

Homework 3: Multi-Agent Search

110550080 何曉嫻

Part I. Implementation (5%):

Part 1

```
136 # Begin your code (Part 1)
137 """
138 1. call minimax function
139 2. depth: the current depth of the tree
140 | agentIndex: index of the current agent
141 3. Roll over agent index and increase current depth if all agents have finished
142 4. Return the value of evaluationFunction if max depth is reached
143 5. If it is max (pacman) turn:
144 | Get the maximum score of successor
145 | Increase agent_index by 1 as it will be next player's (ghost) turn
146 | Pass the new game state generated by pacman's action
147 | Update the best score and action
148 6. If it is min (ghost) turn:
149 | Get the minimum score of successor
150 | Increase agent_index by 1 as it will be next player's (ghost or pacman) turn
151 | Pass the new game state generated by ghost's action
152 | Update the best score and action
153 7. If it is a leaf state with no successor states, return the value of evaluationFunction
154 8. Return the best action and best score
155 9. getAction function return the action
156 """
157 action, score = self.minimax(0, 0, gameState)
158 return action
159 def minimax(self, depth, agentIndex, gameState):
160     if agentIndex >= gameState.getNumAgents():
161         agentIndex = 0
162         depth += 1
163     if (depth == self.depth):
164         return None, self.evaluationFunction(gameState)
165
166     best_action, best_score = None, None
167     actions = gameState.getLegalActions(agentIndex)
168     if agentIndex == 0:
169         for action in actions:
170             next = gameState.getNextState(agentIndex, action)
171             score = self.minimax(depth, agentIndex + 1, next)
172             if best_score == None or score > best_score:
173                 best_score = score
174                 best_action = action
175     else:
176         for action in actions:
177             next = gameState.getNextState(agentIndex, action)
178             score = self.minimax(depth, agentIndex + 1, next)
179             if best_score == None or score < best_score:
180                 best_score = score
181                 best_action = action
182     if best_score is None:
183         return None, self.evaluationFunction(gameState)
184     return best_action, best_score
185
186 raise NotImplementedError("To be implemented")
187 # End your code (Part 1)
```

Part 2

```
199 # Begin your code (Part 2)
200 """
201 1. Same as minimax
202 2. Add alpha and beta
203 | alpha: The best choice (highest-value) we have found so far along the path of Maximizer.
204 | The initial value of alpha is -inf.
205 | beta: The best choice (lowest-value) we have found along the path of Minimizer.
206 | The initial value of beta is inf.
207 3. Prune the tree if alpha is greater than beta
208 """
209 action, score = self.AlphaBeta(0, 0, gameState, -float('inf'), float('inf'))
210 return action
211 def AlphaBeta(self, depth, agentIndex, gameState, alpha, beta):
212     if agentIndex >= gameState.getNumAgents():
213         agentIndex = 0
214         depth += 1
215     if (depth == self.depth):
216         return None, self.evaluationFunction(gameState)
217
218     best_action, best_score = None, None
219     actions = gameState.getLegalActions(agentIndex)
220     if agentIndex == 0:
221         for action in actions:
222             next = gameState.getNextState(agentIndex, action)
223             score = self.AlphaBeta(depth, agentIndex + 1, next, alpha, beta)
224             if best_score == None or score > best_score:
225                 best_score = score
226                 best_action = action
227             if score > alpha:
228                 alpha = score
229             if alpha > beta:
230                 break
231     else:
232         for action in actions:
233             next = gameState.getNextState(agentIndex, action)
234             score = self.AlphaBeta(depth, agentIndex + 1, next, alpha, beta)
235             if best_score == None or score < best_score:
236                 best_score = score
237                 best_action = action
238             if score < beta:
239                 beta = score
240             if alpha > beta:
241                 break
242     if best_score is None:
243         return None, self.evaluationFunction(gameState)
244     return best_action, best_score
245
246 raise NotImplementedError("To be implemented")
```

Part 3

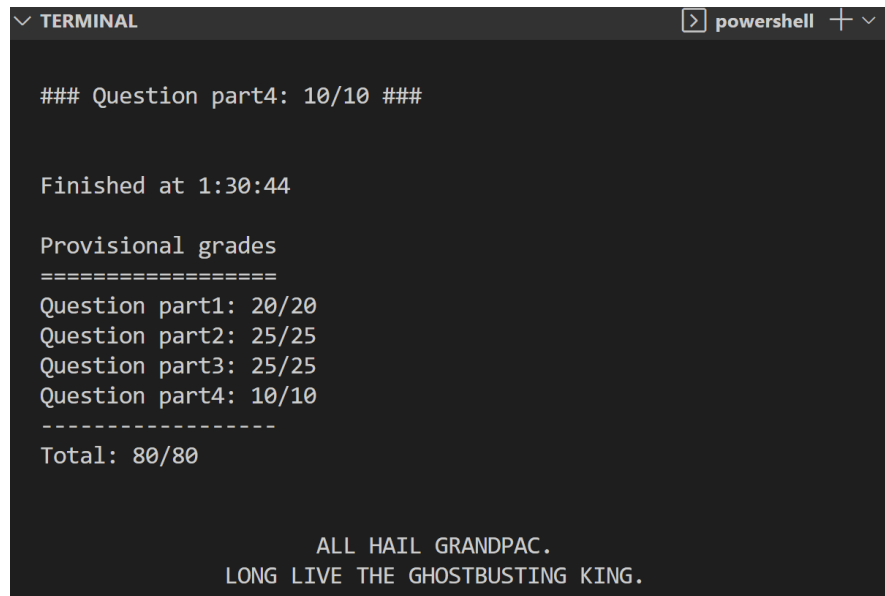
```
261 # Begin your code (Part 3)
262 """
263 1. Same as minimax, but changes the "mini" part to "expect"
264 2. If it is chance (ghost) turn:
265     The score is expected value of its children
266     The best action is choosed uniformly at random
267 """
268 action, score = self.Expectimax(0, 0, gameState)
269 return action
270 def Expectimax(self, depth, agentIndex, gameState):
271     if agentIndex >= gameState.getNumAgents():
272         agentIndex = 0
273         depth += 1
274     if (depth == self.depth):
275         return None, self.evaluationFunction(gameState)
276
277     best_action, best_score = None, None
278     actions = gameState.getLegalActions(agentIndex)
279     if agentIndex == 0:
280         for action in actions:
281             next = gameState.getNextState(agentIndex, action)
282             _, score = self.Expectimax(depth, agentIndex + 1, next)
283             if best_score == None or score > best_score:
284                 best_score = score
285                 best_action = action
286
287     else:
288         n = len(actions)
289         for action in actions:
290             next = gameState.getNextState(agentIndex, action)
291             _, score = self.Expectimax(depth, agentIndex + 1, next)
292             if best_score == None:
293                 best_score = 0
294             best_score += score * 1 / n
295             best_action = action
296
297     if best_score is None:
298         return None, self.evaluationFunction(gameState)
299     return best_action, best_score
300     raise NotImplementedError("To be implemented")
301 # End your code (Part 3)
```

Part 4

```
309 # Begin your code (Part 4)
310 """
311 1. Calculating distance to the closest food pellet
312 2. Calculating the distances from pacman to the ghosts.
313     Also, checking for the ghosts around pacman (at distance of 1).
314 3. Obtaining the number of capsules available
315 4. Combination of the above calculated metrics.
316 """
317 position = currentGameState.getPacmanPosition()
318 foods = currentGameState.getFood().asList()
319 ghostState = currentGameState.getGhostPositions()
320 capsules = currentGameState.getCapsules()
321 min_food = -1
322 for food in foods:
323     distance = manhattanDistance(position, food)
324     if distance < min_food or min_food == -1:
325         min_food = distance
326 ghost_dis = 1
327 scared_ghost = 0
328 for ghost in ghostState:
329     distance = manhattanDistance(position, ghost)
330     ghost_dis += distance
331     if distance <= 1:
332         scared_ghost += 1
333 capsule = len(capsules)
334 return currentGameState.getScore() + (1 / float(min_food)) - (1 / float(ghost_dis)) - scared_ghost - capsule
335 raise NotImplementedError("To be implemented")
336 # End your code (Part 4)
```

Part II. Results & Analysis (5%):

Using manhattanDistance to calculating distance is best, and the scores are close. Tuning these parameters and coming up with these calculation functions are kind of arbitrary to me because the only way to tell if the formula works is to test it on the map. It is empirically proved to be fine, but it may not represent the real optimal evaluation strategy of the game. It is slightly biased by the map.

A terminal window titled 'TERMINAL' with a 'powershell' icon and window controls. It displays the following text:

```
### Question part4: 10/10 ###  
  
Finished at 1:30:44  
  
Provisional grades  
=====
```

Question	Score
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.
```