

Spring 2022

## Introduction to Artificial Intelligence

### Homework 3: Multi-Agent Search 109550159 李驊恩

#### Part I. Implementation

The explanation is in the notation of the code.

#### Part1. Minimax Search

```
136 # Begin your code (Part 1)
137
138 def MinimaxFunction(gameState, agt, depth):
139     ans = []
140
141     # If the game is ended (win or lose)
142     if gameState.isWin():
143         return self.evaluationFunction(gameState), 0
144     if gameState.isLose():
145         return self.evaluationFunction(gameState), 0
146
147     # If it arrive max depth
148     if depth == self.depth:
149         return self.evaluationFunction(gameState), 0
150     if agt == gameState.getNumAgents() - 1:
151         depth += 1
152     if agt == gameState.getNumAgents() - 1:
153         nxt_agt = self.index
154     else:
155         nxt_agt = agt + 1
156
157     # The function that fix the answer with minimax value
158     def FixingFunction():
159         nxt_val = MinimaxFunction(gameState.getNextState(agt, action), nxt_agt, depth)
160         ans.append(nxt_val[0])
161         ans.append(action)
162
163     # The function that check the minimax value
164     def CheckingFunction():
165         prvs_val = ans[0]
166         nxt_val = MinimaxFunction(gameState.getNextState(agt, action), nxt_agt, depth)
167         if agt == self.index and nxt_val[0] > prvs_val:
168             ans[0] = nxt_val[0]
169             ans[1] = action
170         elif agt != self.index and nxt_val[0] <= prvs_val:
171             ans[0] = nxt_val[0]
172             ans[1] = action
173
174     # Find every minimax value
175     for action in gameState.getLegalActions(agt):
176         if len(ans) == 0:
177             FixingFunction()
178         else:
179             CheckingFunction()
180
181     return ans
182
183 return MinimaxFunction(gameState, self.index, 0)[1]
184 # End your code (Part 1)
185
```

## Part 2: Alpha-Beta Pruning

```
196 # Begin your code (Part 2)
197 def AlphaBetaPruning(gameState, agt, depth, alpha, beta):
198     ans = []
199
200     # If the game is ended (win or lose)
201     if gameState.isWin():
202         return self.evaluationFunction(gameState), 0
203     if gameState.isLose():
204         return self.evaluationFunction(gameState), 0
205
206     # If it arrive max depth
207     if depth == self.depth:
208         return self.evaluationFunction(gameState), 0
209     if agt == gameState.getNumAgents() - 1:
210         depth += 1
211     if agt == gameState.getNumAgents() - 1:
212         nxt_agt = self.index
213     else:
214         nxt_agt = agt + 1
215
216
217     for action in gameState.getLegalActions(agt):
218         if len(ans) == 0:
219             nxt_val = AlphaBetaPruning(gameState.getNextState(agt, action), nxt_agt, depth, alpha, beta)
220
221
222             ans.append(nxt_val[0])
223             ans.append(action)
224
225     # Modified the initial value of alpha and beta
226     if agt == self.index:
227         if ans[0] > alpha:
228             alpha = ans[0]
229         else:
230             if ans[0] < beta:
231                 beta = ans[0]
232
233     else:
234         # If the alpha-beta pruning is correct
235         if ans[0] > beta:
236             if agt == self.index:
237                 return ans
238
239         if ans[0] < alpha:
240             if agt != self.index:
241                 return ans
242
243     prvs_val = ans[0]
244     nxt_val = AlphaBetaPruning(gameState.getNextState(agt, action), nxt_agt, depth, alpha, beta)
245
246     # Max agent
247     if agt == self.index:
248         if nxt_val[0] > prvs_val:
249             ans[0] = nxt_val[0]
250             ans[1] = action
251             if ans[0] > alpha:
252                 alpha = ans[0]
253
254     # Min agent
255     else:
256         if nxt_val[0] < prvs_val:
257             ans[0] = nxt_val[0]
258             ans[1] = action
259             if ans[0] < beta:
260                 beta = ans[0]
261
262     return ans
263
264 return AlphaBetaPruning(gameState, self.index, 0, -float("inf"), float("inf"))[1]
265 # End your code (Part 2)
```

### Part 3: Expectimax Search

```
279 # Begin your code (Part 3)
280 def ExpectivemaxFunction(gameState, agt, depth):
281     ans = []
282
283     # If the game is ended (win or lose)
284     if gameState.isWin():
285         return self.evaluationFunction(gameState), 0
286     if gameState.isLose():
287         return self.evaluationFunction(gameState), 0
288
289     # If it arrive max depth
290     if depth == self.depth:
291         return self.evaluationFunction(gameState), 0
292     if agt == gameState.getNumAgents() - 1:
293         depth += 1
294     if agt == gameState.getNumAgents() - 1:
295         nxt_agt = self.index
296     else:
297         nxt_agt = agt + 1
298
299     # The function that fix chance node
300     # p = 1 / total legal actions
301     def FixingFunction():
302
303         nxt_val = ExpectivemaxFunction(gameState.getNextState(agt, action), nxt_agt, depth)
304
305         if (agt != self.index):
306             ans.append((1.0 / len(gameState.getLegalActions(agt))) * nxt_val[0])
307             ans.append(action)
308         else:
309             ans.append(nxt_val[0])
310             ans.append(action)
311
312     # The function that check the minimax value
313     def CheckingFunction():
314         prvs_val = ans[0]
315         nxt_val = ExpectivemaxFunction(gameState.getNextState(agt, action), nxt_agt, depth)
316
317         if agt == self.index:
318             if nxt_val[0] > prvs_val:
319                 ans[0] = nxt_val[0]
320                 ans[1] = action
321             else:
322                 ans[0] = ans[0] + (1.0 / len(gameState.getLegalActions(agt))) * nxt_val[0]
323                 ans[1] = action
324
325         for action in gameState.getLegalActions(agt):
326
327             if len(ans) == 0 :
328                 FixingFunction()
329
330             else:
331                 CheckingFunction()
332
333         return ans
334
335     return ExpectivemaxFunction(gameState, self.index, 0)[1]
336
337 # End your code (Part 3)
```

## Part 4: Evaluation Function

```
344 # Begin your code (Part 4)
345
346 # Get the list of the food
347 food_list = currentGameState.getFood().asList()
348
349 # Get the total number of the food
350 Num_of_food = len(food_list)
351
352 # Get the total number of capsules
353 Num_of_cap = len(currentGameState.getCapsules())
354
355 # Get the position of pacman
356 pacman_pos = currentGameState.getPacmanPosition()
357
358 # Get the position of ghosts
359 ghost_pos = currentGameState.getGhostPositions()
360
361 # Get the state of the ghosts
362 ghost_state = currentGameState.getGhostStates()
363
364 # Initialize the distance of the nearest food
365 nearest_food = 1
366
367 # The lists of normal ghosts and those can be eaten.
368 normal_ghost = []
369 weak_ghost = []
370
371 # The lists of distances to normal ghosts and those can be eaten.
372 normal_ghost_distance = []
373 weak_ghost_distance = []
374
375 # Get the score
376 score = currentGameState.getScore()
377
378 # Initialize the evaluation value
379 Evaluation_Value = 0
380
381 # Add those scared ghost into the weak ghost list and
382 # the other to the normal ghost list
383 for ghost in ghost_state:
384     if ghost.scaredTimer:
385         weak_ghost.append(ghost)
386     else:
387         normal_ghost.append(ghost)
388
389 # The list of distances of pacman to the food
390 food_distances = [manhattanDistance(pacman_pos, food_position) for food_position in food_list]
391
392 # There always be a nearest food while there is food have not been eaten
393 if Num_of_food > 0:
394     nearest_food = min(food_distances)
395
396 # The list of the distances of pacman and ghosts
397 for ghost_position in ghost_pos:
398     ghost_distance = manhattanDistance(pacman_pos, ghost_position)
399
400 # If the ghost is too close to the nearest food,
401 # set the distance of the nearest food a bigger value
402
403     if ghost_distance <= 1:
404         nearest_food = 10000 # Set the value very big if the ghost is super close
405                             # because I don't want it to lose
406     elif ghost_distance <= 2:
407         nearest_food = 3
408
409 # Get manhattan distance
410 for ghost in normal_ghost:
411     weak_ghost_distance.append(manhattanDistance(pacman_pos, ghost.getPosition()))
412
413 for ghost in weak_ghost:
414     weak_ghost_distance.append(manhattanDistance(pacman_pos, ghost.getPosition()))
415
416 # Give value to ghosts that can be eaten
417 for i in weak_ghost_distance:
418     if i < 3:
419         Evaluation_Value += -30 * i
420     else:
421         Evaluation_Value += -10 * i
422
423 # Give value to ghosts that are normal
424 for i in normal_ghost_distance:
425     if i <= 1:
426         Evaluation_Value += 20 * i
427     elif i <= 3:
428         Evaluation_Value += 10 * i
429     elif i <= 5:
430         Evaluation_Value += 5 * i
431     else:
432         Evaluation_Value += 1 * i
433
434 # Add the value of each weights * features
435 Evaluation_Value += 20 / nearest_food
436 Evaluation_Value += 100 * score
437 Evaluation_Value += (-150) * Num_of_food
438 Evaluation_Value += (-30) * Num_of_cap
439
440 return Evaluation_Value
441 # End your code (Part 4)
442
443 # Better Evaluation Function
```

## Part II. Results & Analysis

Question part4

```
Pacman emerges victorious! Score: 1123
Pacman emerges victorious! Score: 1099
Pacman emerges victorious! Score: 1162
Pacman emerges victorious! Score: 1140
Pacman emerges victorious! Score: 1316
Pacman emerges victorious! Score: 1309
Pacman emerges victorious! Score: 1139
Pacman emerges victorious! Score: 1161
Pacman emerges victorious! Score: 1122
Pacman emerges victorious! Score: 1338
Average Score: 1190.9
Scores: 1123.0, 1099.0, 1162.0, 1140.0, 1316.0, 1309.0, 1139.0, 1161.0, 1122.0, 1338.0
Win Rate: 10/10 (1.00)
Record: Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\part4\grade-agent.test (8 of 8 points)
*** EXTRA CREDIT: 2 points
*** 1190.9 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 21:45:29

### 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 is a complex fractal-like structure composed of black lines and symbols. At the top, a dashed rectangle contains a grid of '+' and '/' symbols. Below this, a series of horizontal lines are decorated with 'o' characters. The diagram then branches into several paths, some marked with 'V' and others with '\', leading to more horizontal lines with 'o' characters. The structure is symmetrical and recursive, resembling a tree or a complex network.