha
        the updated alpha_beta list will be changed in the function that call this function,
        because list is mutable.
        '''
        scores=[]
        bestScore = INF
        for action in legalMoves:
            alpha_beta1 = [alpha_beta[0],alpha_beta[1]]
            GameState = gameState.getNextState(index_agent,action)

            thisScore = countmax(depth,GameState,index_agent+1,alpha_beta1)
            scores.append(thisScore)
            bestScore = min(bestScore,thisScore[0])
            if bestScore<alpha_beta[0]:
                return bestScore, action
            alpha_beta1[1] = min(alpha_beta1[1],bestScore)
            alpha_beta[1] = alpha_beta1[1]

        bestIndices = [index for index in range(len(scores)) if scores[index][0] == bestScore]
        chosenIndex = bestIndices[-1]

        return bestScore,legalMoves[chosenIndex]
```

Part3.
The “IF,ELSE” condition means the same as Part1, so I will not explain it again.
In addition, I will explain the implementation of ExpectimaxAgent.

```python
class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent (Part 3)
    """



    def getAction(self, gameState):
        """
        Returns the expectimax action using self.depth and self.evaluationFunction

        All ghosts should be modeled as choosing uniformly at random from their
        legal moves.
        """
        # Begin your code (Part 3)
        def countmax(depth,gameState,index_agent): ...

        bestScore,bestmove = countmax(self.depth,gameState,0)
        return bestmove
        # End your code (Part 3)
```

Details of the countmax function.

```python
        def countmax(depth,gameState,index_agent):
            num_agents = gameState.getNumAgents()
            legalMoves = gameState.getLegalActions(index_agent)
            if len(legalMoves)==0:
                return self.evaluationFunction(gameState),"0"
            if depth ==1 and index_agent == num_agents-1:
                scores = []
                for action in legalMoves:
                    GameState = gameState.getNextState(index_agent,action)
                    scores.append(self.evaluationFunction(GameState))

                total = 0
                for value in scores:
                    total+=value
                bestScore = total/len(scores)
                '''
                Instead of finding the minimum of the scores,
                find the average score as the best score
                bestmove is not importent here, so just return legalMoves[0] it's ok.
                '''
                return bestScore,legalMoves[0]
            elif index_agent==0:
                scores = []
                for action in legalMoves:
                    GameState = gameState.getNextState(0,action)
                    scores.append(countmax(depth,GameState,1))
                bestScore = max(scores)[0]
                bestIndices = [index for index in range(len(scores)) if scores[index][0] == bestScore]
                chosenIndex = random.choice(bestIndices)
                '''
                the max_player, which is the pacman, remains the same as minimaxAgent
                '''
                return bestScore,legalMoves[chosenIndex]
```

```
385             elif index_agent == num_agents-1:
386                 scores=[]
387                 for action in legalMoves:
388                     GameState = gameState.getNextState(index_agent,action)
389                     scores.append(countmax(depth-1,GameState,0))
390                 total = 0
391                 for value in scores:
392                     total+=value[0]
393                 bestScore = total/len(scores)
394                 '''
395                 Instead of finding the minimum of the scores,
396                 find the average score as the best score
397                 bestmove is not importent here, so just return legalMoves[0] it's ok.
398                 '''
399                 return bestScore,legalMoves[0]
400             else:
401 
402                 scores=[]
403                 for action in legalMoves:
404                     GameState = gameState.getNextState(index_agent,action)
405                     scores.append(countmax(depth,GameState,index_agent+1))
406                 total=0
407                 for value in scores:
408                     total+=value[0]
409                 bestScore = total/len(scores)
410                 '''
411                 Instead of finding the minimum of the scores,
412                 find the average score as the best score
413                 bestmove is not importent here, so just return legalMoves[0] it's ok.
414                 '''
415                 return bestScore,legalMoves[0]
```

Part4.

```
422 def betterEvaluationFunction(currentGameState):
423     """
424     Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable
425     evaluation function (Part 4).
426     """
427     # Begin your code (Part 4)
428     '''
429     take the score, position, food, Ghoststates, from currentGameState
430     using the function used in the reflexagent.
431     '''
432     nowScore = currentGameState.getScore()
433     Pos = currentGameState.getPacmanPosition()
434     Food = currentGameState.getFood()
435     GhostStates = currentGameState.getGhostStates()
436     SacredTimes = [ghostState.scaredTimer for ghostState in GhostStates]
437 
438     #count the minGhostDistance using manhattanDistance,
439     #manhattanDistance is useful since pacman only go top,left,right,or down.
440     minGhostDistance = min([manhattanDistance(Pos, state.getPosition()) for state in GhostStates])
441     #count nearestfood distances
442     FoodsDistances = [manhattanDistance(Pos, food) for food in Food.asList()]
443     nearestFoodDistance = 0 if not FoodsDistances else min(FoodsDistances)
```

```
    '''
    The evaluationFunction I came up myself.
    1.If there is a ghost that is sacred and near pacman, pacman will chase it
    because I give this situation high points.
    2.If there is a ghost near pacman that is not sacred, give 0 points.
    3.no ghost near pacman and nearestFoodDistance<1: 10points
    4.else 5 points.
    '''
    for i in range(len(GhostStates)):
        if SacredTimes[i]>0 and manhattanDistance(Pos, GhostStates[i].getPosition())<SacredTimes[i]:
            return 300+nowScore
        elif SacredTimes[i]<0 and  manhattanDistance(Pos, GhostStates[i].getPosition())<2:
            return 0+nowScore
    if minGhostDistance<2:
        return 0+nowScore
    elif nearestFoodDistance<1:
        return 10+nowScore
    else:
        return 5+nowScore

    # 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**.
  Detail result of Part4 and the provisional grades.

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

Pacman emerges victorious! Score: 892
Pacman emerges victorious! Score: 1211
Pacman emerges victorious! Score: 1015
Pacman emerges victorious! Score: 937
Pacman emerges victorious! Score: 1031
Pacman emerges victorious! Score: 1124
Pacman emerges victorious! Score: 1102
Pacman emerges victorious! Score: 900
Pacman emerges victorious! Score: 1100
Pacman emerges victorious! Score: 963
Average Score: 1027.5
Scores:        892.0, 1211.0, 1015.0, 937.0, 1031.0, 1124.0, 1102.0, 900.0, 1100.0, 963.0
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
***     1027.5 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 ###


Finished at 19:14:29

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

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

Surprised by the last photo, when I finished this project at the first time.

Analysis and problems I met:

1. A problem I met at first is that although I finished the minimax with depth=1, I can’t push the progress to depth>1. I tried ways in order not to use recursive, but I failed finally. After trial and errors, I wrote a new recursive function which I named countmax. Finally, I finished this project.
2. I didn’t struggle at my own evaluation for a long time, since I read the code of reflexagent, and I found that there are some useful information such as nowposition, food, manhattandistance.
3. My evaluation looks good to me, hope it can pass the hidden testcases.