

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,

Part1:

```
136         # Begin your code (Part 1)
137         value,action=self.minimax(gameState,0,0)#call the minimax function
138         print(action)
139         return action
140         raise NotImplementedError("To be implemented")
141
142
143     def minimax(self,gameState,depth,agentIndex):
144         if agentIndex>=gameState.getNumAgents():#if every agent in the game return the value and action, then we finish one round
145             agentIndex=0#set agentIndex=0 means that it's time to player's turn
146             depth+=1#depth represent of many round we going to do
147         if gameState.isWin() or gameState.isLose() or (depth>=self.depth):#(case1: if player win or lose/ case2:if finish the expected number of round)
148             return self.evaluationFunction(gameState), None#in this case, we evaluate the extent of nextState if it is a good choice
149         if agentIndex==0:#is player's turn
150             return self.max_agent(gameState,depth,agentIndex)#player should return the max value
151         else:#is ghost turn
152             return self.min_agent(gameState,depth,agentIndex)#ghost should return the min value
153
154     def max_agent(self,gameState,depth,agentIndex):#for player
155         bestValue=-float('inf')#to do the comparison, initial the value equal to -inf
156         NextAction = None#variable use for record the best next action
157         for action in gameState.getLegalActions(agentIndex):#travel all the possible legal actions
158             NextState=gameState.getNextState(agentIndex,action)#get the nextstate correspond to the legal action
159             value,_=self.minimax(NextState,depth,agentIndex+1)#we should get the low level of value first then we can do the comparison
160             if value>bestValue:#to find the max value
161                 bestValue=value#record the max value
162                 NextAction=action#and record the next action
163         return bestValue,NextAction
164
165     def min_agent(self,gameState,depth,agentIndex):#for ghost
166         bestValue=float('inf')#to do the comparison, initial the value equal to inf
167         NextAction = None
168         for action in gameState.getLegalActions(agentIndex):
169             NextState=gameState.getNextState(agentIndex,action)
170             value,_=self.minimax(NextState,depth,agentIndex+1)
171             if value<bestValue:#to find the min value
172                 bestValue=value#record the min value
173                 NextAction=action#and return next action
174         return bestValue,NextAction
175         # End your code (Part 1)
```

Part2:

```
187     # Begin your code (Part 2)
188     """
189     the difference between part2 and part 1 is the addition variable alpha and beta
190     the initial value of alpha equal to -inf
191     the initial value of beta equal to inf
192     """
193     value,action=self.minimax(gameState,0,0,float('-inf'),float('inf'))#call minimax function
194     print(action)
195     return action
196     raise NotImplementedError("To be implemented")
197
198
199 def minimax(self,gameState,depth,agentIndex,alpha,beta):
200     if agentIndex>=gameState.getNumAgents():
201         agentIndex=0
202         depth+=1
203     if gameState.isWin() or gameState.isLose() or (depth>=self.depth):
204         return self.evaluationFunction(gameState), None
205     if agentIndex==0:
206         return self.max_agent(gameState,depth,agentIndex,alpha,beta)#add two addition variable
207     else:
208         return self.min_agent(gameState,depth,agentIndex,alpha,beta)
209
210 def max_agent(self,gameState,depth,agentIndex,alpha,beta):
211     bestValue=-float('inf')
212     NextAction = None
213     for action in gameState.getLegalActions(agentIndex):
214         NextState=gameState.getNextState(agentIndex,action)
215         value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
216         if value>bestValue:
217             bestValue=value
218             NextAction=action
219     """
220     add the judgmental of alpha and beta
221     """
222     if bestValue>beta:
223         break
224     if bestValue>alpha:
225         alpha=max(alpha,bestValue)
226     return bestValue,NextAction
227
228 def min_agent(self,gameState,depth,agentIndex,alpha,beta):
229     bestValue=float('inf')
230     NextAction = None
231     for action in gameState.getLegalActions(agentIndex):
232         NextState=gameState.getNextState(agentIndex,action)
233         value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
234         if value<bestValue:
235             bestValue=value
236             NextAction=action
237         if bestValue<alpha:
238             break
239         if bestValue<beta:
240             beta=min(beta,bestValue)
241     return bestValue,NextAction
242     # End your code (Part 2)
```

Part3:

```
257     # Begin your code (Part 3)
258     """
259     change the min_agent function in part 3
260     the algorithm should change the outcomes to average-case, not worst-case
261     """
262     value,action=self.minimax(gameState,0,0,float('-inf'),float('inf'))
263     print(action)
264     return action
265     raise NotImplementedError("To be implemented")
266
267
268     def minimax(self,gameState,depth,agentIndex,alpha,beta):
269         if agentIndex>=gameState.getNumAgents():
270             agentIndex=0
271             depth+=1
272         if gameState.isWin() or gameState.isLose() or (depth>=self.depth):
273             return self.evaluationFunction(gameState), None
274         if agentIndex==0:
275             return self.max_agent(gameState,depth,agentIndex,alpha,beta)
276         else:
277             return self.min_agent(gameState,depth,agentIndex,alpha,beta)
278
279     def max_agent(self,gameState,depth,agentIndex,alpha,beta):
280         bestValue=-float('inf')
281         NextAction = None
282         for action in gameState.getLegalActions(agentIndex):
283             NextState=gameState.getNextState(agentIndex,action)
284             value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
285             if value>bestValue:
286                 bestValue=value
287                 NextAction=action
288             if bestValue>beta:
289                 break
290             if bestValue>alpha:
291                 alpha=max(alpha,bestValue)
292         return bestValue,NextAction
293
294     def min_agent(self,gameState,depth,agentIndex,alpha,beta):
295         total_value=0
296         actionNum=0
297         NextAction = None
298         for action in gameState.getLegalActions(agentIndex):
299             NextState=gameState.getNextState(agentIndex,action)
300             value,_=self.minimax(NextState,depth,agentIndex+1,alpha,beta)
301             total_value+=value
302             actionNum+=1
303         bestValue=total_value/actionNum#calculate the average-case value
304         return bestValue,NextAction
305     # End your code (Part 3)
```

Part4:

```
313 # Begin your code (Part 4)
314 GhostStates = currentGameState.getGhostStates() #all the ghost states
315 Pacman_Pos = currentGameState.getPacmanPosition()
316 food_list = (currentGameState.getFood()).asList() #get all the food as list.
317 capsule_list = currentGameState.getCapsules() #get all the capsules.
318 no_food = len(food_list) #check that if there still have food or not
319 no_capsule = len(capsule_list) #chenck if there still have capsule or not
320
321 state_score=0 #variable for record the final score
322 if currentGameState.getNumAgents() > 1:#>1 represent that there are ghost exist
323     ghost_dis = min( [manhattanDistance(Pacman_Pos, ghost.getPosition()) for ghost in GhostStates])# search the nearest ghost distance
324     if (ghost_dis <= 1):
325         return -10000 #if nearest ghost distance is small than 1, this is the worst case so return -10000
326     state_score -= 1.0/ghost_dis #minus the score with 1/(distance between player and ghost)
327 #Feature 3 food positions
328 current_food = Pacman_Pos
329 for food in food_list: #calculate all the food distance and update the score
330     closestFood = min(food_list, key=lambda x: manhattanDistance(x, current_food))
331     state_score += 1.0/(manhattanDistance(current_food, closestFood)) #plus the score with 1/(the distance between player and food)
332     current_food = closestFood #then change the pacman position to the previous eaten food
333     food_list.remove(closestFood) # remove the food already been eaten
334 #Feature 4 capsule positions, same idea as food position
335 current_capsule = Pacman_Pos
336 for capsule in capsule_list:
337     closest_capsule = min(capsule_list, key=lambda x: manhattanDistance(x, current_capsule))
338     state_score += 1.0/(manhattanDistance(current_capsule, closest_capsule))
339     current_capsule = closest_capsule
340     capsule_list.remove(closest_capsule)
341 #Feature 4 Score of the game
342 state_score += 8*(currentGameState.getScore())
343
344 #Feature 5: remaining food and capsule
345 state_score -= 6*(no_food + no_capsule)
346
347 return state_score
348 # 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.**

```
=====
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.

[illegible]