109550073 陳宥安

Homework 3: Multi-Agent Search

Part I. Implementation (5%):

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

- **Part 1: Minmax Search**

```
# Begin your code (Part 1)
def minmaxagent(agent, depth, state):
    # if win or lose or reach deepest depth -> return value
    if state.isWin() or state.isLose() or depth > self.depth:
        return self.evaluationFunction(state)
    scores = []
    # get what agent can do now
    actions = state.getLegalActions(agent)
    # evaluate every actions that the current agent can do
    for action in actions:
        nextstate = state.getNextState(agent, action)
        if (agent+1 == state.getNumAgents()):       #if PACMAN and GHOSTS has moved
            scores.append(minmaxagent(0, depth+1, nextstate))
        else:
            scores.append(minmaxagent(agent+1, depth, nextstate))
    if agent == 0:              # PACMAN = MAX
        if depth == 1:          # if it is first round
            return scores
        else:
            temp_score = max(scores)
    else:                       # GHOSTS = MIN
        temp_score = min(scores)

    return temp_score
```

```
PACMAN = 0
legalactions = gameState.getLegalActions(PACMAN)
# start the recursion with PACMAN and the depth 1
scores = minmaxagent(PACMAN, 1, gameState)
# get max evaluation value and find its index
maxscore = max(scores)
indices = []
for index in range(len(scores)):
    if scores[index] == maxscore:
        indices.append(index)
chosendindex = random.choice(indices)
return legalactions[chosendindex]
# End your code (Part 1)
```

- **Part 2: Alpha-Beta Pruning**

```
# Begin your code (Part 2)
def alphabetaagent(agent, depth, state, alpha, beta):
    # if win or lose or reach deepest depth -> return value
    if state.isWin() or state.isLose() or depth > self.depth:
        return self.evaluationFunction(state)
    scores = []
    # get what agent can do now
    actions = state.getLegalActions(agent)
    # evaluate every actions that the current agent can do
    for action in actions:
        nextstate = state.getNextState(agent, action)
        if (agent+1 == state.getNumAgents()):       # if PACMAN and GHOSTS has moved
            temp_score = alphabetaagent(0, depth+1, nextstate, alpha, beta)
            scores.append(temp_score)
        else:
            temp_score = alphabetaagent(agent+1, depth, nextstate, alpha, beta)
            scores.append(temp_score)
```

```
        # alpha beta pruning part
        if agent == 0:
            if temp_score > beta:
                return temp_score
            alpha = max(alpha, temp_score)
        else:
            if temp_score < alpha:
                return temp_score
            beta = min(beta, temp_score)
    if agent == 0:              #PACMAN = MAX
        if depth == 1:          #First round
            return scores
        else:
            temp_score = max(scores)
    else:                       #GHOSTS = MIN
        temp_score = min(scores)
    return temp_score
```

```
PACMAN = 0
legalactions = gameState.getLegalActions(PACMAN)
# alpha and beta initialization
alpha, beta = -1000000, 1000000
# start the recursion with PACMAN and the depth 1
scores = alphabetaagent(PACMAN, 1, gameState, alpha, beta)
# get max evaluation value and find its index
maxscore = max(scores)
indices = []
for index in range(len(scores)):
    if scores[index] == maxscore:
        indices.append(index)
chosendindex = random.choice(indices)
return legalactions[chosendindex]
# raise NotImplementedError("To be implemented")
# End your code (Part 2)
```

- **Part 3: Expectimax Search**

```
# Begin your code (Part 3)
def expectimaxagent(agent, depth, state):
    # if win or lose or reach deepest depth -> return value
    if state.isWin() or state.isLose() or depth > self.depth:
        return self.evaluationFunction(state)
    scores = []
    # get what agent can do now
    actions = state.getLegalActions(agent)
    # evaluate every actions that the current agent can do
    for action in actions:
        nextstate = state.getNextState(agent, action)
        if (agent+1 == state.getNumAgents()):       #if PACMAN and GHOSTS has moved
            scores.append(expectimaxagent(0, depth+1, nextstate))
        else:
            scores.append(expectimaxagent(agent+1, depth, nextstate))
    if agent == 0:              # PACMAN = MAX
        if depth == 1:          # First round
            return scores
        else:
            temp_score = max(scores)
    else:                       #GHOSTS
        temp_score = float(sum(scores)/len(scores))    #expectimax calculate
    return temp_score
```

```
PACMAN = 0
legalactions = gameState.getLegalActions(PACMAN)
# start the recursion with PACMAN and the depth 1
scores = expectimaxagent(PACMAN, 1, gameState)
# get max evaluation value and find its index
maxscore = max(scores)
indices = []
for index in range(len(scores)):
    if scores[index] == maxscore:
        indices.append(index)
chosendindex = random.choice(indices)
return legalactions[chosendindex]
# End your code (Part 3)
```

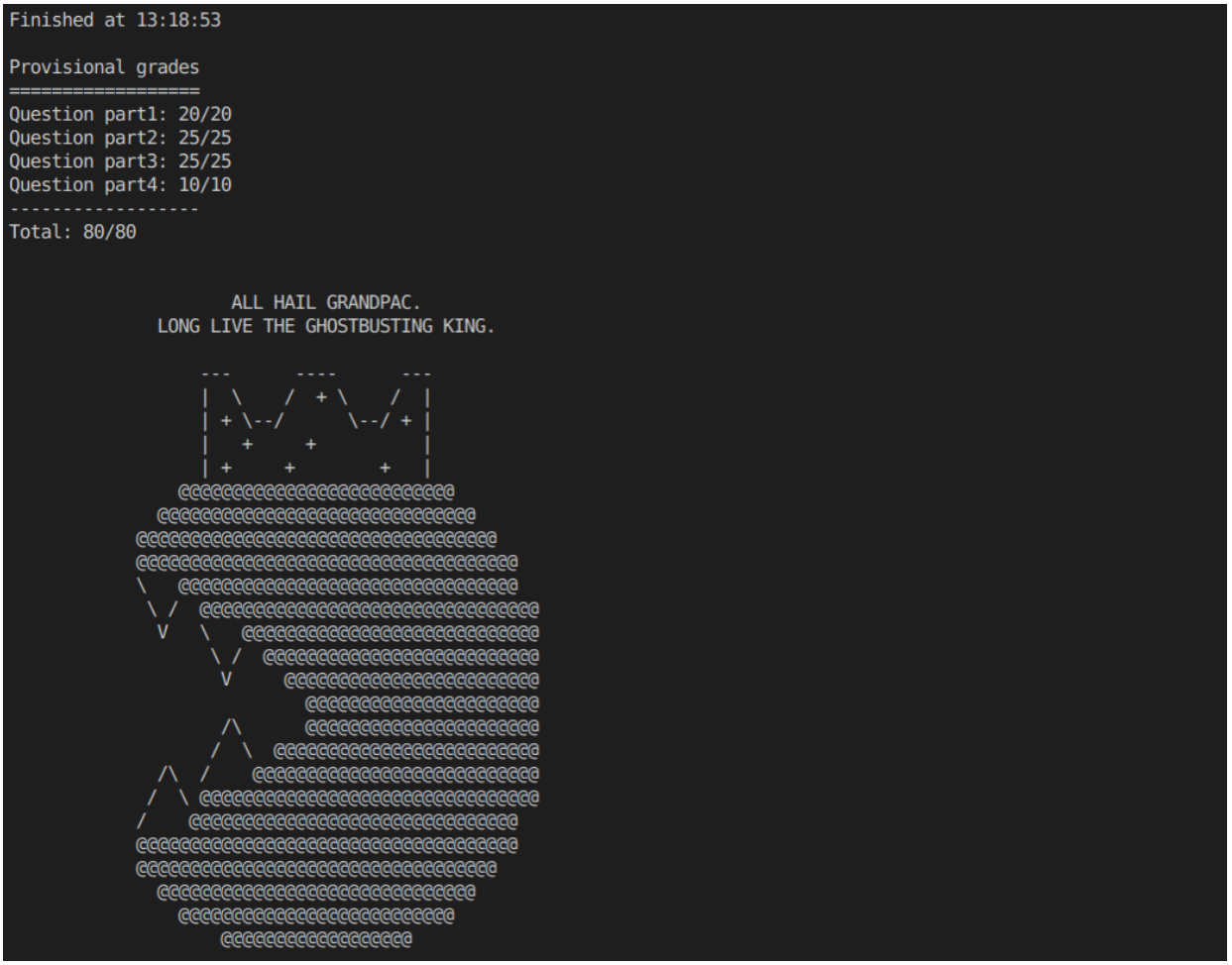
- **Part 4: Evaluation Function**

```
# Begin your code (Part 4)
curscore = currentGameState.getScore()  #get current score
pacmanpos = currentGameState.getPacmanPosition()    #get pacman position
ghostpos = currentGameState.getGhostPositions()     #get ghosts position
foodlist = currentGameState.getFood().asList()      #get food list
foodnum = len(foodlist)
capsulelist = currentGameState.getCapsules()        #get capsule list
capsulenum = len(capsulelist)
closestfood = 1
closestcapsule = 1

# calculate distances from pacman to all food and capsule
fooddis= [manhattanDistance(pacmanpos, foodpos) for foodpos in foodlist]
capsuledis= [manhattanDistance(pacmanpos, capsulepos) for capsulepos in capsulelist]

# Find min food_distance and capsule_distance
if len(foodlist) > 0:
    closestfood = min(fooddis)
if len(capsulelist) > 0:
    closestcapsule = min(capsuledis)
# Find distances from pacman to ghosts
for position in ghostpos:
    ghostdis = manhattanDistance(pacmanpos, position)
    # If ghost is close to pacman, escape from ghosts first
    # by resetting the value of closestfood and closestcapsule
    if ghostdis < 3:
        closestfood = 99999
        closestcapsule= 99999
```

```
# if closest capsule distance is bigger than closest food distance
# Go for food first by resetting closestcapsule
if closestcapsule > closestfood:
    closestcapsule= 99999
# if closest capsule distance is smaller than closest food distance
# Go for food first by resetting closestfood
else:
    closestfood= 99999
# set evaluation features and their weights
features = [1.0/closestfood, 1.0/closestcapsule, curscore, foodnum, capsulenum]
weights = [1, 1, 50, -1, -1]
# Linear combination of features
return sum([feature * weight for feature, weight in zip(features, weights)])
# End your code (Part 4)
```

Part II. Results & Analysis (5%):

Please screenshot the results.



My better evaluation function basically tries to make PACMAN eat more capsules and food. Except for when the ghosts are too close to the PACMAN, it will escape from them first by resetting the variables' values.

It's quite hard to find the right weight for each feature to get a higher score. It takes me a lot of time to find the best weight. Thankfully, It worked for the homework.