

Homework 3: Multi-Agent Search

Part I. Implementation (5%):

Part 1 Minimax

```
122 # Begin your code (Part 1)
123
124 def countmax(depth, gameState, index_agent):
125     '''
126     depth = the depth this state is in
127     gameState = the state that is going to be execute
128     index_agent = the index of the agent (pacman = 0, ghosts = 1~(num_agents-1))
129     bestscore is either min/max score in ghost/pacman
130     bestmove only matters when it comes to pacman(index_agent=0)
131     countmax returns the bestscore and bestmove
132     '''
133     num_agents = gameState.getNumAgents() # number of agents
134     legal_moves = gameState.getLegalActions(index_agent) # legal moves of the executing state
135     if not len(legal_moves): # no legal_moves = done with game at the state (win or lose)
136         return self.evaluationFunction(gameState), "0" # return the points
137     '''
138     in each condition, scores means the scores of all childstates
139     take all possible childstates from legalmove and count the min/max of it
140     '''
141     if depth == 1 and index_agent == num_agents-1:
142         '''
143         depth = 1 and index = num_agents-1 means it is the terminal state
144         it is also the leaf process of the recursion function call
145         simply uses the evaluationFunction to calculate the scores since it's the terminal tate
146         return best_score, which is the minimum score of all possible scores
147         '''
148         scores = []
149         for action in legal_moves:
150             GameState = gameState.getNextState(index_agent, action)
151             scores.append(self.evaluationFunction(GameState))
152         best_score = min(scores)
153         best_indices = [index for index in range(len(scores)) if scores[index] == best_score]
154         chosen_idx = random.choice(best_indices) # pick randomly among the best
155         return best_score, legal_moves[chosen_idx]
156
157     elif index_agent == 0:
158         '''
159         index_agent = 0 means it is the time when pacman(max_player)'s time
160         countmax(depth, GameState, 1) is used because the ghost with index 1 starts
161         return best_score (max score of scores)
162         and bestMove (which matters because the original get_action function relies on it)
163         '''
164         scores = []
165         for action in legal_moves:
166             GameState = gameState.getNextState(0, action)
167             scores.append(countmax(depth, GameState, 1))
168         best_score = max(scores)[0]
169         best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
170         chosen_idx = random.choice(best_indices) # pick randomly among the best
171         return best_score, legal_moves[chosen_idx]
172
```

```

173         elif index_agent == num_agents-1:
174             '''
175             index_agent = num_agents-1 but depth ≠ 1 means the childstate is pacman in the next depth
176             thus, countmax(depth-1, GameState, 0) is used
177             instead of countmax(depth, GameState, index_agent+1)
178             return best_score, which is the minimum of scores.
179             '''
180             scores=[]
181             for action in legal_moves:
182                 GameState = gameState.getNextState(index_agent,action)
183                 scores.append(countmax(depth-1, GameState, 0))
184             best_score = min(scores)[0]
185             best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
186             chosen_idx = random.choice(best_indices) # pick randomly among the best
187             return best_score, legal_moves[chosen_idx]
188         else:
189             '''
190             those 0 < index_agent < num_agent-1 (no matter the depth) would be execute here
191             countmax(depth, GameState, index_agent+1) is used since there is no need to change depth
192             and the next agent to be executed is index_agent+1
193             '''
194             scores=[]
195             for action in legal_moves:
196                 GameState = gameState.getNextState(index_agent, action)
197                 scores.append(countmax(depth, GameState, index_agent+1))
198             best_score = min(scores)[0]
199             best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
200             chosen_idx = random.choice(best_indices) # pick randomly among the best
201             return best_score, legal_moves[chosen_idx]
202
203     # take the best move from function countmax recursively
204     best_score, bestmove = countmax(self.depth,gameState,0)
205     return bestmove
206     # End your code (Part 1)

```

Part 2 AlphaBeta

```

218     # Begin your code (Part 2)
219
220     def countmax(depth, gameState, index_agent, alpha_beta):
221         # alpha_beta[0] = alpha
222         # alpha_beta[1] = beta
223         num_agents = gameState.getNumAgents()
224         legal_moves = gameState.getLegalActions(index_agent)
225         if not len(legal_moves): # no legal_moves = done with game at the state (win or lose)
226             return self.evaluationFunction(gameState),"0" # return the points
227         if depth == 1 and index_agent == num_agents-1:
228             '''
229             this is a min_player, so update beta and prune if best_score(v) is less than alpha
230             the updated alpha_beta list will be changed in the function that call this function
231             because list is mutable
232             '''
233             scores = []
234             best_score = float("inf")
235             for action in legal_moves:
236                 GameState = gameState.getNextState(index_agent, action)
237                 thisScore = self.evaluationFunction(GameState)
238                 scores.append(thisScore)
239                 best_score = min(best_score, thisScore)
240                 if best_score < alpha_beta[0]:
241                     return best_score, action
242                 alpha_beta[1] = min(alpha_beta[1], best_score)
243             best_indices = [index for index in range(len(scores)) if scores[index] == best_score]
244             chosen_idx = random.choice(best_indices) # pick randomly among the best
245             return best_score, legal_moves[chosen_idx]
246

```

```

247 elif index_agent == 0:
248     '''
249     this is a max_player, so update alpha and prune if best_score is larger than beta
250     the updated alpha_beta list will be changed in the function that call this function
251     because list is mutable
252     ab is used in order to not change the beta's value
253     (since this is a max_player, value of beta shouldn't be changed)
254     '''
255     scores = []
256     best_score = -float("inf")
257
258     for action in legal_moves:
259         GameState = gameState.getNextState(0, action)
260         ab = [alpha_beta[0], alpha_beta[1]]
261         thisScore = countmax(depth, GameState, 1, ab)
262         scores.append(thisScore)
263         best_score = max(best_score, thisScore[0])
264         if best_score > alpha_beta[1]:
265             return best_score, action
266         ab[0] = max(ab[0], best_score)
267         alpha_beta[0] = ab[0]
268     best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
269     chosen_idx = random.choice(best_indices) # pick randomly among the best
270     return best_score, legal_moves[chosen_idx]
271

```

```

272 elif index_agent == num_agents-1:
273     '''
274     this is a min_player, so update beta and prune if best_score(v) is less than alpha
275     the updated alpha_beta list will be changed in the function that call this function
276     because list is mutable
277     '''
278     scores = []
279     best_score = float("inf")
280     for action in legal_moves:
281         ab = [alpha_beta[0], alpha_beta[1]]
282         GameState = gameState.getNextState(index_agent, action)
283         thisScore = countmax(depth-1, GameState, 0, ab)
284         scores.append(thisScore)
285         best_score = min(best_score, thisScore[0])
286         if best_score < alpha_beta[0]:
287             return best_score, action
288         ab[1] = min(ab[1], best_score)
289         alpha_beta[1] = ab[1]
290     best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
291     chosen_idx = random.choice(best_indices) # pick randomly among the best
292     return best_score, legal_moves[chosen_idx]

```

```

294     else:
295         '''
296         those 0 < index_agent < num_agent-1 (no matter the depth) would be execute here
297         this is a min_player, so update beta and prune if best_score(v) is less than alpha
298         the updated alpha_beta list will be changed in the function that call this function,
299         because list is mutable
300         '''
301         scores = []
302         best_score = float("inf")
303         for action in legal_moves:
304             ab = [alpha_beta[0], alpha_beta[1]]
305             GameState = gameState.getNextState(index_agent, action)
306             thisScore = countmax(depth, GameState, index_agent+1, ab)
307             scores.append(thisScore)
308             best_score = min(best_score, thisScore[0])
309             if best_score < alpha_beta[0]:
310                 return best_score, action
311             ab[1] = min(ab[1], best_score)
312             alpha_beta[1] = ab[1]
313         best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
314         chosen_idx = random.choice(best_indices) # pick randomly among the best
315         return best_score, legal_moves[chosen_idx]
316 #alpha_beta is sent as a parameter, which is new from minimax
317 best_score, bestmove = countmax(self.depth, gameState, 0, [-float("inf"), float("inf")])
318 return bestmove
319 # End your code (Part 2)

```

Part 3 Expectimax

```

333 # Begin your code (Part 3)
334
335 def countmax(depth, gameState, index_agent):
336     num_agents = gameState.getNumAgents()
337     legal_moves = gameState.getLegalActions(index_agent)
338     if not len(legal_moves): # no legal_moves = done with game at the state (win or lose)
339         return self.evaluationFunction(gameState), "0" # return the points
340     if depth == 1 and index_agent == num_agents-1:
341         scores = []
342         for action in legal_moves:
343             GameState = gameState.getNextState(index_agent, action)
344             scores.append(self.evaluationFunction(GameState))
345         total = 0
346         for value in scores: total += value
347         best_score = total/len(scores)
348         '''
349         instead of finding the minimum of the scores,
350         find the average score as the best score
351         bestmove is not important here,
352         so just return legal_moves[0] is okay
353         '''
354         return best_score, legal_moves[0]
355

```

```

356         elif index_agent == 0:
357             scores = []
358             for action in legal_moves:
359                 GameState = gameState.getNextState(0, action)
360                 scores.append(countmax(depth, GameState, 1))
361             best_score = max(scores)[0]
362             best_indices = [index for index in range(len(scores)) if scores[index][0] == best_score]
363             chosen_idx = random.choice(best_indices) # pick randomly among the best
364             '''
365             the max_player, which is the pacman, remains the same as minimaxAgent
366             '''
367             return best_score, legal_moves[chosen_idx]
368
369         elif index_agent == num_agents-1:
370             scores = []
371             for action in legal_moves:
372                 GameState = gameState.getNextState(index_agent, action)
373                 scores.append(countmax(depth-1, GameState, 0))
374             total = 0
375             for value in scores:
376                 total += value[0]
377             best_score = total/len(scores)
378             '''
379             instead of finding the minimum of the scores,
380             find the average score as the best score
381             bestmove is not important here, so just return legal_moves[0] is okay
382             '''
383             return best_score, legal_moves[0]
384
385         else:
386             '''
387             those 0 < index_agent < num_agent-1 (no matter the depth) would be execute here
388             '''
389             scores = []
390             for action in legal_moves:
391                 GameState = gameState.getNextState(index_agent, action)
392                 scores.append(countmax(depth, GameState, index_agent+1))
393             total = 0
394             for value in scores:
395                 total += value[0]
396             best_score = total/len(scores)
397             '''
398             instead of finding the minimum of the scores,
399             find the average score as the best score
400             bestmove is not important here, so just return legal_moves[0] is okay
401             '''
402             return best_score, legal_moves[0]
403
404     best_score, bestmove = countmax(self.depth, gameState, 0)
405     return bestmove
406     # End your code (Part 3)

```

Part II. Results & Analysis (5%):

```
AI_HW3 — -zsh — 80x24

***      >= 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 20:33:28

Provisional grades
=====
Question part1: 20/20
Question part2: 25/25
Question part3: 25/25
Question part4: 10/10
-----
Total: 80/80
```