Part 1 – MinimaxAgent -- Description

1. Defined a recursive function minimax() in getAction().
2. getAction() will call minimax(state, agentIndex=0, cur_depth=0)

```
agentIndex, cur_depth = 0, 0
_, action = minmax(gameState, agentIndex , cur_depth)
return action
```

3. In minimax() :

① check if gameState is terminal states or not

by isWin() or sLose() or (depth == self.depth).

---

② next_agentIndex = (agentIndex +1) % num(agents), which is ranging from 0 to num(agents)-1 since Pacman has agentIndex = 0 and Ghosts have agentIndex > 0.

③ next_depth is increased based on rounds.

next_depth +=1 at Pacman's turns.

```
#---------------- terminal state ----------------
if gameState.isWin() or gameState.isLose() or cur_depth == self.depth :
    return self.evaluationFunction(gameState), None
#------------------ pre-defined -----------------
next_agentIndex = (agentIndex + 1) % gameState.getNumAgents()
next_depth = (cur_depth + 1) if (next_agentIndex == 0) else cur_depth
```

---

④ If agentIndex == 0 (Pacman's turn) :

Loop through all legal actions and return (max_value, max_action)

Else (Ghost's turn) :

Loop through all legal actions and return (min_value, min_action)

⑤ What we do for each action in the above loops is getting next_state by gameState.getNextState() and then getting next_state's value by calling minimax().

```
#-------------------- Pacman --------------------
if(agentIndex==0):
    # default value, action
    max_value = float("-inf")
    max_action = None
    # Loop through legal actions to get the maximal value
    # For each action, we can get a next_state and next_state's value
    # If next_state's value is the larger than other past states's value, save it
    for action in gameState.getLegalActions(agentIndex):
        next_state = gameState.getNextState(agentIndex, action)
        next_value, _ = minmax(next_state, next_agentIndex, cur_depth)
        if next_value > max_value:
            max_value = next_value
            max_action = action
    return max_value, max_action
#-------------------- Ghost --------------------
else:
    min_value = float("inf")
    min_action = None
    for action in gameState.getLegalActions(agentIndex):
        next_state = gameState.getNextState(agentIndex, action)
        next_value, _ = minmax(next_state, next_agentIndex ,next_depth)
        if next_value < min_value:
            min_value = next_value
            min_action = action
    return min_value, min_action
```

Part 1 – MinimaxAgent -- Code

```
class MinimaxAgent(MultiAgentSearchAgent):

    def getAction(self, gameState):
        # Begin your code (Part 1)
        def minmax(gameState, agentIndex, cur_depth):
            #---------------- terminal state ----------------
            if gameState.isWin() or gameState.isLose() or cur_depth == self.depth :
                return self.evaluationFunction(gameState), None
            #------------------ pre-defined ----------------
            next_agentIndex = (agentIndex + 1) % gameState.getNumAgents()
            next_depth = (cur_depth + 1) if (next_agentIndex == 0) else cur_depth
            #-------------------- Pacman --------------------
            if(agentIndex==0):
                # default value, action
                max_value = float("-inf")
                max_action = None
                # Loop through legal actions to get the maximal value
                # For each action, we can get a next_state and next_state's value
                # If next_state's value is the larger than other past states's value, save it
                for action in gameState.getLegalActions(agentIndex):
                    next_state = gameState.getNextState(agentIndex, action)
                    next_value, _ = minmax(next_state, next_agentIndex, cur_depth)
                    if next_value > max_value:
                        max_value = next_value
                        max_action = action
                return max_value, max_action
            #-------------------- Ghost --------------------
            else:
                min_value = float("inf")
                min_action = None
                for action in gameState.getLegalActions(agentIndex):
                    next_state = gameState.getNextState(agentIndex, action)
                    next_value, _ = minmax(next_state, next_agentIndex ,next_depth)
                    if next_value < min_value:
                        min_value = next_value
                        min_action = action
                return min_value, min_action
        #---------------------------
        agentIndex, cur_depth = 0, 0
        _, action = minmax(gameState, agentIndex , cur_depth)
        return action
        # End your code (Part 1)
```

Part2 – AlphaBetaAgent -- Description

1. There are 3 functions in class AlphaBataAgent. They are getAction(), max_value(), and min_value().
2. getAction() call max_value(state, index=0, depth=0, alpha=-inf, Beta=inf)

```
def getAction(self, gameState):
    agentIndex = 0
    cur_depth = 0
    _, action = self.max_value(gameState, agentIndex, cur_depth, float('-inf'),float('inf'))
    return action
```

3. In max_value() :

   ① Check terminal states.

   next_agentIndex and next_depth are mentioned in Part0.

```
if gameState.isWin() or gameState.isLose() or cur_depth == self.depth :
    return self.evaluationFunction(gameState), None

next_agentIndex = (agentIndex + 1) % gameState.getNumAgents()
next_depth = (cur_depth + 1) if (next_agentIndex == 0) else cur_depth
```

   ----

   ② Loop through all legal actions of current state. For each action:

   (1) get next_state by calling getNextState()

   (2) get next_state's value by calling max_value() if next_agentIndex == 0 ; otherwise call min_value().

   (3) After next_state's value being returned, update max_value, alpha.

   (4) If (max_value > Beta) : return immediately to prune the state tree.

```
# loop through actions to get the max_value
max_value = float("-inf")
max_action = None
for action in gameState.getLegalActions(agentIndex):
    next_state = gameState.getNextState(agentIndex, action)
    # if next_state is Pacman's turn, call max_value() to get next_state's value
    if(next_agentIndex==0):
        next_value, _ = self.max_value(next_state, next_agentIndex, next_depth, alpha, beta)
    # if next_state is Ghost's turn, call min_value() to get next_state's value
    else:
        next_value, _ = self.min_value(next_state, next_agentIndex, next_depth, alpha, beta)
    # update max_value
    if next_value > max_value:
        max_value = next_value
        max_action = action
    # update alpha
    alpha = max(alpha, max_value)
    # Pruning: if max_value > Beta
    if max_value > beta:
        return max_value, action
return max_value, max_action
```

4. In min_value() :

   ①② steps are same as max_value().

   ③Loop through all legal actions of current state. For each action:

   (1)(2) are same as max_value().

   (3) After next_state's value being returned, update min_value, beta.

   (4) If (min_value < alpha) : return immediately to prune the state tree.

```
    # update min_value
    if next_value < min_value:
        min_value = next_value
        min_action = action
    # update beta
    beta = min(beta, min_value)
    # Pruning: if min_value < alpha
    if min_value < alpha:
        return min_value, action
return min_value, min_action
```

Part 2 – AlphaBetaAgent -- Code

```python
class AlphaBetaAgent(MultiAgentSearchAgent):
    """
    Your minimax agent with alpha-beta pruning
    """
    def getAction(self, gameState):
        agentIndex = 0
        cur_depth = 0
        _, action = self.max_value(gameState, agentIndex, cur_depth, float('-inf'),float('inf'))
        return action

    def max_value(self, gameState, agentIndex, cur_depth, alpha, beta):
        # terminal states
        if gameState.isWin() or gameState.isLose() or cur_depth == self.depth :
            return self.evaluationFunction(gameState), None
        # pre-defined next_state's info
        next_agentIndex = (agentIndex + 1) % gameState.getNumAgents()
        next_depth = (cur_depth + 1) if (next_agentIndex == 0) else cur_depth
        # loop through actions to get the max_value
        max_value = float("-inf")
        max_action = None
        for action in gameState.getLegalActions(agentIndex):
            next_state = gameState.getNextState(agentIndex, action)
            # if next_state is Pacman's turn, call max_value() to get next_state's value
            if(next_agentIndex==0):
                next_value, _ = self.max_value(next_state, next_agentIndex, next_depth, alpha, beta)
            # if next_state is Ghost's turn, call min_value() to get next_state's value
            else:
                next_value, _ = self.min_value(next_state, next_agentIndex, next_depth, alpha, beta)
            # update max_value
            if next_value > max_value:
                max_value = next_value
                max_action = action
            # Pruning: if max_value > Beta
            alpha = max(alpha, max_value)
            if max_value > beta:
                return max_value, action
        return max_value, max_action

    def min_value(self, gameState, agentIndex, cur_depth, alpha, beta):
        if gameState.isWin() or gameState.isLose() or cur_depth == self.depth :
            return self.evaluationFunction(gameState), None

        next_agentIndex = (agentIndex + 1) % gameState.getNumAgents()
        next_depth = (cur_depth + 1) if (next_agentIndex == 0) else cur_depth
        # loop through actions to get the min_value
        min_value = float("inf")
        min_action = None
        for action in gameState.getLegalActions(agentIndex):
            next_state = gameState.getNextState(agentIndex, action)
            # if next_state is Pacman's turn, call max_value() to get next_state's value
            if(next_agentIndex==0):
                next_value, _ = self.max_value(next_state, next_agentIndex, next_depth, alpha, beta)
            # if next_state is Ghost's turn, call min_value() to get next_state's value
            else:
                next_value, _ = self.min_value(next_state, next_agentIndex, next_depth, alpha, beta)
            # update min_value
            if next_value < min_value:
                min_value = next_value
                min_action = action
            # Pruning: if min_value < alpha
            beta = min(beta, min_value)
            if min_value < alpha:
                return min_value, action
        return min_value, min_action
```

Part3 – ExpectimaxAgent -- Description

1. Define a recursive function expectimax() in getAction().
2. getAction() will call expectimax(state, agentIndex=0, cur_depth=0)

```
agentIndex = 0
cur_depth = 0
_, action = self.expectimax(gameState, agentIndex, cur_depth)
return action
```

3. In expectimax() :

① check terminal states, predefine next_agentIndex and next_depth as MinimaxAgent and AlphaBeataAgent.

```
#--------------- terminal state ---------------
if game_state.isWin() or game_state.isLose() or depth == self.depth :
    return self.evaluationFunction(game_state), None
#----------------- pre-defined -----------------
next_agentIndex = (agentIndex + 1) % game_state.getNumAgents()
next_depth = (depth + 1) if (next_agentIndex == 0) else depth
LegalActions = game_state.getLegalActions(agentIndex)
```

----

② If agentIndex == 0 (Pacman's turn) :

Loop through all legal actions and return (max_value, max_action)

Else (Ghost's turn) :

Loop through all legal actions and return (expected_value, expected _action)

③ The loop in Pacman's part is same as what applied in MinimaxAgent's.

④ The loop in Ghost's part return (expected_value, expected _action), where expected value = $\sum_{i=1}^{num(next\_states)} \mathrm{p} * \mathrm{value}_i$ with p = 1/num(next_states).

```
#-------------------- Pacman --------------------
if agentIndex == 0:
    max_value = float("-inf")
    max_action = None

    for action in LegalActions:
        next_state = game_state.getNextState(agentIndex, action)
        next_value, _ = self.expectimax(next_state, next_agentIndex, next_depth)
        if next_value > max_value:
            max_value = next_value
            max_action = action
    return max_value, max_action
#-------------------- Ghost --------------------
else:
    expected_value = 0
    expected_action = None
    probability = 1.0 / len(LegalActions)

    for action in LegalActions:
        next_state = game_state.getNextState(agentIndex, action)
        next_value, _ = self.expectimax(next_state, next_agentIndex, next_depth)
        expected_value += probability * next_value
    return expected_value, expected_action
```

Part3 – ExpectimaxAgent -- Code

```
class ExpectimaxAgent(MultiAgentSearchAgent):
    """
      Your expectimax agent
    """
    def getAction(self, gameState):
        agentIndex = 0
        cur_depth = 0
        _, action = self.expectimax(gameState, agentIndex, cur_depth)
        return action

    def expectimax(self, game_state, agentIndex, depth):
        #---------------- terminal state ----------------
        if game_state.isWin() or game_state.isLose() or depth == self.depth :
            return self.evaluationFunction(game_state), None
        #------------------ pre-defined ------------------
        next_agentIndex = (agentIndex + 1) % game_state.getNumAgents()
        next_depth = (depth + 1) if (next_agentIndex == 0) else depth
        LegalActions = game_state.getLegalActions(agentIndex)
        #-------------------- Pacman --------------------
        if agentIndex == 0:
            max_value = float("-inf")
            max_action = None

            for action in LegalActions:
                next_state = game_state.getNextState(agentIndex, action)
                next_value, _ = self.expectimax(next_state, next_agentIndex, next_depth)
                if next_value > max_value:
                    max_value = next_value
                    max_action = action
            return max_value, max_action
        #-------------------- Ghost --------------------
        else:
            expected_value = 0
            expected_action = None
            probability = 1.0 / len(LegalActions)

            for action in LegalActions:
                next_state = game_state.getNextState(agentIndex, action)
                next_value, _ = self.expectimax(next_state, next_agentIndex, next_depth)
                expected_value += probability * next_value
            return expected_value, expected_action
```

Part4 – betterEvaluationFunction -- Description & Code

1. Assume 2 kinds of factors : min_food_dist and game_score
2. factor1 -- min_food_dist :
   ① look for the minimal distance from pacman to food.
   ② pacman's position is rechieved by getPacmanPosition(), and food's position are rechieved by getFood().
   ③ The function used for caculateing distance between Pacman and Food is manhattanDistance(), which is imported at default.
3. factor2 --game_score is rechieved by getScore()
4. The weight for factors is set by try and error.
5. Since smaller min_food_dist is better, factor1 is set by 1.0/(1.0+min_food_dist). Add min_food_dist by 1 to avoid division by zero.
6. The final evaluated value = factor1*weight1 + factor2*weight2.

```python
def betterEvaluationFunction(currentGameState):
    """
    Your extreme evaluation function
    """
    # 2 kinds of possible factors
    min_food_dist = float('inf')
    game_score = currentGameState.getScore()

    # the first factor -- min_food_dist
    pacman_pos = currentGameState.getPacmanPosition()
    for food_position in currentGameState.getFood().asList():
        pac_food_dist = manhattanDistance(pacman_pos, food_position)
        if(pac_food_dist < min_food_dist):
            min_food_dist = pac_food_dist

    # modify weights manually
    min_food_dist_weight = 1
    game_score_weight = 1

    # Final_value = Sum(factor_i * weight_i)
    Final_value = 1.0/(1.0+min_food_dist) * min_food_dist_weight  \
                + game_score * game_score_weight

    return Final_value
```

Part5 – problems you meet and how you solve them

The biggest problem is the definition of provided function. Though there's a short usage description at the top of MinimaxAgent code, what objects those function will return are still confusing. I believe we should go to pacman.py and utils.py frequently to look for the answers. By the way, the type hint is very helpful in exploring default function and class members(X

~~Praise for type hint~~