# HW3 Report

109550206 陳品劭 Self link

## Part I. Implementation

### Part 1

```
# Begin your code (Part 1)
    #raise NotImplementedError("To be implemented")
    """
        1. Call maxValue function
        2. Action compute by maxValue
    """
    _, nextAction = self.maxValue(gameState, 0, 0);
    return nextAction
def terminalState(self, gameState, agentIndex, curDepth):
    """
        1. return this game is finish or not and the depth is reached or not
    """
    if gameState.isWin() or gameState.isLose() or curDepth >= self.depth:
        return True
    return False
def nextValue(self, gameState, agentIndex, curDepth, nextAgent, legalAction):
    """
        1. For given state, agent, action call next function (max or min)
            - pacman is max player
            - ghosts are min player
    """
    nextState = gameState.getNextState(agentIndex, legalAction)
    legalValue = float()
    if nextAgent == 0:
        legalValue, _ = self.maxValue(nextState, nextAgent, curDepth + 1)
    else:
        legalValue = self.minValue(nextState, nextAgent, curDepth)
    return legalValue
def maxValue(self, gameState, agentIndex, curDepth):
    """
        1. If game is finish return the evaluation of that state
        2. Get next agent ID
        3. Create scores list and get all legal action for this agent
        4. Get all values of every legalAction in next layer by call nextValue and append that
value to scores
        5. Select one of maximun and return value and action
    """
    if( self.terminalState(gameState, agentIndex, curDepth) ):
        return self.evaluationFunction(gameState), None
    nextAgent = (agentIndex + 1) % gameState.getNumAgents()
    scores = list()
    legalMoves = gameState.getLegalActions(agentIndex)
    for legalAction in legalMoves:
        legalValue = self.nextValue(gameState, agentIndex, curDepth, nextAgent, legalAction)
        scores.append(legalValue)
    bestScore = max(scores)
    bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
    chosenIndex = random.choice(bestIndices)
    return bestScore, legalMoves[chosenIndex]
def minValue(self, gameState, agentIndex, curDepth):
```

```
    """
        1. If game is finish return the evaluation of that state
        2. Get next agent ID
        3. Create a float variable to load min value
        4. Get all values of every legalAction in next layer by call nextValue and select min one
        5. Return min value
    """
    if( self.terminalState(gameState, agentIndex, curDepth) ):
        return self.evaluationFunction(gameState)
    nextAgent = (agentIndex + 1) % gameState.getNumAgents()
    bestScore = float("inf")
    for legalAction in gameState.getLegalActions(agentIndex):
        legalValue = self.nextValue(gameState, agentIndex, curDepth, nextAgent, legalAction)
        bestScore = min(bestScore, legalValue)
    return bestScore
    # End your code (Part 1)
```

## Part 2

```
    # Begin your code (Part 2)
    #raise NotImplementedError("To be implemented")
    """
        1. Almost same with part 1
        2. We need give two value to function: alpha (load maximun), beta (load minimun)
    """
    _, nextAction = self.maxValue(gameState, 0, 0, float("-inf"), float("inf"));
    return nextAction
def terminalState(self, gameState, agentIndex, curDepth):
    """
        1. Same with part 1
    """
    if gameState.isWin() or gameState.isLose() or curDepth >= self.depth:
        return True
    return False
def nextValue(self, gameState, agentIndex, curDepth, nextAgent, legalAction, alpha, beta):
    """
        1. Almost same with part 1
        2. Let alpha and beta can be got by next layer
    """
    nextState = gameState.getNextState(agentIndex, legalAction)
    legalValue = float()
    if nextAgent == 0:
        legalValue, _ = self.maxValue(nextState, nextAgent, curDepth + 1, alpha, beta)
    else:
        legalValue = self.minValue(nextState, nextAgent, curDepth, alpha, beta)
    return legalValue
def maxValue(self, gameState, agentIndex, curDepth, alpha, beta):
    """
        1. Almost same with part 1
        2. When legalValue is greater than beta then return that value and action
        3. Load max value for alpha for every legalAction
    """
    if( self.terminalState(gameState, agentIndex, curDepth) ):
        return self.evaluationFunction(gameState), None
    nextAgent = (agentIndex + 1) % gameState.getNumAgents()
    scores = list()
    legalMoves = gameState.getLegalActions(agentIndex)
    for legalAction in legalMoves:
        legalValue = self.nextValue(gameState, agentIndex, curDepth, nextAgent, legalAction,
alpha, beta)
        scores.append(legalValue)
```

```
        if legalValue > beta:
            return legalValue, legalAction
        alpha = max(alpha, legalValue)
    bestScore = max(scores)
    bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
    chosenIndex = random.choice(bestIndices)
    return bestScore, legalMoves[chosenIndex]
def minValue(self, gameState, agentIndex, curDepth, alpha, beta):
    """
        1. Almost same with part 1
        2. When legalValue is fewer than alpha then return that value and action
        3. Load min value for beta for every legalAction
    """
    if( self.terminalState(gameState, agentIndex, curDepth) ):
        return self.evaluationFunction(gameState)
    nextAgent = (agentIndex + 1) % gameState.getNumAgents()
    bestScore = float("inf")
    for legalAction in gameState.getLegalActions(agentIndex):
        legalValue = self.nextValue(gameState, agentIndex, curDepth, nextAgent, legalAction,
alpha, beta)
        bestScore = min(bestScore, legalValue)
        if legalValue < alpha:
            return legalValue
        beta = min(beta, bestScore)
    return bestScore
    # End your code (Part 2)
```

## Part 3

```
    # Begin your code (Part 3)
    #raise NotImplementedError("To be implemented")
    """
        1. Same with part 1
    """
    _, nextAction = self.maxValue(gameState, 0, 0);
    return nextAction
def terminalState(self, gameState, agentIndex, curDepth):
    """
        1. Same with part 1
    """
    if gameState.isWin() or gameState.isLose() or curDepth >= self.depth:
        return True
    return False
def nextValue(self, gameState, agentIndex, curDepth, nextAgent, legalAction):
    """
        1. Same with part 1
    """
    nextState = gameState.getNextState(agentIndex, legalAction)
    legalValue = float()
    if nextAgent == 0:
        legalValue, _ = self.maxValue(nextState, nextAgent, curDepth + 1)
    else:
        legalValue = self.minValue(nextState, nextAgent, curDepth)
    return legalValue
def maxValue(self, gameState, agentIndex, curDepth):
    """
        1. Same with part 1
    """
    if( self.terminalState(gameState, agentIndex, curDepth) ):
        return self.evaluationFunction(gameState), None
```

```
    nextAgent = (agentIndex + 1) % gameState.getNumAgents()
    scores = list()
    legalMoves = gameState.getLegalActions(agentIndex)
    for legalAction in legalMoves:
        legalValue = self.nextValue(gameState, agentIndex, curDepth, nextAgent, legalAction)
        scores.append(legalValue)
    bestScore = max(scores)
    bestIndices = [index for index in range(len(scores)) if scores[index] == bestScore]
    chosenIndex = random.choice(bestIndices)
    return bestScore, legalMoves[chosenIndex]
def minValue(self, gameState, agentIndex, curDepth):
    """
        1. Almost same with part 1
        2. Create scores list to load values with every action
        3. Return sum of values divide by number of values
    """
    if( self.terminalState(gameState, agentIndex, curDepth) ):
        return self.evaluationFunction(gameState)
    nextAgent = (agentIndex + 1) % gameState.getNumAgents()
    scores = list()
    for legalAction in gameState.getLegalActions(agentIndex):
        legalValue = self.nextValue(gameState, agentIndex, curDepth, nextAgent, legalAction)
        scores.append(legalValue)
    meanScore = sum(scores) / len(scores)
    return meanScore
    # End your code (Part 3)
```

## Part 4

```
# Begin your code (Part 4)
#raise NotImplementedError("To be implemented")
"""
    1. Get some information of this state by the function of this game
"""
pacemanDir = currentGameState.getPacmanState().getDirection()
pacmanPos = currentGameState.getPacmanPosition()
ghostsState = [(manhattanDistance(pacmanPos, currentGameState.getGhostPosition(Id)), Id) for Id in
range(1, currentGameState.getNumAgents())]
minGhostDist, minGhostId = (0, 0) if len(ghostsState) == 0 else min(ghostsState)
ghostsDist = [x for x, y in ghostsState]
avgGhostDist = 0 if len(ghostsDist) == 0 else sum (ghostsDist) / len(ghostsDist)
isEat = currentGameState.data.agentStates[minGhostId].scaredTimer > 1
curScore = currentGameState.getScore()
foodDist = [manhattanDistance(pacmanPos, food) for food in currentGameState.getFood().asList()]
numFood = currentGameState.getNumFood()
minFoodDist = 1 if numFood == 0 else min(foodDist)
avgFoodDist = 1 if numFood == 0 else sum(foodDist) / len(foodDist)
numCapsules = len(currentGameState.getCapsules())
capsulesDist = [manhattanDistance(pacmanPos, capsule) for capsule in
currentGameState.getCapsules()]
minCapsuleDist = 1 if len(currentGameState.getCapsules()) == 0 else min( capsulesDist )
wall = currentGameState.hasWall(0,0)
numAgent = currentGameState.getNumAgents() > 1
"""
    2. Return evaluation by some rules with some information
        - If pacman can eat nearest ghost, then return 10 * curScore + (-30 * minGhostDist)
            - Let pacman go to eat ghost
        - Else if pacman is very close with ghosts then return 10 * curScore + (10 * minGhostDist)
            - Keep pacman away from ghosts
```

```
        - Else return 10 * curScore + (-1 * minFoodDist) + (-20 * minCapsuleDist) + (-5 * numFood)
+ (-100 * numCapsules)
            - Let pacman got eat capsule and food
"""
if isEat:
    return 10 * curScore + (-30 * minGhostDist)
elif minGhostDist < 3:
    return 10 * curScore + (10 * minGhostDist)
else:
    return 10 * curScore + (-1 * minFoodDist) + (-20 * minCapsuleDist) + (-5 * numFood) + (-100 *
numCapsules)
# End your code (Part 4)
```

## Part II. Results & Analysis

```
***          >= 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 15:46:19

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

For this game, we want to get more scores and do not die. First, pacman can not be ate by ghosts, so we need to let the value be lower when pacman is close with ghosts and we only care this thing when pacman is very close with ghosts. Second, we want to get more point, so pacman should go to eat ghost after eat capsule. We let the value be better when pacman close to capsule and the number of capsule be fewer. Next we need to judge that pacman can eat ghost or not, if yes then let the value be better when pacman is closer with ghost. Last, we need to finish this game, so we need to let pacman go to eat all food by let the value be better when pacman is closer with nearest food and the number of food is fewer.

But there is some effect between these values, which may cause pacman to wander around food or capsules or ghosts. For example, if pacman eat the food, then it will be farther away next food, then the value will be worse, and pacman need to be close with nearest food, pacman will wander around the food. Other exmple, we try to let pacman go to eat ghosts, so we let that value can be better by constant. But this causes pacman not to eat ghosts because there is no additional constant after eating. There are many examples like above.

Hence we try to make less food a more valuable thing than being close to food. Other examples are handled in a similar way. Then we get a good values' relation like above code.