Question5: Evaluation Function

B05901030 電機三 陳欽安

Construction of Evaluation Function

```
from searchAgents import mazeDistance
import search
```

In the beginning, I import ***mazeDistance*** from ***searchAgents*** and ***search***.

```
def betterEvaluationFunction(currentGameState):


    x,y = currentGameState.getPacmanPosition()
    food_list = currentGameState.getFood().asList()
    cap_list = currentGameState.getCapsules()
    ghost_states = currentGameState.getGhostStates()
    lower_bd = currentGameState.getFood().width + currentGameState.getFood().height
    dis_f = [manhattanDistance((x,y),food) for food in food_list]
    dis_cap = [mazeDistance((x,y),cap,currentGameState) for cap in cap_list]
    dis_ghost_can_eat = []
    dis_ghost_escape = []
    eva = lower_bd
```

In ***betterEvaluationFunction***, I initialize the variables above to store what I'll need for later process.

1. Store coordinates of Pacman, food, capsules.
2. Initialize a variable, ***lower_bd***, to the sum of maze's width and height, due to this value is the maximum possible distance between Pacman and food.
3. Store the Manhattan distance between each food and Pacman in ***dis_f***.
4. Store the Maze distance between each capsule and Pacman in ***dis_cap***.
5. Initialize two empty lists ready to store the distance between ghost and Pacman in ***dis_ghost_can_eat*** and ***dis_ghost_escape***.
6. Initialize variable ***eva*** = 0 to store our result of evaluation.

```
for ghost in ghost_states:
            distance_g = manhattanDistance((x,y),ghost.getPosition())
            if ghost.scaredTimer > 2 :
                    dis_ghost_can_eat.append(distance_g)
```

```
            else:
                if distance_g < 4 :
                    dis_ghost_escape.append(distance_g)
```

Later, it's time to deal with ghosts.

1. If ghost is scared (it can be eaten now), append the Manhattan distance between it and Pacman in the list ***dis_ghost_can_eat***.
   Here, I decide to regard every ghost with scared time longer than 3 steps as eatable ghost.
2. If ghost isn't scared (Pacman should rather run away), append the Manhattan distance between it and Pacman in the list ***dis_ghost_escape***.
   Here, I decide to consider every ghost with distance between Pacman shorter than 3 to be dangerous ghost that Pacman should immediately run away from it.

```
    if dis_f:
        eva = eva + (lower_bd - min(dis_f))
    if dis_cap:
        eva = eva + 3*(lower_bd - min(dis_cap))
    if dis_ghost_can_eat:
        eva = eva + 2*(lower_bd - min(dis_ghost_can_eat))
    if dis_ghost_escape:
        eva = eva + 4*min(dis_ghost_escape) + (sum(dis_ghost_escape)/len(dis_ghost_escape))/2
```

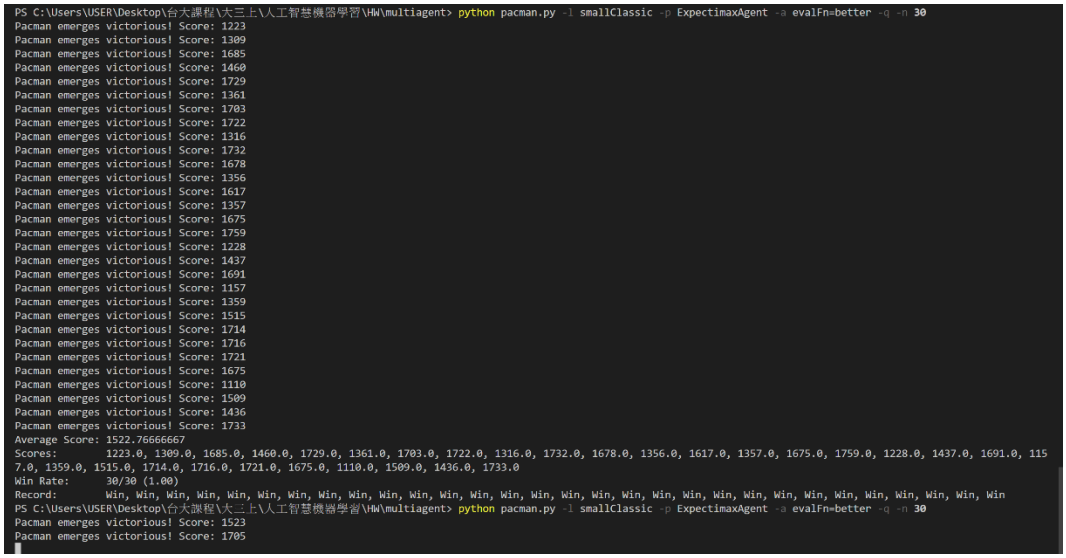
After storing every information, we might need to evaluate the state, I'm going to give different variables different weight based on their distribution.

1. To avoid death, I give distribution of dangerous ghost the highest weight, 4.
   The farther the better, so the distribution is ***min(dis_ghost_escape)***.
   Later, I add the average of ghosts' distances to fine-tune the performance.
2. To make ghost eatable, I give distribution of capsule the second-high weight, 3.
   The closer the better, so the distribution is ***lower_bd – min(dis_cap)***.
3. After eat capsule, we can chase ghost. I give distribution of eatable ghost the third-high weight, 2.
   The closer the better, so the distribution is ***lower_bd-min(dis_ghost_can_eat)***.
4. Last, give the distribution of food the lowest weight, 1.
   The closer the better, so the distribution is ***lower_bd-min(dis_f)***.

```
return eva + currentGameState.getScore()
```

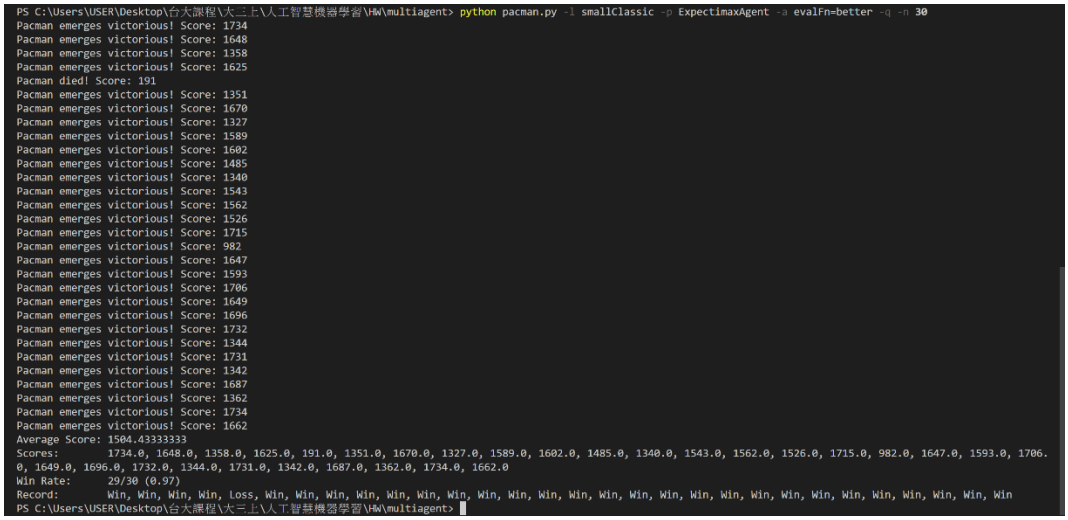
Eventually, I return the value of ***eva*** plus current score.

Performance of Evaluation Function



Win rate: 1.00

Average Score: 1522.77



Win rate: 0.97

Average Score: 1504.43