Homework 3: Multi-Agent Search

Please keep the title of each section and delete examples.

Part I. Implementation (5%):

- Part 1:

```
    # Begin your code (Part 1)
    # call function "selectBest"
    result = self.selectBest(gameState, 0, 0)
    return result[1]
"""
selectBest(state):
    no-actions: return score
    pacman: return max-value(state)
    ghost:  return min_value(state)
"""
def selectBest(self, gameState, index, depth):
    if len(gameState.getLegalActions(index))==0 or depth==self.depth:
        return gameState.getScore(), ""
    if index==0:    # pacman index = 0
        return self.max_value(gameState, index, depth)
    else:           # ghost index ≥ 1
        return self.min_value(gameState, index, depth)
"""
max_value(state):
    get all legal actions and save into "actions"
    for each a in actions:
        let successor be the child state
        get the largest value of selectBest(successor)
    return max_value and its action
"""
def max_value(self, gameState, index, depth):
    actions = gameState.getLegalActions(index)
    v = float('-inf')
    max_a = ""
    for a in actions:
        successor = gameState.getNextState(index, a)
        successor_index = index + 1
        successor_depth = depth
        if successor_index==gameState.getNumAgents():
            successor_index = 0
            successor_depth += 1
        cur_v = self.selectBest(successor, successor_index, successor_depth)[0]
        if cur_v > v:
            v = cur_v
            max_a = a
    return v, max_a
"""
```

```
"""
min_value(state):
    get all legal actions and save into "actions"
    for each a in actions:
        let successor be the child state
        get the least value of selectBest(successor)
    return min_value and its action
"""
def min_value(self, gameState, index, depth):
    actions = gameState.getLegalActions(index)
    v = float('inf')
    min_a = ""
    for a in actions:
        successor = gameState.getNextState(index, a)
        successor_index = index + 1
        successor_depth = depth
        if successor_index==gameState.getNumAgents():
            successor_index = 0
            successor_depth += 1
        cur_v = self.selectBest(successor, successor_index, successor_depth)[0]
        if cur_v < v:
            v = cur_v
            max_a = a

    return v, min_a
    # End your code (Part 1)
```

- Part 2:

```
    # Begin your code (Part 2)
    # call function "selectBest"
    result = self.selectBest(gameState, 0, 0, float('-inf'), float('inf'))
    return result[1]
"""
selectBest(state):
    no-actions: return score
    pacman: return max-value(state)
    ghost:  return min_value(state)
"""
def selectBest(self, gameState, index, depth, alpha, beta):
    if len(gameState.getLegalActions(index))==0 or depth==self.depth:
        return gameState.getScore(), ""
    if index==0:    # pacman index = 0
        return self.max_value(gameState, index, depth, alpha, beta)
    else:           # ghost index >= 1
        return self.min_value(gameState, index, depth, alpha, beta)
"""
max_value(state):
    get all legal actions and save into "actions"
    for each a in actions:
        let successor be the child state
        get the largest value of selectBest(successor)
        alpha = max(alpha, value)
        if value > beta: then cut directly and return because it won't use the beta side anymore.
    return max_value and its action
"""
def max_value(self, gameState, index, depth, alpha, beta):
    actions = gameState.getLegalActions(index)
    v = float('-inf')
    max_a = ""
    for a in actions:
        successor = gameState.getNextState(index, a)
        successor_index = index + 1
        successor_depth = depth
        if successor_index==gameState.getNumAgents():
            successor_index = 0
            successor_depth += 1
        cur_v= self.selectBest(successor, successor_index, successor_depth, alpha, beta)[0]
        if cur_v > v:
            v = cur_v
            max_a = a
        alpha = max(alpha, v)
        if v > beta:
            return v, max_a
    return v, max_a
"""
```

```
    """
    min_value(state):
        get all legal actions and save into "actions"
        for each a in actions:
            let successor be the child state
            get the least value of selectBest(successor)
            beta = min(alpha, value)
            if value > alpha: then cut directly and return because it won't use the alpha side anymore.
        return min_value and its action
    """
    def min_value(self, gameState, index, depth, alpha, beta):
        actions = gameState.getLegalActions(index)
        v = float('inf')
        min_a = ""
        for a in actions:
            successor = gameState.getNextState(index, a)
            successor_index = index + 1
            successor_depth = depth
            if successor_index==gameState.getNumAgents():
                successor_index = 0
                successor_depth += 1
            cur_v= self.selectBest(successor, successor_index, successor_depth, alpha, beta)[0]
            if cur_v < v:
                v = cur_v
                min_a = a
            beta = min(beta, v)
            if v < alpha:
                return v, min_a
        return v, min_a
        # End your code (Part 2)
```

- Part 3:

```
        # Begin your code (Part 3)
        # call function "selectBest"
        result = self.selectBest(gameState, 0, 0)
        return result[1]
    """
    selectBest(state):
        no-actions: return score
        pacman: return max-value(state)
        ghost:  return expected_value(state)    # ghost will move stochastically
    """
    def selectBest(self, gameState, index, depth):
        if len(gameState.getLegalActions(index))==0 or depth==self.depth:
            return self.evaluationFunction(gameState), ""
        if index==0:    # pacman index = 0
            return self.max_value(gameState, index, depth)
        else:           # expectimax-ghost index ≥ 1
            return self.expected_value(gameState, index, depth)

    """
    same as Part 1's max_value
    """
    def max_value(self, gameState, index, depth):
        actions = gameState.getLegalActions(index)
        v = float('-inf')
        max_a = ""
        for a in actions:
            successor = gameState.getNextState(index, a)
            successor_index = index + 1
            successor_depth = depth
            if successor_index==gameState.getNumAgents():
                successor_index = 0
                successor_depth += 1
            cur_v = self.selectBest(successor, successor_index, successor_depth)[0]
            if cur_v > v:
                v = cur_v
                max_a = a
        return v, max_a
    """
```

```
    """
    expected_value(state):
        get all legal actions and save into "actions"
        calculate the probability "pr" = 1/N
        for each a in actions:
            let successor be the child state
            get the least value of selectBest(successor)
            add all the expected utilities (pr*value)
        return expected_value and its action
    """
    def expected_value(self, gameState, index, depth):
        actions = gameState.getLegalActions(index)
        v = 0
        pr = 1.0/len(actions)
        for a in actions:
            successor = gameState.getNextState(index, a)
            successor_index = index + 1
            successor_depth = depth
            if successor_index==gameState.getNumAgents():
                successor_index = 0
                successor_depth += 1
            cur_v = self.selectBest(successor, successor_index, successor_depth)[0]
            v += pr*cur_v
        return v, ""
        # End your code (Part 3)
```

- Part 4:

```
# Begin your code (Part 4)
"""
take these information to estimate the evalution:
1. 1.0/closest_food: if value is low, which means closest_food is large, which means there is no food nearby,
                     should have positive but small evalution.
                     if the ghost is too close to pac-man, it will unconditionally set to a high number.
                     if the ghost is too far away from pac-man, set higher number.
2. food_cnt: less food left, more probability win
3. capsule_cnt:getCapsules()
4. cur_score: the more score I got, the more probability I win.
At last, calculate all w*f to evalution and return it.
"""
pacman_pos = currentGameState.getPacmanPosition()
ghost_pos = currentGameState.getGhostPositions()
GhostDistance = [manhattanDistance(pacman_pos, ghost) for ghost in ghost_pos]

foods = currentGameState.getFood().asList()
food_cnt = len(foods)
FoodDistance = [manhattanDistance(pacman_pos, food) for food in foods]

capsule_cnt = len(currentGameState.getCapsules())
closest_food = 1
cur_score = currentGameState.getScore()
if food_cnt:
    closest_food = min(FoodDistance)

for gd in GhostDistance:
    if gd > 6:
        closest_food = 100
    if gd < 2:
        closest_food = 100

f=[1.0/closest_food, food_cnt, capsule_cnt, cur_score]
w=[1, -10, -1, 10]
evaluation = sum([fi*wi for fi, wi in zip(f, w)])
return evaluation
# End your code (Part 4)
```

Part II. Results & Analysis (5%):