ESIEE Paris

Artificial Intelligence and Cybersecurity

Artificial Intelligence

Lab 3 Report

Bruno Luiz Dias Alves de Castro
Victor Gabriel Mendes Sündermann

April 7, 2023

Contents

1 Introduction

In this report, we will present the results of the third project of the Artificial Intelligence course. The project consists of implementing

2 Tests

To run the program, we use the following command:

```
python3 pacman_AIC.py -p ReflexAgent [-l testClassic]
```

Running the command without the flag *-l testClassic*, renders a version of the maze without any walls.

The first two tests ran as expected, the Pacman tries to eat all the food in the field while avoiding all the ghosts. The mai problem with both tests is that, while avoiding the ghosts, it'll stay in place most of the time, and not go after the food.

This can be improved by implementing an evaluation function, which will be developed in the next section.

3 Reflex Agent

In this part, we were purposed to improve the **ReflexAgent** function in the **multiAgents.py** file in the project by proposing an evaluation function.

3.1 ReflexAgent improvement

In order to improve the **ReflexAgent**, we need a good evalution function to help the AI decide what to do next. A great evaluation function should be able to tell the agent what is the best action to take in a given state, rewarding good actions and punishing bad ones.

For our problem, we decided that a good action should be eating a food pellet, and a bad action should be getting eaten by a ghost. So, our evaluation function calculates the distance of Pacman to the food pallets and the ghosts. It then returns a score like the following:

```
    return score + closestFoodScore - closestGhostScore
```

We also decided on using the Manhattan distance to calculate the distance between Pacman, the ghosts and the food pellets, because it is a less computationally expensive sollution to this problem, and don't really affect the final score.

Finally, the result we got is the following:

```
def evaluationFunction(self, currentGameState, action):

    """ VARIABLES """

    foodList = newFood.asList()

    # manhattan distance
    foodDistances = \
        [manhattanDistance(newPos, food) for food in foodList]
    ghostDistances = \
        [manhattanDistance(newPos, ghost) for ghost in newGhostsPos]

    # closest food and ghost
    closestFood = min(foodDistances) if len(foodDistances) > 0 else 0
    closestGhost = min(ghostDistances) if len(ghostDistances) > 0 else 0

    # score calculation for food and ghost
    closestFoodScore = 0 if closestFood == 0 else 1.0/closestFood
    closestGhostScore = 0 if closestGhost == 0 else 1.0/closestGhost

    score = closestFoodScore - closestGhostScore

    return successorGameState.getScore() + score
```

3.2 Tests

In order to test our new evaluation function, we executed the following tests:

3.2.1 testClassic

Command:

```
    python3 pacman_AIC.py -p ReflexAgent -l testClassic
```

Output:

```
Pacman emerges victorious! Score: 562
Average Score: 562.0
Scores:        562.0
Win Rate:      1/1 (1.00)
Record:        Win
```

3.2.2 mediumClassic

- One ghost test run

Command:

```
python3 pacman_AIC.py -p ReflexAgent -k 1
```

Output:

```
Pacman emerges victorious! Score: 1485
Average Score: 1485.0
Scores:        1485.0
Win Rate:      1/1 (1.00)
Record:        Win
```

- Two ghosts test run

Command:

```
python3 pacman_AIC.py -p ReflexAgent -k 2
```

Output:

```
Pacman emerges victorious! Score: 1407
Average Score: 1407.0
Scores:        1407.0
Win Rate:      1/1 (1.00)
Record:        Win
```

3.2.3 No layout 20 runs average

During this part of the testing, we ran the **ReflexAgent** 20 times on each of the 6 conbinations of the settings listed below, and calculated the average score and the win rate. The possible settings are as listed (indentified with the letters a-e):

- Possible settings:
  (a) One ghost
  (b) Two ghosts
  (c) Random ghosts
  (d) Random ghosts with fixed seed
  (e) Non random ghost

The results obteined are shown in Table 1.

3.2.4 Results

The **ReflexAgent** was sucessful in clearing most of the purposed tests. Especially the the ran in sections 3.2.1 and 3.2.2 (**testClassic** and **mediumClassic**), were no problem for the agent.

Things start to fall short when it comes to test in section 3.2.3. The agent obtained decent success in the test with a single ghost, but performenced declined on tests with two ghost. The agent only cleared 6 out of 20 tests with random ghosts, and was onle able to clear a single test with non random ghosts.

Configuration	(a, c)	(a, d)	(a, e)	(b, c)	(b, d)	(b, e)
Average Score	671.65	785.2	449.9	464.7	228.7	-14.25
Win Rate	14/20 (0.70)	18/20 (0.90)	10/20 (0.50)	10/20 (0.50)	6/20 (0.30)	1/20 (0.05)

Table 1: ReflexAgent test results

4 Minimax

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vitae nisl nec nunc placerat lacinia.

4.1 Minimax implementation

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vitae nisl nec nunc placerat lacinia.

```
def MAX_VALUE(self, gameState, d):
    if d == 0 or gameState.isWin() or gameState.isLose():
        # base case: return evaluation function
        return self.evaluationFunction(gameState), Directions.STOP

    bestScore, bestAction = -9999, Directions.STOP

    for action in gameState.getLegalActions(0):
        successors = gameState.generateSuccessor(0, action)

        # call minimaxer agent (ghosts)
        value, _ = self.MIN_VALUE(successors, d, 1)

        # update best score and action
        if value > bestScore:
            bestScore, bestAction = value, action

    return bestScore, bestAction
```

Table 2: Maximizer function

4.2 Tests

In order to test the Minimax algorithm implemented, we ran it 20 times, using different depths, and the results are shown in the table below:

```python
def MIN_VALUE(self, gameState, d, indexAgent):
    if d == 0 or gameState.isWin() or gameState.isLose():
        # base case: return evaluation function
        return self.evaluationFunction(gameState), Directions.STOP

    bestScore, bestAction = 9999, Directions.STOP

    for action in gameState.getLegalActions(indexAgent):
        successors = gameState.generateSuccessor(indexAgent, action)
        if indexAgent == gameState.getNumAgents() - 1:
            # call pacman
            value, _ = self.MAX_VALUE(successors, d - 1)
        else:
            # call next ghost
            value, _ = self.MIN_VALUE(successors, d, indexAgent + 1)

        # update best score and action
        if value < bestScore:
            bestScore, bestAction = value, action

    return bestScore, bestAction
```

Table 3: Minimizer function

5 AlphaBeta algorithm

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla vitae nisl nec nunc placerat lacinia.

5.1 AlphaBeta implementation

In order to implement the AlphaBeta algorithm, we used the same functions as the Minimax algorithm, but added two parameters to the functions, alpha and beta. The alpha and beta parameters are used to prune the search tree, and are initialized as -9999 and 9999 respectively. The alpha and beta parameters are updated in the minimizer function, and the maximizer function, as shown in the code below:

Depth	2	3	4
Iteration	-	-	-
1	-231 (L)	-276 (L)	840 (W)
2	-206 (L)	1254 (W)	32 (L)
3	-244 (L)	-336 (L)	1732 (W)
4	-154 (L)	-263 (L)	-314 (L)
5	1249 (W)	966 (W)	311 (W)
6	-688 (L)	-96 (L)	540 (L)
7	-185 (L)	1138 (W)	204 (L)
8	-203 (L)	959 (W)	479 (L)
9	-239 (L)	1531 (W)	1171 (W)
10	-140 (L)	116 (L)	-97 (L)
11	-317 (L)	1373 (W)	896 (W)
12	178 (L)	1039 (W)	1218 (W)
13	-176 (L)	-174 (L)	1048 (W)
14	-624 (L)	1168 (W)	223 (L)
15	-180 (L)	-156 (L)	402 (L)
16	-238 (L)	749 (W)	1238 (W)
17	-243 (L)	-267 (L)	1202 (W)
18	8 (L)	1147 (W)	1631 (W)
19	-368 (L)	1319 (W)	277 (L)
20	-454 (L)	-272 (L)	985 (W)
Average	-173	546	701
Best	1249	1531	1732
Win Rate	1/20 (0.05)	11/20 (0.55)	11/20 (0.55)

Table 4: Minimax test results

```
""" CODE """

if value > beta:
  return value, action

alpha = max(alpha, value)

""" CODE """
```

Table 5: Maximizer function with beta pruning

```
""" CODE """

if value < alpha:
  return value, action

""" CODE """
```

Table 6: Minimizer function with alpha pruning

Depth	2	3	4	5
Average	-121.5	413.6	635.85	0
Best	1001	1604	1701	0
Win Rate	1/20 (0.05)	9/20 (0.45)	11/20 (0.55)	0

Table 7: Minimax test results