

Homework 3: Multi-Agent Search 109550171 陳存佩

Part I. Implementation (5%):

Part 1

```
106 class MinimaxAgent(MultiAgentSearchAgent):
107     """Your minimax agent (Part 1)"""
108     def getAction(self, gameState):
109         """...
110
111         # Begin your code (Part 1)
112         def performMinimax(depth, agentIndex, gameState):
113             if (gameState.isWin() or gameState.isLose() or depth > self.depth): # Terminal condition (win or lose or exceed depth)
114                 return self.evaluationFunction(gameState) # Get evaluation
115
116             value = [] # Store the values for this node
117             todo = gameState.getLegalActions(agentIndex) # Get all legal actions (a list) of an agent
118             for action in todo:
119                 successor = gameState.getNextState(agentIndex, action) # The gameState after taking the legal action
120                 if ((agentIndex+1) >= gameState.getNumAgents()): # When all agents are done
121                     value += [performMinimax(depth+1, 0, successor)] # Pacman go to the next level
122                 else:
123                     value += [performMinimax(depth, agentIndex+1, successor)]
124                 # Calculate ghosts's min value for this successor(next possible state)
125                 # Ghost n will use ghost n-1's successors. After getting the last ghost's value, we go backtracking.
126
127             if agentIndex == 0: # Pacman
128                 if(depth == 1): # Back to root : return action
129                     for i in range( len(value) ):
130                         if (value[i] == max(value)):
131                             return todo[i]
132                 else:
133                     return max(value) # Not a root : return max value
134             elif agentIndex > 0: # Ghosts : return min value
135                 return min(value)
136
137         return performMinimax(1, 0, gameState) # Go to the function with agent 0 (pacman),depth 1
138
139         # End your code (Part 1)
```

Part 2

```
158 class AlphaBetaAgent(MultiAgentSearchAgent):
159     """Your minimax agent with alpha-beta pruning (Part 2)"""
160     def getAction(self, gameState):
161         """Returns the minimax action using self.depth and self.evaluationFunction"""
162         # Begin your code (Part 2)
163         def performAlphaBeta(depth, agentIndex, gameState, alpha, beta): # alpha : best for max, beta : best for min
164             if (gameState.isWin() or gameState.isLose() or depth > self.depth): # Terminal condition (win or lose or exceed depth)
165                 return self.evaluationFunction(gameState) # Get evaluation
166
167             valueList = [] # Store the values for this node
168             todo = gameState.getLegalActions(agentIndex) # Get all legal actions (a list) of an agent
169             for action in todo:
170                 successor = gameState.getNextState(agentIndex, action) # The gameState after taking the legal action
171                 if ((agentIndex+1) >= gameState.getNumAgents()): # When all agents are done
172                     value = performAlphaBeta(depth+1, 0, successor, alpha, beta) # Pacman go to the next level
173                 else:
174                     value = performAlphaBeta(depth, agentIndex+1, successor, alpha, beta)
175                     # Calculate ghosts's min value for this successor(next possible state) (We carry the current alpha and beta)
176                     # Ghost n will use ghost n-1's successors. After getting the last ghost's value, we go backtracking.
177                 if(agentIndex == 0 and value > beta): # Pacman : impossible to go this branch : cut
178                     return value
179                 if (agentIndex > 0 and value < alpha): # Ghost : impossible to go this branch : cut
180                     return value
181                 if (agentIndex == 0 and value > alpha): # Pacman : find value > alpha : replace it
182                     alpha = value
183                 if (agentIndex > 0 and value < beta): # Ghost : find value < beta : replace it
184                     beta = value
185                 valueList += [value]
186
187             if agentIndex == 0: # Pacman
188                 if(depth == 1): # Back to root : return action
189                     for i in range(len(valueList)):
190                         if (valueList[i] == max(valueList)):
191                             return todo[i]
192                 else:
193                     return max(valueList) # Not a root : return max value
194             elif agentIndex > 0: # Ghosts
195                 return min(valueList) # Ghosts : return min value
196         return performAlphaBeta(1, 0, gameState, -99999, 99999) # Go to the function with very small alpha, very big beta
197
198         # End your code (Part 2)
```

Part 3

```
199 class ExpectimaxAgent(MultiAgentSearchAgent):
200     """Your expectimax agent (Part 3)"""
201     def getAction(self, gameState):
202         """
203         Returns the expectimax action using self.depth and self.evaluationFunction
204         All ghosts should be modeled as choosing uniformly at random from their legal moves.
205         """
206         # Begin your code (Part 3)
207         def performExpectimax(depth, agentIndex, gameState):
208             if (gameState.isWin() or gameState.isLose() or depth > self.depth): # Terminal condition (win or lose or exceed depth)
209                 return self.evaluationFunction(gameState) # Get evaluation
210
211             value = [] # Store the values for this node
212             todo = gameState.getLegalActions(agentIndex) # Get all legal actions (a list) of an agent
213
214             for action in todo:
215                 successor = gameState.getNextState(agentIndex, action) # The gameState after taking the legal action
216                 if ((agentIndex+1) >= gameState.getNumAgents()): # When all agents are done
217                     value += [performExpectimax(depth+1, 0, successor)] # Pacman go to the next level
218                 else:
219                     value += [performExpectimax(depth, agentIndex+1, successor)]
220                 # Calculate ghosts's expected utilities for this successor(next possible state)
221                 # Ghost n will use ghost n-1's successors. After getting the last ghost's expected utilities, we go backtracking.
222             if agentIndex == 0: # Pacman
223                 if (depth == 1): # Back to root : return action
224                     for i in range(len(value)):
225                         if (value[i] == max(value)):
226                             return todo[i]
227                 else:
228                     return max(value) # Not a root : return max value
229             elif agentIndex > 0: # Ghosts
230                 s = sum(value)
231                 l = len(value)
232                 return float(s/l) # Ghosts : return expected utilities
233             return performExpectimax(1, 0, gameState) # Go to the function with agent 0 (pacman), depth 1
234         # End your code (Part 3)
```

Part 4

```
237 def betterEvaluationFunction(currentGameState):
238     """Your extreme ghost-hunting, pellet-nabbing, food-gobbling, unstoppable evaluation function (Part 4)."""
239     # Begin your code (Part 4)
240     newPos = currentGameState.getPacmanPosition() # Get pacman position data
241     newFood = currentGameState.getFood() # Get food position data
242     newCapsules = currentGameState.getCapsules() # Get capsule position data
243     newGhostStates = currentGameState.getGhostStates() # Get ghosts position data
244     #Set up the weight
245     INF = 1000000000000.0
246     WEIGHT_FOOD = 5.0
247     WEIGHT_CAP = 10.0
248     WEIGHT_GHOST = -10.0
249     WEIGHT_SCARED_GHOST = 200.0
250     adjust_score = currentGameState.getScore() # Adjust score is modified from base score
251
252     distancesToFoodList = [util.manhattanDistance(newPos, foodPos) for foodPos in newFood.asList()] # Calculate the distances to food
253     if len(distancesToFoodList) > 0:
254         adjust_score += ( WEIGHT_FOOD / min(distancesToFoodList) ) # Consider the closest food. The food is closer, the score is higher.
255
256     distancesToCapList = []
257     for ii in range(len(newCapsules)):
258         distancesToCapList.append(abs(newPos[0] - newCapsules[ii][0]) + abs(newPos[1] - newCapsules[ii][1])) #Calculate the distances between pacman and capsule
259     if len(distancesToCapList) > 0:
260         adjust_score += WEIGHT_CAP / min(distancesToCapList) # Consider the closest capsule
261
262     for ghost in newGhostStates:
263         distance = manhattanDistance(newPos, ghost.getPosition()) # Calculate the distances to ghost
264         if distance > 0:
265             if ghost.scaredTimer > 1:
266                 adjust_score += WEIGHT_SCARED_GHOST / distance # In scared time : the ghost is closer, the score is higher.
267             elif ghost.scaredTimer > 0:
268                 adjust_score += (WEIGHT_SCARED_GHOST - 50) / distance # Scared time is about to end : WEIGHT decreases
269             else:
270                 adjust_score += WEIGHT_GHOST / distance # Not in scared time : the ghost is closer, the score is lower. (WEIGHT_GHOST is negative)
271         else: # Distance <= 0 : pacman died
272             return -INF
273     return adjust_score
274     # End your code (Part 4)
```

Part II. Results & Analysis (5%):

```
Question part1
=====

*** PASS: test_cases\part1\0-eval-function-lose-states-1.test
*** PASS: test_cases\part1\0-eval-function-lose-states-2.test
*** PASS: test_cases\part1\0-eval-function-win-states-1.test
*** PASS: test_cases\part1\0-eval-function-win-states-2.test
*** PASS: test_cases\part1\0-lecture-6-tree.test
*** PASS: test_cases\part1\0-small-tree.test
*** PASS: test_cases\part1\1-1-minmax.test
*** PASS: test_cases\part1\1-2-minmax.test
*** PASS: test_cases\part1\1-3-minmax.test
*** PASS: test_cases\part1\1-4-minmax.test
*** PASS: test_cases\part1\1-5-minmax.test
*** PASS: test_cases\part1\1-6-minmax.test
*** PASS: test_cases\part1\1-7-minmax.test
*** PASS: test_cases\part1\1-8-minmax.test
*** PASS: test_cases\part1\2-1a-vary-depth.test
*** PASS: test_cases\part1\2-1b-vary-depth.test
*** PASS: test_cases\part1\2-2a-vary-depth.test
*** PASS: test_cases\part1\2-2b-vary-depth.test
*** PASS: test_cases\part1\2-3a-vary-depth.test
*** PASS: test_cases\part1\2-3b-vary-depth.test
*** PASS: test_cases\part1\2-4a-vary-depth.test
*** PASS: test_cases\part1\2-4b-vary-depth.test
*** PASS: test_cases\part1\2-one-ghost-3level.test
*** PASS: test_cases\part1\3-one-ghost-4level.test
*** PASS: test_cases\part1\4-two-ghosts-3level.test
*** PASS: test_cases\part1\5-two-ghosts-4level.test
*** PASS: test_cases\part1\6-tied-root.test
*** PASS: test_cases\part1\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part1\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part1\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores: 84.0
Win Rate: 0/1 (0.00)
Record: Loss

Question part2
=====

*** PASS: test_cases\part2\0-eval-function-lose-states-1.test
*** PASS: test_cases\part2\0-eval-function-lose-states-2.test
*** PASS: test_cases\part2\0-eval-function-win-states-1.test
*** PASS: test_cases\part2\0-eval-function-win-states-2.test
*** PASS: test_cases\part2\0-lecture-6-tree.test
*** PASS: test_cases\part2\0-small-tree.test
*** PASS: test_cases\part2\1-1-minmax.test
*** PASS: test_cases\part2\1-2-minmax.test
*** PASS: test_cases\part2\1-3-minmax.test
*** PASS: test_cases\part2\1-4-minmax.test
*** PASS: test_cases\part2\1-5-minmax.test
*** PASS: test_cases\part2\1-6-minmax.test
*** PASS: test_cases\part2\1-7-minmax.test
*** PASS: test_cases\part2\1-8-minmax.test
*** PASS: test_cases\part2\2-1a-vary-depth.test
*** PASS: test_cases\part2\2-1b-vary-depth.test
*** PASS: test_cases\part2\2-2a-vary-depth.test
*** PASS: test_cases\part2\2-2b-vary-depth.test
*** PASS: test_cases\part2\2-3a-vary-depth.test
*** PASS: test_cases\part2\2-3b-vary-depth.test
*** PASS: test_cases\part2\2-4a-vary-depth.test
*** PASS: test_cases\part2\2-4b-vary-depth.test
*** PASS: test_cases\part2\2-one-ghost-3level.test
*** PASS: test_cases\part2\3-one-ghost-4level.test
*** PASS: test_cases\part2\4-two-ghosts-3level.test
*** PASS: test_cases\part2\5-two-ghosts-4level.test
*** PASS: test_cases\part2\6-tied-root.test
*** PASS: test_cases\part2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part2\7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores: 84.0
Win Rate: 0/1 (0.00)
Record: Loss

Question part3
=====

*** PASS: test_cases\part3\0-eval-function-lose-states-1.test
*** PASS: test_cases\part3\0-eval-function-lose-states-2.test
*** PASS: test_cases\part3\0-eval-function-win-states-1.test
*** PASS: test_cases\part3\0-eval-function-win-states-2.test
*** PASS: test_cases\part3\0-expectimax1.test
*** PASS: test_cases\part3\1-expectimax2.test
*** PASS: test_cases\part3\2-one-ghost-3level.test
*** PASS: test_cases\part3\3-one-ghost-4level.test
*** PASS: test_cases\part3\4-two-ghosts-3level.test
*** PASS: test_cases\part3\5-two-ghosts-4level.test
*** PASS: test_cases\part3\6-1a-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1b-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1c-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part3\6-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part3\6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores: 84.0
Win Rate: 0/1 (0.00)
Record: Loss

Question part4
=====

Pacman emerges victorious! Score: 1367
Pacman emerges victorious! Score: 1368
Pacman emerges victorious! Score: 1312
Pacman emerges victorious! Score: 1316
Pacman emerges victorious! Score: 1312
Pacman emerges victorious! Score: 1351
Pacman emerges victorious! Score: 1209
Pacman emerges victorious! Score: 1344
Pacman emerges victorious! Score: 1334
Pacman emerges victorious! Score: 1355
Average Score: 1326.6
Scores: 1367.0, 1368.0, 1312.0, 1314.0, 1312.0, 1351.0, 1209.0, 1344.0, 1334.0, 1355.0
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
*** 1326.6 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 16:08:26

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

My evaluation function :

I consider the following 4 elements :

- Food : I consider the closest food. Eating the closest food first can save time, it's like a kind of greedy algorithm idea.
- Capsule : I also consider the capsules. Capsules have special effect, so its weight is higher than food.
- Ghost : Pacman have to avoid encountering ghosts when it's not scared time, so the WEIGHT_GHOST is negative.
- Scared time : In scared time, pacman eating ghosts can get much more score than eating food, so I assign higher weight for WEIGHT_SCARED_GHOST than WEIGHT_FOOD.

When the scared time is about to end, I decrease the WEIGHT_SCARED_GHOST to avoid encountering the ghost when time's up.