# Homework 3: Multi-Agent Search

## Implementation (5%)

- Please screenshot your code snippets of Part 1 ~ Part 4, and explain your implementation.

### Part 1

```
# Begin your code (Part 1)
#raise NotImplementedError("To be implemented")

num_of_agents = gameState.getNumAgents()
def minimax(gameState,depth,agentIndex):
    if agentIndex >= num_of_agents:
        agentIndex = agentIndex - num_of_agents
    if (depth == 0 and agentIndex == 0) or gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)

    ret = 0
    if depth == self.depth:
        MAX = -2e20
        for pacman_action in gameState.getLegalActions(agentIndex):
            tmp = minimax(gameState.getNextState(agentIndex,pacman_action),depth-1,agentIndex+1)
            if MAX < tmp:
                MAX = tmp
                ret = pacman_action
        return ret

    if agentIndex == 0:
        MAX = -2e20
        for pacman_action in gameState.getLegalActions(agentIndex):
            MAX = max(MAX,minimax(gameState.getNextState(agentIndex,pacman_action),depth-1,agentIndex+1))
        return MAX
    else:
        MIN = 2e20
        for ghost_action in gameState.getLegalActions(agentIndex):
            MIN = min(MIN,minimax(gameState.getNextState(agentIndex,ghost_action),depth,agentIndex+1))
        return MIN

return minimax(gameState,self.depth,0)
# End your code (Part 1)
```

The program uses recursion to calculate the answers of the evaluated value and return it, expect of the first round since the get Action function requires to return the action.

If it is ghost turns, the program will calculate the least value it can achieve. On the contrary, the program consider the maximum value it can achieve in the pacman turn.

### Part 2

```python
# Begin your code (Part 2)
num_of_agents = gameState.getNumAgents()
def alpha_beta_pruning(gameState,depth,agentIndex,alpha,beta):
    if agentIndex >= num_of_agents:
        agentIndex = agentIndex - num_of_agents
    if (depth == 0 and agentIndex == 0) or gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)

    ret = 0
    eval_alpha = alpha
    eval_beta = beta
    if depth == self.depth:
        MAX = -2e20
        for pacman_action in gameState.getLegalActions(agentIndex):
            tmp = alpha_beta_pruning(gameState.getNextState(agentIndex,pacman_action),depth-1,agentIndex+1,eval_alpha,eval_beta)
            eval_alpha = max(eval_alpha,tmp)
            if MAX < tmp:
                MAX = tmp
                ret = pacman_action
        return ret

    if agentIndex == 0:
        MAX = -2e20
        eval_MAX = -2e20
        for pacman_action in gameState.getLegalActions(agentIndex):
            tmp = alpha_beta_pruning(gameState.getNextState(agentIndex,pacman_action),depth-1,agentIndex+1,eval_alpha,eval_beta)
            MAX = max(MAX,tmp)
            eval_alpha = max(eval_alpha,tmp)
            if MAX > eval_beta:
                return MAX
        return MAX
    else:
        MIN = 2e20
        for ghost_action in gameState.getLegalActions(agentIndex):
            if agentIndex == 1:
                tmp = alpha_beta_pruning(gameState.getNextState(agentIndex,ghost_action),depth,agentIndex+1,eval_alpha,eval_beta)
            else:
                tmp = alpha_beta_pruning(gameState.getNextState(agentIndex,ghost_action),depth,agentIndex+1,eval_alpha,eval_beta)
            eval_beta = min(eval_beta,tmp)
            MIN = min(MIN,tmp)
            if MIN < eval_alpha:
                return MIN
        return MIN

return alpha_beta_pruning(gameState,self.depth,0,-2e20,2e20)
# End your code (Part 2)
```

It is closed to Minimax method, but the program prune the branch that it is not necessary to consider. Same as the implementation of Minimax, the program consider two cases.

The first round must return the action, and the others return the evaluated value. Then the program consider the pacman turns and the ghost turns. The outline of the implementation is similar to the pseudo code on the homework requirement pdf.

## Part 3

```
# Begin your code (Part 3)
#raise NotImplementedError("To be implemented")
num_of_agents = gameState.getNumAgents()
def expectimax(gameState,depth,agentIndex):
    if agentIndex >= num_of_agents:
        agentIndex = agentIndex - num_of_agents
    if (depth == 0 and agentIndex == 0) or gameState.isWin() or gameState.isLose():
        return self.evaluationFunction(gameState)

    ret = 0
    if depth == self.depth:
        MAX = -2e20
        for pacman_action in gameState.getLegalActions(agentIndex):
            tmp = expectimax(gameState.getNextState(agentIndex,pacman_action),depth-1,agentIndex+1)
            if MAX < tmp:
                MAX = tmp
                ret = pacman_action
        return ret

    if agentIndex == 0:
        MAX = -2e20
        for pacman_action in gameState.getLegalActions(agentIndex):
            MAX = max(MAX,expectimax(gameState.getNextState(agentIndex,pacman_action),depth-1,agentIndex+1))
        return MAX
    else:
        MIN = 0
        expect = 1/len(gameState.getLegalActions(agentIndex))
        for ghost_action in gameState.getLegalActions(agentIndex):
            MIN = MIN + expectimax(gameState.getNextState(agentIndex,ghost_action),depth,agentIndex+1)*expect
        return MIN

return expectimax(gameState,self.depth,0)
# End your code (Part 3)
```

Instead of pruning the branch, this method consider expected value of each move. Since the ghosts walk randomly, it is not necessary to use alpha beta pruning.

## Part 4

```
# Begin your code (Part 4)
if currentGameState.isLose():
    return -500000000

GhostStates = currentGameState.getGhostStates()
PacmanPosition = currentGameState.getPacmanPosition()
minGhostDistance = min([manhattanDistance(PacmanPosition, state.getPosition()) for state in GhostStates])
score = minGhostDistance * 0.05 + 1000

Foods = currentGameState.getFood()
minFoodDistance = Foods.height + Foods.width
capsule = currentGameState.getCapsules()

eat = 0
for x in range(0,Foods.width):
    for y in range(0,Foods.height):
        if  Foods[x][y]:
            minFoodDistance = min(minFoodDistance,manhattanDistance(PacmanPosition,[x,y]))
            eat = eat + 1

score = score - minFoodDistance - (Foods.height+Foods.width) * eat * 50 - len(capsule) * (Foods.height+Foods.width) * 500 + currentGameState.getScore() * 0.02
return score
# End your code (Part 4)
```

This evaluation function consider five scores.

First, the minimum manhattan distance between the pacman and the ghost. The program expect that far from the ghost can gain more scores. But the program didn’t consider it as an important parameter, the value has the small weight.

Second, the program considers the the minimun manhattan distance between food and pacman. This value has the normal weight since we hope it close to the food and shorten the

time to end the game.

Third, the program considers the number of food in the game. It got the large weight since we hope it can eat those food.

Fourth, the number of capsule. The program encourage the pacman to eat the capsule when it close to capsule. The reason is obvious that the capsule give the invincible time.

Fifth, the current score. We hopes that the pacman will tend to end the game faster with higher score. However, we don’t give it the high weight because it may do something strange if we give it high weight.

# Results & Analysis (5%)

## Autograder

The evaluation function work great. It get the high score in the test.

```
Question part4
==============

Pacman emerges victorious! Score: 1175
Pacman emerges victorious! Score: 978
Pacman emerges victorious! Score: 1174
Pacman emerges victorious! Score: 1166
Pacman emerges victorious! Score: 1171
Pacman emerges victorious! Score: 1364
Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1169
Pacman emerges victorious! Score: 1070
Average Score: 1201.3
Scores:        1175.0, 978.0, 1174.0, 1166.0, 1171.0, 1364.0, 1373.0, 1373.0, 1169.0, 1070.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

## AlphaBetaAgent trappedClassic depth=3 with better evaluation

It work as good as expectimax with default evaluation. It prove that my evaluation function work great.

```
Average Score: 56.36
Scores:        -502.0, -502.0, -502.0, 532.0, -502.0, 532.0, 532.0, 532.0, 532.0, -502.0, -502.0, 532.0, -502
.0, 532.0, 532.0, 532.0, 532.0, 532.0, -502.0, 532.0, -502.0, 532.0, 532.0, -502.0, 532.0, -502.0, 532.0, 532
.0, 532.0, 532.0, 532.0, 532.0, 532.0, -502.0, 532.0, -502.0, -502.0, 532.0, -502.0, -502.0, 532.0, 532.0, 53
2.0, 532.0, 532.0, -502.0, -502.0, 532.0, 532.0, -502.0, -502.0, 532.0, -502.0, -502.0, 532.0, -502.0, -502.0
, 532.0, -502.0, -502.0, -502.0, -502.0, 532.0, 532.0, -502.0, -502.0, -502.0, -502.0, 532.0, -502.0, 532.0,
-502.0, 532.0, 532.0, 532.0, -502.0, -502.0, -502.0, -502.0, -502.0, 532.0, -502.0, 532.0, 532.0, 532.0, 532.
0, -502.0, 532.0, 532.0, -502.0, 532.0, 532.0, -502.0, -502.0, 532.0, 532.0, -502.0, 532.0, -502.0, 532.0
Win Rate:      54/100 (0.54)
Record:        Loss, Loss, Loss, Win, Loss, Win, Win, Win, Win, Loss, Loss, Win, Loss, Win, Win, Win, Win, Wi
n, Loss, Win, Loss, Win, Win, Loss, Win, Loss, Win, Win, Win, Win, Win, Win, Win, Loss, Win, Loss, Loss, Win,
 Loss, Loss, Win, Win, Win, Win, Win, Loss, Loss, Win, Win, Loss, Loss, Win, Loss, Loss, Win, Loss, Loss, Win
, Loss, Loss, Loss, Loss, Win, Win, Loss, Loss, Loss, Loss, Win, Loss, Win, Loss, Win, Win, Win, Loss, Loss,
Loss, Loss, Loss, Win, Loss, Win, Win, Win, Win, Loss, Win, Win, Loss, Win, Win, Loss, Loss, Win, Win, Loss,
Win, Loss, Win
```

## MinimaxAgent smallClassic depth=3 with better evaluation

It has great result, but it looks a little bit lag in GUI mode.

What if we consider depth = 2 ?

```
Pacman emerges victorious! Score: 1373
Pacman emerges victorious! Score: 1579
Pacman emerges victorious! Score: 1374
Pacman emerges victorious! Score: 1175
Pacman emerges victorious! Score: 1572
Pacman emerges victorious! Score: 1141
Pacman emerges victorious! Score: 1776
Pacman emerges victorious! Score: 1176
Pacman emerges victorious! Score: 1568
Pacman emerges victorious! Score: 1379
Average Score: 1411.3
Scores:        1373.0, 1579.0, 1374.0, 1175.0, 1572.0, 1141.0, 1776.0, 1176.0, 1568.0, 1379.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

## MinimaxAgent smallClassic depth=2 with better evaluation

In GUI mode, it works smoothly. And it also win 10 games.

```
Pacman emerges victorious! Score: 1739
Pacman emerges victorious! Score: 1537
Pacman emerges victorious! Score: 1544
Pacman emerges victorious! Score: 1133
Pacman emerges victorious! Score: 1369
Pacman emerges victorious! Score: 1160
Pacman emerges victorious! Score: 874
Pacman emerges victorious! Score: 973
Pacman emerges victorious! Score: 1163
Pacman emerges victorious! Score: 1372
Average Score: 1286.4
Scores:        1739.0, 1537.0, 1544.0, 1133.0, 1369.0, 1160.0, 874.0, 973.0, 1163.0, 1372.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```

## ExpectimaxAgent smallClassic depth=2 with better evaluation

Great score. we can also discover that the expectimax works better than minimax agent. And the reason is obvious that the ghost move randomly.

```
Pacman emerges victorious! Score: 1356
Pacman emerges victorious! Score: 1170
Pacman emerges victorious! Score: 1556
Pacman emerges victorious! Score: 1566
Pacman emerges victorious! Score: 1233
Pacman emerges victorious! Score: 1231
Pacman emerges victorious! Score: 1169
Pacman emerges victorious! Score: 1479
Pacman emerges victorious! Score: 1721
Pacman emerges victorious! Score: 958
Average Score: 1343.9
Scores:        1356.0, 1170.0, 1556.0, 1566.0, 1233.0, 1231.0, 1169.0, 1479.0, 1721.0, 958.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
```