TP3 Report
Artificial Intelligence

Bruno Luiz Dias Alves de Castro
Victor Gabriel Mendes Sündermann

April 2023

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:

```python
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 test:

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.

```python
def MAX_VALUE(self, gameState, d):
  if d == 0 or gameState.isWin() or gameState.isLose():
      return self.evaluationFunction(gameState), Directions.STOP

  bestScore, bestAction = -9999, Directions.STOP

  for action in gameState.getLegalActions(0):
      successors = gameState.generateSuccessor(0, action)
      value, _ = self.MIN_VALUE(successors, d, 1)
      if value > bestScore:
          bestScore, bestAction = value, action

  return bestScore, bestAction
```

Table 1: 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():
      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:
          value, _ = self.MAX_VALUE(successors, d - 1)
      else:
          value, _ = self.MIN_VALUE(successors, d, indexAgent + 1)

      if value < bestScore:
          bestScore, bestAction = value, action

  return bestScore, bestAction
```

Table 2: Minimizer function

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 3: Minimax test results