Introduction to Artificial Intelligence
Homework 3: Multi-Agent Search
0616076 朱彥勳
Part 1: Minimax Search (25%)
1. In line 107, the function 'getAction' is used to begin the code and return the final value after Minimax search. Also, in line 140, the function 'minimax' is defined as the process of Minimax search.
2. In line 141 and 142, when all of the agents are counted, or the game states is in a terminal state(either winning state or losing state), the function will return the final value called 'evaluationFunction(gameState)'.
3. From line 145 to 152, it is the main process of Minimax search. First, 'maxv' and 'minv' are defined as two best values(-∞ and +∞). Then, if the agent is a Pacman(agentIndex=0), it means that he wants to find the largest value of the children(steps) of him(Since the Pacman desires to win, he wants the most point). After finding all children(steps), he returns the largest value which he have found. On the contrary, if the agent is a ghost(agentIndex>=1), it means that he wants to find the smallest value of the children(steps) of him(Since the ghost desires Pacman not to win, he wants Pacman having the least point). After finding all children(steps), he returns the smallest value which he have found.
4. Back to line 134, Pacman moves first, so the first 'minimax' job is for Pacman to do.

```
102 class MinimaxAgent(MultiAgentSearchAgent):
103     """
104     Your minimax agent
105     """
106
107     def getAction(self, gameState):
108         """
109         Returns the minimax action from the current gameState using self.depth
110         and self.evaluationFunction.
111
112         Here are some method calls that might be useful when implementing minimax.
113
114         gameState.getLegalActions(agentIndex):
115         Returns a list of legal actions for an agent
116         agentIndex=0 means Pacman, ghosts are >= 1
117
118         gameState.getNextState(agentIndex, action):
119         Returns the child game state after an agent takes an action
120
121         gameState.getNumAgents():
122         Returns the total number of agents in the game
123
124         gameState.isWin():
125         Returns whether or not the game state is a winning state
126
127         gameState.isLose():
128         Returns whether or not the game state is a losing state
129         """
130         # Begin your code (Part 1)
131         maxv = ("max", -float("inf"))
132         agentIndex = 0
133         depth = 0
134         for step in gameState.getLegalActions(agentIndex):
135             cval = (step, self.minimax(gameState.getNextState(agentIndex, step), (depth + 1)%gameState.getNumAgents(), depth + 1))
136             if cval[1] > maxv[1]:
137                 maxv = cval
138         return maxv[0]
139
140     def minimax(self, gameState, agentIndex, depth):
141         if depth == self.depth * gameState.getNumAgents() or gameState.isLose() or gameState.isWin():
142             return self.evaluationFunction(gameState)
143         maxv = ("max", -float("inf"))
144         minv = ("min", float("inf"))
145         for step in gameState.getLegalActions(agentIndex):
146             cval = (step, self.minimax(gameState.getNextState(agentIndex, step), (depth + 1)%gameState.getNumAgents(), depth + 1))
147             if agentIndex == 0 and cval[1] > maxv[1]:
148                 maxv = cval
149             elif agentIndex >= 1 and cval[1] < minv[1]:
150                 minv = cval
151         if agentIndex == 0: return maxv[1]
152         else: return minv[1]
```

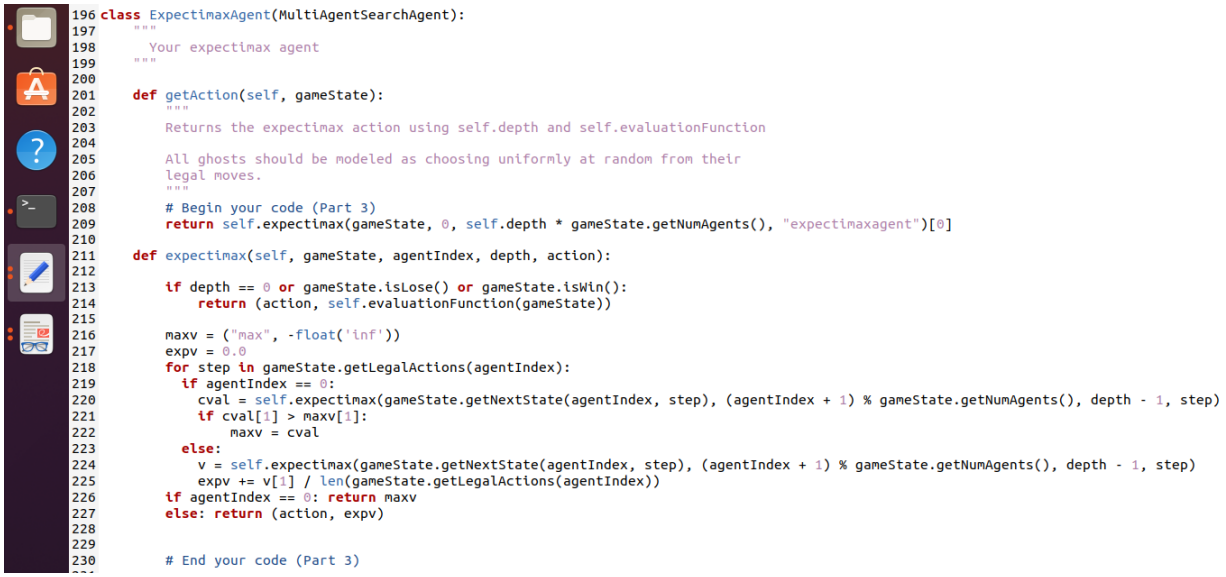
Part 2: Alpha-Beta Pruning (30%)

1. In line 160, the function 'getAction' is used to begin the code and return the final value after AlphaBeta search. Also, in line 177, the function 'alphabeta' is defined as the process of AlphaBeta search.
2. In line 178 and 179, when all of the agents are counted, or the game states is in a terminal state(either winning state or losing state), the function will return the final value called 'evaluationFunction(gameState)'.
3. From line 180 to 193, it is the main process of AlphaBeta search. First, 'maxv' and 'minv' are defined as two best values(-∞ and +∞). Then, if the agent is a Pacman(agentIndex=0), it means that he wants to find the largest value of the children(steps) of him(Since the Pacman desires to win, he wants the most point). However, the process would not be the same as Minimax Search. Actually, it is a revised version: α is the lower bound and β is the upper bound. In line 185, at the beginning of the loop, maxv will be the first searched child(step) value. Also, the value of α will be revised as maxv. Then, maxv will try to find a larger state value and also revise the value of α until this value exceed β(in line 186). Now, we notice that there is no overlap between α and β, so we can stop evaluating the children(steps) of the state. Finally, we can return the value of β of the state. If the agent is a ghost, we do the similary things, but change maxv to minv, α to β, and β to α.
4. Back to line 170, Pacman moves first, so the first 'alphabeta' job is for Pacman to do.

```
155 class AlphaBetaAgent(MultiAgentSearchAgent):
156     """
157     Your minimax agent with alpha-beta pruning
158     """
159
160     def getAction(self, gameState):
161         """
162         Returns the minimax action using self.depth and self.evaluationFunction
163         """
164         # Begin your code (Part 2)
165         maxv = ("max", -float("inf"))
166         agentIndex = 0
167         depth = 0
168         alpha = -float("inf")
169         beta = float("inf")
170         for step in gameState.getLegalActions(agentIndex):
171             cval = (step, self.alphabeta(gameState.getNextState(agentIndex, step), (depth + 1)%gameState.getNumAgents(), depth + 1, alpha, beta))
172             if cval[1] > maxv[1]: maxv = cval
173             if maxv[1] > beta: return maxv
174             elif maxv[1] > alpha: alpha = maxv[1]
175         return maxv[0]
176
177     def alphabeta(self, gameState, agentIndex, depth, alpha, beta):
178         if depth == self.depth * gameState.getNumAgents() or gameState.isLose() or gameState.isWin():
179             return self.evaluationFunction(gameState)
180         maxv = ("max", -float("inf"))
181         minv = ("min", float("inf"))
182         for step in gameState.getLegalActions(agentIndex):
183             cval = (step, self.alphabeta(gameState.getNextState(agentIndex, step), (depth + 1)%gameState.getNumAgents(), depth + 1, alpha, beta))
184             if agentIndex == 0:
185                 if cval[1] > maxv[1]: maxv = cval
186                 if maxv[1] > beta: return maxv[1]
187                 elif maxv[1] > alpha: alpha = maxv[1]
188             else:
189                 if cval[1] < minv[1]: minv = cval
190                 if minv[1] < alpha: return minv[1]
191                 elif minv[1] < beta: beta = minv[1]
192         if agentIndex == 0: return maxv[1]
193         else: return minv[1]
194         # End your code (Part 2)
```

Part 3: Expectimax Search (30%)

1. In line 201, the function 'getAction' is used to begin the code and return the final value after Expectimax search. Also, in line 211, the function 'expectimax' is defined as the process of AlphaBeta search.
2. In line 213 and 214, when all of the agents are counted, or the game states is in a terminal state(either winning state or losing state), the function will return the final value called 'evaluationFunction(gameState)'.
3. From line 216 to 227, it is the main process of Expectimax search. If the agent is a Pacman, it does the same thing as 'minimax search'. If the agent is a ghost, it needs to return the average of all the children(steps) which he has gone through.



```
196 class ExpectimaxAgent(MultiAgentSearchAgent):
197     """
198       Your expectimax agent
199     """
200
201     def getAction(self, gameState):
202         """
203         Returns the expectimax action using self.depth and self.evaluationFunction
204
205         All ghosts should be modeled as choosing uniformly at random from their
206         legal moves.
207         """
208         # Begin your code (Part 3)
209         return self.expectimax(gameState, 0, self.depth * gameState.getNumAgents(), "expectimaxagent")[0]
210
211     def expectimax(self, gameState, agentIndex, depth, action):
212
213         if depth == 0 or gameState.isLose() or gameState.isWin():
214             return (action, self.evaluationFunction(gameState))
215
216         maxv = ("max", -float('inf'))
217         expv = 0.0
218         for step in gameState.getLegalActions(agentIndex):
219           if agentIndex == 0:
220             cval = self.expectimax(gameState.getNextState(agentIndex, step), (agentIndex + 1) % gameState.getNumAgents(), depth - 1, step)
221             if cval[1] > maxv[1]:
222                 maxv = cval
223           else:
224             v = self.expectimax(gameState.getNextState(agentIndex, step), (agentIndex + 1) % gameState.getNumAgents(), depth - 1, step)
225             expv += v[1] / len(gameState.getLegalActions(agentIndex))
226         if agentIndex == 0: return maxv
227         else: return (action, expv)
228
229
230         # End your code (Part 3)
```

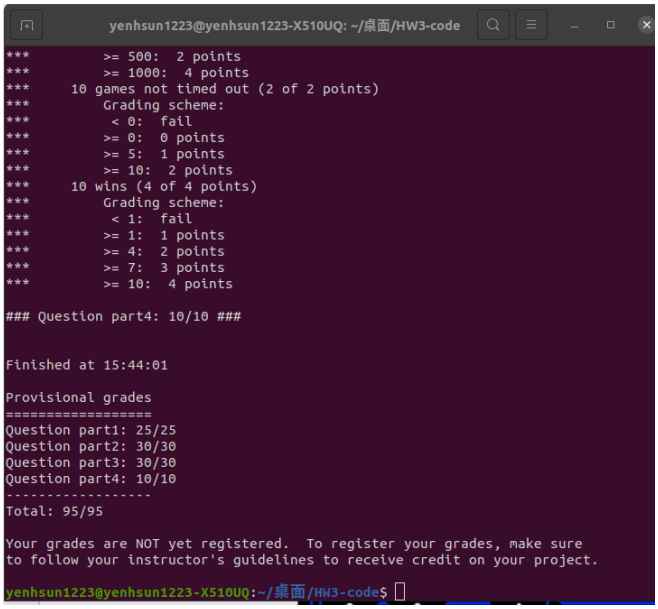
Part 4: Evaluation Function (Bonus) (10%)

1. In line 240 and 241, the different distances from all the food to Pacman are recorded into a list called 'distance', and in line 242, the least distance is founded.

2. In line 244, return the value of 'currentGameState.getScore()' minus the least distance.

```
232 def betterEvaluationFunction(currentGameState):
233     """
234     Your extreme evaluation function
235     """
236     # Begin your code (Part 4)
237     distance = []
238     min_distance = 0
239     t = 0
240     for i in currentGameState.getFood().asList():
241         distance.append(manhattanDistance(i, currentGameState.getPacmanPosition()))
242     if len(distance):
243         min_distance = min(distance)
244     return currentGameState.getScore() - min_distance
245     # End your code (Part 4)
```

Result:



Problem I met: Although I have known the use and the difference of these search, I did not know how to start this code and how much function I should provide in each class. Then, I found out that I only need two function: one is for initializing and the other is for main searching. However, what parameters that the function needed was another problem. Fortunately, I could follow pseudocode and start the code. After ending up the minimax search, others are similary to the first one. In part four, I did not know the exact difference between 'currentGameState.getScore()' and the correct value. Then, I tried to add up any distance or score from 'currentGameState.getScore()'. Finally, when I used 'currentGameState.getScore()' to minus the least distance from the food to the Pacman, it worked.