Multi-Agent Search report

071125 葉長瀚

Part I. Implementation:

Minimax:

```python
define a minimax funciton first to get the value recursively. If the agent is Pacman, pick the
maximun value; If the agent is Ghosts, pick the minimun value. Go through all possible actions
for every agent in each step.
'''
def minimax(gameStatee, agentIndex, depthh):
    if gameStatee.isWin() or gameStatee.isLose() or depthh==self.depth:
        return self.evaluationFunction(gameStatee)
    elif not agentIndex:
        v=float('-inf')
        for x in gameStatee.getLegalActions(0):
            v=max(v,minimax(gameStatee.getNextState(0,x),1,depthh))
        return v
    else:
        v=float('inf')
        nextIndex= agentIndex+1
        if nextIndex == gameStatee.getNumAgents():
            nextIndex=0
            depthh+=1
        for x in gameStatee.getLegalActions(agentIndex):
            v=min(v, minimax(gameStatee.getNextState(agentIndex,x),nextIndex, depthh))
        return v
```

```python
'''
Begin with the top pacman, return the action result in maximun value.
'''
maxV= float('-inf')
maxAct=0
for x in gameState.getLegalActions(0):
    temp=minimax(gameState.getNextState(0,x),1,0)
    if maxV<temp:
        maxV=temp
        maxAct=x
return maxAct
```

Alphabeta pruning:

```python
        Define the alphabeta function first too. In this function there's 2 addition arguments alpha and beta.
        In picking maximun part, if the value is larger than beta, which means this value wont be chosen by the
        upper minimun part, so we can prun it directly; Same in picking minimum part, if the value is smaller than
        alpha, we can prun it directly.
        '''
        def alphabeta(state, agentIndex,d, a, b):
            if state.isWin() or state.isLose() or d==self.depth:
                return self.evaluationFunction(state)
            elif not agentIndex:
                v=float('-inf')
                for x in state.getLegalActions(0):
                    v=max(v,alphabeta(state.getNextState(0,x),1,d,a,b))
                    if v>b:
                        return v
                    a=max(a,v)
                return v
            else :
                nextIndex=agentIndex+1
                if nextIndex==state.getNumAgents():
                    nextIndex=0
                    d+=1
                v=float('inf')
                for x in state.getLegalActions(agentIndex):
                    v=min(v,alphabeta(state.getNextState(agentIndex,x),nextIndex,d,a,b))
                    if v<a:
                        return v
                    b=min(v,b)
                return v
        '''
        Begin with the top pacman as the former part, but update the alpha value for each action.
        '''
        maxV=float('-inf')
        alpha=float('-inf')
        beta=float('inf')
        maxAct=0
        for x in gameState.getLegalActions(0):
            temp=alphabeta(gameState.getNextState(0,x), 1,0,alpha,beta)
            if temp>maxV:
                maxV=temp
                maxAct=x
                alpha=temp
        return maxAct
```

Expectimax:

```python
'''
In this part I also define a expectimax function to get the possible action with maximun value.
If the agent is ghost, sum the value of all possible action and devide it with the amount of possible
actions.
'''
def expectimax(state, index,d):
    if state.isWin() or state.isLose() or d==self.depth:
        return self.evaluationFunction(state)
    elif not index:
        v=float('-inf')
        for x in state.getLegalActions(0):
            v=max(v,expectimax(state.getNextState(0,x),1,d))
        return v
    else:
        nextIndex=index+1
        if nextIndex==state.getNumAgents():
            nextIndex=0
            d+=1
        sun=0.0
        for x in state.getLegalActions(index):
            sun+=expectimax(state.getNextState(index,x), nextIndex, d)
        return sun/ float(len(state.getLegalActions(index)))
```

```python
maxAct=0
v=float('-inf')
for x in gameState.getLegalActions(0):
    temp=expectimax(gameState.getNextState(0,x),1,0)
    if temp>v:
        v=temp
        maxAct=x
return maxAct
```

Better evaluation function:

```python
'''
In this better evaluation function part, I take the remain food number, remain capsule number, minimum manhattan
distance to the active ghost and minimum manhattan distance to the scared ghost into consideration. Weighted these
attributes and minus them from the current score.
'''
pos=currentGameState.getPacmanPosition()
cap=currentGameState.getCapsules()
foodNum=currentGameState.getNumFood()
bonus=0.0
scaredGhosts, activeGhosts = [], []
for g in currentGameState.getGhostStates():
    if not g.scaredTimer:
        activeGhosts.append(g)
    else:
        scaredGhosts.append(g)

minGhostDis=0
minSghostDis=0
if activeGhosts:
    minGhostDis = min(map(lambda g:manhattanDistance(pos,g.getPosition()),activeGhosts))

if scaredGhosts:
    minSghostDis = min(map(lambda g:manhattanDistance(pos, g.getPosition()), scaredGhosts))
    # bonus-=0.5*minSghostDis
    bonus+=1/minSghostDis
bonus+=3*minGhostDis
bonus+=5*foodNum
bonus+=20*len(cap)
return currentGameState.getScore()-bonus
```

Part II. Results & Analysis:

```
Average Score: 1070.7
Scores:        811.0, 1360.0, 1149.0, 912.0, 1126.0, 1111.0, 1175.0, 1123.0, 910.0, 1030.0
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
***     1070.7 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 ###
```

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

By repeatedly testing, I finally weighted the attributes as

- Amount of remaining food: 5
- Amount of remaining capsules: 20
- Minimum Manhattan distance to the active ghost: 3
- Minimum Manhattan distance to the scared ghost: reciprocal

Initially, I also want to consider the total distance of all remaining food, and their density degree, but after implementation, I found that it's extremely time consuming if I run through the whole map to calculate the total distance, causing nearly time out in every game.