HW3: Multi-Agent Search

Part I. Implementation (20%):

Part1:Minimax Search

```
    # Begin your code (Part 1)
    """
    1. check foe termination condition if the game is win or lose
    or if it reach the limit depth terminal
    2.Expand the node find all legal action for current agent and
    remove the stop action(to make it faster) calculate the next state
    of game
    3. Recursive case handling if current agent is not the last one
    call minimax ,if it is last start a new depth level with first agent
    4. if the agent is pacman retrun the max or the action
    if is enemy return the min
    """
    def minimax(state, depth, agentIndex):
        isTerminal = state.isWin() or state.isLose()
        if isTerminal or depth == self.depth:
            return self.evaluationFunction(state)
        valnodes=[]
        actions=state.getLegalActions(agentIndex)
        if Directions.STOP in actions:
            actions.remove(Directions.STOP)
        for action in actions:
            nextstate=state.getNextState(agentIndex, action)
            if (agentIndex !=(gameState.getNumAgents()-1)):
                valnodes.append(minimax(nextstate, depth, agentIndex+1))
            else:
                valnodes.append(minimax(nextstate,depth+1,0))
        if agentIndex==0:
            if depth==0:
                best_score = max(valnodes)
                best_action_index = valnodes.index(best_score)
                return actions[best_action_index]
            else:
                return max(valnodes)
        else:
            return min(valnodes)
    return minimax(gameState, 0, 0)
    # End your code (Part 1)
```

Part 2: Alpha-Beta Pruning

```
# Begin your code (Part 2)
"""
1. Check for termination condition: if the game is won or lost, or if it reaches
  the limit depth, then return the evaluation of the current state.
2. Expand the node: Find all legal actions for the current agent and remove the STOP
action to keep the game moving. Calculate the next state of the game.
3. Recursive case handling: If the current agent is not the last one, call alphabeta
for the next agent. If it is the last agent, start a new depth level with the first agent.
4. Apply alpha-beta pruning to cut off branches that cannot possibly affect the final decision:
- If the agent is Pac-Man (maximizer), update alpha with the maximum value found and
  prune the remaining branches if alpha is greater than to beta.
- If the agent is an enemy (minimizer), update beta with the minimum value found and
  prune the remaining branches if beta is less than to alpha.
5. If the agent is Pac-Man, return the max value or the best action; if it is an enemy,
  return the min value.
"""
def alphabeta(state, depth, agentIndex, alpha, beta):
    isTerminal = state.isWin() or state.isLose()
    if isTerminal or depth == self.depth:
        return self.evaluationFunction(state)
    actions = state.getLegalActions(agentIndex)
    if Directions.STOP in actions:
        actions.remove(Directions.STOP)
    if agentIndex == 0:
        bestValue = float("-inf")
        bestAction = None
        for action in actions:
            nextState = state.getNextState(agentIndex, action)
            value = alphabeta(nextState, depth, agentIndex + 1, alpha, beta)
            if value > bestValue:
                bestValue = value
                bestAction = action
            alpha = max(alpha, bestValue)
            if beta < alpha:
                break
        if depth == 0:
            return bestAction
        else:
            return bestValue
    else:
```

```
            return bestValue
    else:
        minValue = float("inf")
        for action in actions:
            nextState = state.getNextState(agentIndex, action)
            if (agentIndex ==(gameState.getNumAgents()-1)):
                value = alphabeta(nextState, depth + 1, 0, alpha, beta)
            else:
                value = alphabeta(nextState, depth, agentIndex+1, alpha, beta)
            minValue = min(minValue, value)
            beta = min(beta, minValue)
            if beta < alpha:
                break
        return minValue
return alphabeta(gameState, 0, 0, float("-inf"), float("inf"))
# End your code (Part 2)
```

Part 3: Expectimax Search

```
        # Begin your code (Part 3)
        """
        1. check foe termination condition if the game is win or lose
        or if it reach the limit depth terminal
        2.Expand the node find all legal action for current agent and
        remove the stop action(to make it faster) calculate the next state
        of game
        3. Recursive case handling if current agent is not the last one
        call minimax ,if it is last start a new depth level with first agent
        4. if the agent is pacman retrun the max or the action
        if is enemy return the expected value
        """
        def expectimax(state, depth, agentIndex):
            isTerminal = state.isWin() or state.isLose()
            if isTerminal or depth == self.depth:
                return self.evaluationFunction(state)
            valnodes=[]
            actions=state.getLegalActions(agentIndex)
            if Directions.STOP in actions:
                actions.remove(Directions.STOP)
            for action in actions:
                nextstate=state.getNextState(agentIndex, action)
                if (agentIndex !=(gameState.getNumAgents()-1)):
                    valnodes.append(expectimax(nextstate, depth, agentIndex+1))
                else:
                    valnodes.append(expectimax(nextstate,depth+1,0))
            if agentIndex==0:
                if depth==0:
                    best_score = max(valnodes)
                    best_action_index = valnodes.index(best_score)
                    return actions[best_action_index]
                else:
                    return max(valnodes)
            else:
                return sum(valnodes)/len(valnodes)
        return expectimax(gameState, 0, 0)
        # End your code (Part 3)
```

Part 4:Evalution Fuction

```
# Begin your code (Part 4)
"""
1. Calculate the basic score from the current game state.
2. Determine the Pac-Man's current position, list of remaining food, and capsules.
3. Calculate distances to the nearest food item, nearest capsule, and scared ghosts.
4. Calculate additional scores based on these distances and the number of remaining food items to prioritize food
collection, capsule collection, and chasing scared ghosts.
5. Return the total score which includes the base game score and bonuses from food, capsules, and scared ghosts.
"""
score = currentGameState.getScore()
pos = currentGameState.getPacmanPosition()
foodList = currentGameState.getFood().asList()
capsuleList = currentGameState.getCapsules()
ghostStates = currentGameState.getGhostStates()

minFoodDist = float('inf')
minCapsuleDist = float('inf')
scaredGhostDist = float('inf')

for food in foodList:
    minFoodDist=min(minFoodDist,manhattanDistance(pos,food))
for capsule in capsuleList:
    minCapsuleDist=min(minCapsuleDist,manhattanDistance(pos,capsule))
for ghost in ghostStates:
    if ghost.scaredTimer>0:
        scaredGhostDist=min(scaredGhostDist,manhattanDistance(pos, ghost.getPosition()))
foodRemainingScore =len(foodList)
return score+(30/(minFoodDist+foodRemainingScore))+(45/(minCapsuleDist))+(500/(scaredGhostDist))
# End your code (Part 4)
```

Part II. Results s Analysis (10%):

Q1

```
Question part1
==============

*** PASS: test_cases\part1\0-eval-function-lose-states-1.test
*** PASS: test_cases\part1\0-eval-function-lose-states-2.test
*** PASS: test_cases\part1\0-eval-function-win-states-1.test
*** PASS: test_cases\part1\0-eval-function-win-states-2.test
*** PASS: test_cases\part1\0-lecture-6-tree.test
*** PASS: test_cases\part1\0-small-tree.test
*** PASS: test_cases\part1\1-1-minmax.test
*** PASS: test_cases\part1\1-2-minmax.test
*** PASS: test_cases\part1\1-3-minmax.test
*** PASS: test_cases\part1\1-4-minmax.test
*** PASS: test_cases\part1\1-5-minmax.test
*** PASS: test_cases\part1\1-6-minmax.test
*** PASS: test_cases\part1\1-7-minmax.test
*** PASS: test_cases\part1\1-8-minmax.test
*** PASS: test_cases\part1\2-1a-vary-depth.test
*** PASS: test_cases\part1\2-1b-vary-depth.test
*** PASS: test_cases\part1\2-2a-vary-depth.test
*** PASS: test_cases\part1\2-2b-vary-depth.test
*** PASS: test_cases\part1\2-3a-vary-depth.test
*** PASS: test_cases\part1\2-3b-vary-depth.test
*** PASS: test_cases\part1\2-4a-vary-depth.test
*** PASS: test_cases\part1\2-4b-vary-depth.test
*** PASS: test_cases\part1\2-one-ghost-3level.test
*** PASS: test_cases\part1\3-one-ghost-4level.test
*** PASS: test_cases\part1\4-two-ghosts-3level.test
*** PASS: test_cases\part1\5-two-ghosts-4level.test
*** PASS: test_cases\part1\6-tied-root.test
*** PASS: test_cases\part1\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part1\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part1\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part1\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part1\8-pacman-game.test

### Question part1: 15/15 ###
```

Q2:

```
Question part2
==============

*** PASS: test_cases\part2\0-eval-function-lose-states-1.test
*** PASS: test_cases\part2\0-eval-function-lose-states-2.test
*** PASS: test_cases\part2\0-eval-function-win-states-1.test
*** PASS: test_cases\part2\0-eval-function-win-states-2.test
*** PASS: test_cases\part2\0-lecture-6-tree.test
*** PASS: test_cases\part2\0-small-tree.test
*** PASS: test_cases\part2\1-1-minmax.test
*** PASS: test_cases\part2\1-2-minmax.test
*** PASS: test_cases\part2\1-3-minmax.test
*** PASS: test_cases\part2\1-4-minmax.test
*** PASS: test_cases\part2\1-5-minmax.test
*** PASS: test_cases\part2\1-6-minmax.test
*** PASS: test_cases\part2\1-7-minmax.test
*** PASS: test_cases\part2\1-8-minmax.test
*** PASS: test_cases\part2\2-1a-vary-depth.test
*** PASS: test_cases\part2\2-1b-vary-depth.test
*** PASS: test_cases\part2\2-2a-vary-depth.test
*** PASS: test_cases\part2\2-2b-vary-depth.test
*** PASS: test_cases\part2\2-3a-vary-depth.test
*** PASS: test_cases\part2\2-3b-vary-depth.test
*** PASS: test_cases\part2\2-4a-vary-depth.test
*** PASS: test_cases\part2\2-4b-vary-depth.test
*** PASS: test_cases\part2\2-one-ghost-3level.test
*** PASS: test_cases\part2\3-one-ghost-4level.test
*** PASS: test_cases\part2\4-two-ghosts-3level.test
*** PASS: test_cases\part2\5-two-ghosts-4level.test
*** PASS: test_cases\part2\6-tied-root.test
*** PASS: test_cases\part2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\part2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part2\7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part2\8-pacman-game.test

### Question part2: 20/20 ###
```

Q3:

```
Question part3
==============

*** PASS: test_cases\part3\0-eval-function-lose-states-1.test
*** PASS: test_cases\part3\0-eval-function-lose-states-2.test
*** PASS: test_cases\part3\0-eval-function-win-states-1.test
*** PASS: test_cases\part3\0-eval-function-win-states-2.test
*** PASS: test_cases\part3\0-expectimax1.test
*** PASS: test_cases\part3\1-expectimax2.test
*** PASS: test_cases\part3\2-one-ghost-3level.test
*** PASS: test_cases\part3\3-one-ghost-4level.test
*** PASS: test_cases\part3\4-two-ghosts-3level.test
*** PASS: test_cases\part3\5-two-ghosts-4level.test
*** PASS: test_cases\part3\6-1a-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1b-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-1c-check-depth-one-ghost.test
*** PASS: test_cases\part3\6-2a-check-depth-two-ghosts.test
*** PASS: test_cases\part3\6-2b-check-depth-two-ghosts.test
*** PASS: test_cases\part3\6-2c-check-depth-two-ghosts.test
*** Running ExpectimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running ExpectimaxAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\part3\7-pacman-game.test

### Question part3: 20/20 ###
```

Q4:

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

Pacman emerges victorious! Score: 1334
Pacman emerges victorious! Score: 1330
Pacman emerges victorious! Score: 1356
Pacman emerges victorious! Score: 1336
Pacman emerges victorious! Score: 1348
Pacman emerges victorious! Score: 1347
Pacman emerges victorious! Score: 1340
Pacman emerges victorious! Score: 1311
Pacman emerges victorious! Score: 1354
Pacman emerges victorious! Score: 1289
Average Score: 1334.5
Scores:        1334.0, 1330.0, 1356.0, 1336.0, 1348.0, 1347.0, 1340.0, 1311.0, 1354.0, 1289.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\part4\grade-agent.test (8 of 8 points)
*** EXTRA CREDIT: 2 points
***     1334.5 average score (4 of 4 points)
***         Grading scheme:
***          < 600:  0 points
***         >= 600:  2 points
***         >= 1200:  4 points
***     10 games not timed out (2 of 2 points)
***         Grading scheme:
***          < 0:  fail
***         >= 0:  0 points
***         >= 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 1:38:15

Provisional grades
==================
Question part1: 15/15
Question part2: 20/20
Question part3: 20/20
Question part4: 10/10
------------------
Total: 65/65
```

Discussion

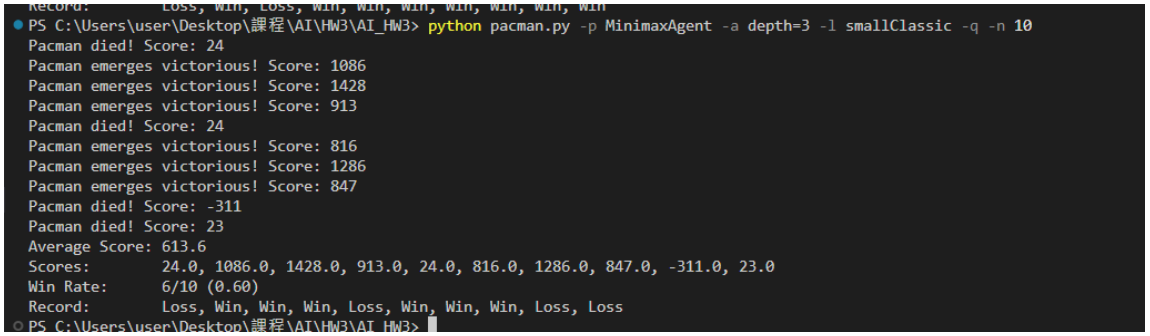
I assume that the formula is:

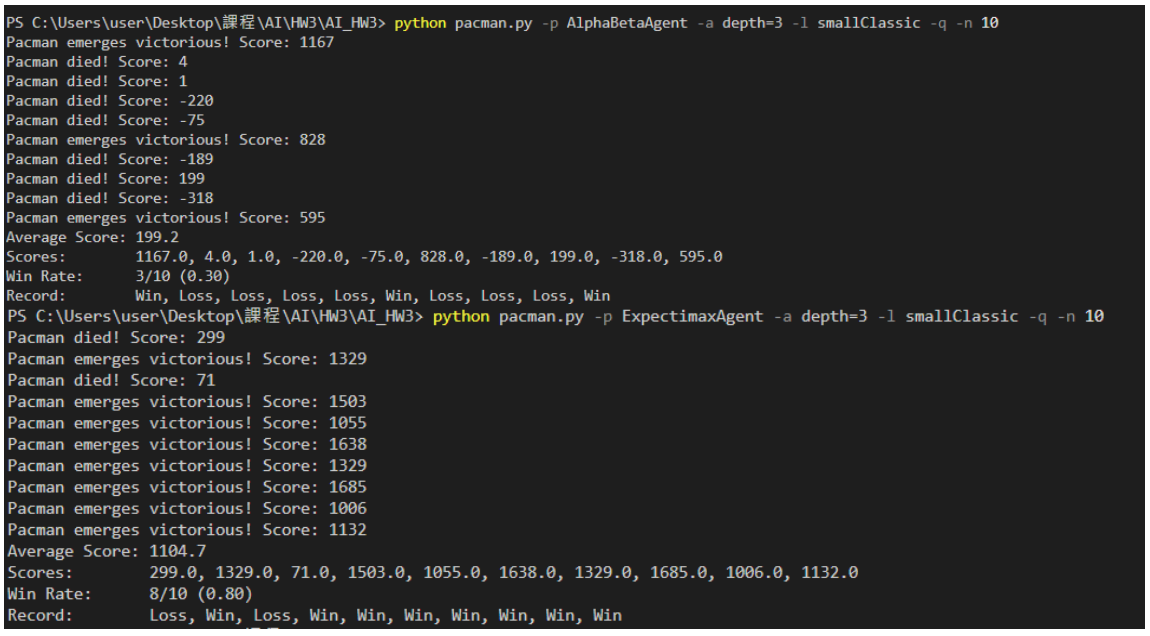
Current score+(x/(minfooddis+remainingfood))+(y/mincapsuledist)+(z/scaredGhostDist)

I make x,y,z with10 50 200,but I find it is more suitable for 30 45 500.And the remainingfood it is because I think the less food it is, the more important of the food.it can improve the performance of the evaluation.

Analysis:

I try every algorithms by python pacman.py -p Agent -a depth=3 -l smallClassic -q -n 10





The ExpectimaxAgent seems to perform the best in terms of both average score and win rate. This suggests that the expectimax algorithm might be better suited for the stochastic nature of the Pac-Man game in these simulations.

The MinimaxAgent performs better than the AlphaBetaAgent but is not as good as the ExpectimaxAgent. The scores indicate that it has a reasonable strategy but not as robust as ExpectimaxAgent.

The AlphaBetaAgent has the lowest average score and the highest number of losses, which might suggest that while alpha-beta pruning is more efficient in terms of computation, it may not always lead to the best outcomes in this scenario.