

# Multi-Agent Search report

071125 葉長瀚

## Part I. Implementation:

### Minimax:

```
138     define a minimax function first to get the value recursively. If the agent is Pacman, pick the
139     maximun value; If the agent is Ghosts, pick the minimun value. Go through all possible actions
140     for every agent in each step.
141     '''
142     def minimax(gameStateee, agentIndex, depthh):
143         if gameStateee.isWin() or gameStateee.isLose() or depthh==self.depth:
144             return self.evaluationFunction(gameStateee)
145         elif not agentIndex:
146             v=float('-inf')
147             for x in gameStateee.getLegalActions(0):
148                 v=max(v,minimax(gameStateee.getNextState(0,x),1,depthh))
149             return v
150         else:
151             v=float('inf')
152             nextIndex= agentIndex+1
153             if nextIndex == gameStateee.getNumAgents():
154                 nextIndex=0
155                 depthh+=1
156             for x in gameStateee.getLegalActions(agentIndex):
157                 v=min(v, minimax(gameStateee.getNextState(agentIndex,x),nextIndex, depthh))
158             return v
159
160     '''
161     Begin with the top pacman, return the action result in maximun value.
162     '''
163     maxV= float('-inf')
164     maxAct=0
165     for x in gameState.getLegalActions(0):
166         temp=minimax(gameState.getNextState(0,x),1,0)
167         if maxV<temp:
168             maxV=temp
169             maxAct=x
170     return maxAct
```

## Alphabeta pruning:

```
184     Define the alphabeta function first too. In this function there's 2 addition arguments alpha and beta.
185     In picking maximum part, if the value is larger than beta, which means this value wont be chosen by the
186     upper minimum part, so we can prune it directly; Same in picking minimum part, if the value is smaller than
187     alpha, we can prune it directly.
188     '''
189     def alphabeta(state, agentIndex,d, a, b):
190         if state.isWin() or state.isLose() or d==self.depth:
191             return self.evaluationFunction(state)
192         elif not agentIndex:
193             v=float('-inf')
194             for x in state.getLegalActions(0):
195                 v=max(v,alphabeta(state.getNextState(0,x),1,d,a,b))
196                 if v>b:
197                     return v
198                 a=max(a,v)
199             return v
200         else :
201             nextIndex=agentIndex+1
202             if nextIndex==state.getNumAgents():
203                 nextIndex=0
204                 d+=1
205             v=float('inf')
206             for x in state.getLegalActions(agentIndex):
207                 v=min(v,alphabeta(state.getNextState(agentIndex,x),nextIndex,d,a,b))
208                 if v<a:
209                     return v
210                 b=min(v,b)
211             return v
212
213     '''
214     Begin with the top pacman as the former part, but update the alpha value for each action.
215     '''
216     maxV=float('-inf')
217     alpha=float('-inf')
218     beta=float('inf')
219     maxAct=0
220     for x in gameState.getLegalActions(0):
221         temp=alphabeta(gameState.getNextState(0,x), 1,0,alpha,beta)
222         if temp>maxV:
223             maxV=temp
224             maxAct=x
225             alpha=temp
226     return maxAct
```

## Expectimax:

```
242  '''
243  In this part I also define a expectimax function to get the possible action with maximun value.
244  If the agent is ghost, sum the value of all possible action and devide it with the amount of possible
245  actions.
246  '''
247  def expectimax(state, index,d):
248      if state.isWin() or state.isLose() or d==self.depth:
249          return self.evaluationFunction(state)
250      elif not index:
251          v=float('-inf')
252          for x in state.getLegalActions(0):
253              v=max(v,expectimax(state.getNextState(0,x),1,d))
254          return v
255      else:
256          nextIndex=index+1
257          if nextIndex==state.getNumAgents():
258              nextIndex=0
259              d+=1
260          sun=0.0
261          for x in state.getLegalActions(index):
262              sun+=expectimax(state.getNextState(index,x), nextIndex, d)
263          return sun/ float(len(state.getLegalActions(index)))

265  maxAct=0
266  v=float('-inf')
267  for x in gameState.getLegalActions(0):
268      temp=expectimax(gameState.getNextState(0,x),1,0)
269      if temp>v:
270          v=temp
271          maxAct=x
272  return maxAct
```

## Better evaluation function:

```
282  '''
283  In this better evaluation function part, I take the remain food number, remain capsule number, minimum manhattan
284  distance to the active ghost and minimum manhattan distance to the scared ghost into consideration. Weighted these
285  attributes and minus them from the current score.
286  '''
287  pos=currentGameState.getPacmanPosition()
288  cap=currentGameState.getCapsules()
289  foodNum=currentGameState.getNumFood()
290  bonus=0.0
291  scaredGhosts, activeGhosts = [], []
292  for g in currentGameState.getGhostStates():
293      if not g.scaredTimer:
294          activeGhosts.append(g)
295      else:
296          scaredGhosts.append(g)
297
298  minGhostDis=0
299  minSghostDis=0
300  if activeGhosts:
301      minGhostDis = min(map(lambda g:manhattanDistance(pos,g.getPosition()),activeGhosts))
302
303  if scaredGhosts:
304      minSghostDis = min(map(lambda g:manhattanDistance(pos, g.getPosition()), scaredGhosts))
305      # bonus=-0.5*minSghostDis
306      bonus+=1/minSghostDis
307  bonus+=3*minGhostDis
308  bonus+=5*foodNum
309  bonus+=20*len(cap)
310  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.

The diagram illustrates the construction of a binary tree from a sequence of 16 '0's. The root node is a dashed box containing a '+' sign. It branches into two nodes, each containing a '+' sign. These nodes further branch into four nodes, each containing a '+' sign. The tree continues to branch down to a level of 16 leaf nodes, each containing a '0'. The tree is shown as a series of nested dashed boxes, with the root box at the top and the leaf nodes at the bottom. The tree is labeled 'V' on the left side.

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.