University of Science and Technology at Zewail City

Communication and Information Engineering

CIE 458
Artificial Intelligence

Pacman Project Report

Student Information

Name: Fatma Moanes
ID: 201700346

Course Instructor

Dr. Mohamed Elshenawy

Contents

1 Reflex Agent

A Reflex Agent examines all the possible actions of a state and decides the best action to take according to the Evaluation Function.

1.1 Evaluation Function Approach

Pacman's main goal in the game is to eat all the food, without being eaten from ghosts. Accordingly, the evaluation function was designed so as to give penalties for approaching a ghost, or going away from food.

First of all, the Manhattan's distances between Pacman and both the ghosts and food was computed. Then, the evaluation score was boosted as the distance to the nearest ghost increased and reduced as the distance to the nearest food increased, by certain weights. However, it was found experimentally that using the reciprocals of distances gave better results. Finally, trial and error approach was used to get the weights. The function returned the summation of the score and evaluation score.

To sum up, the Evaluation Function was designed to make Pacman run away from ghosts and chase food.

```
# Adding a penalty if pacman gets near a ghost, or away from food
        eval_score -= 70 * 1 / min(pacman_ghosts_distances)
        eval_score += 10 * 1 / min(pacman_food_distances)
```

1.2 Results

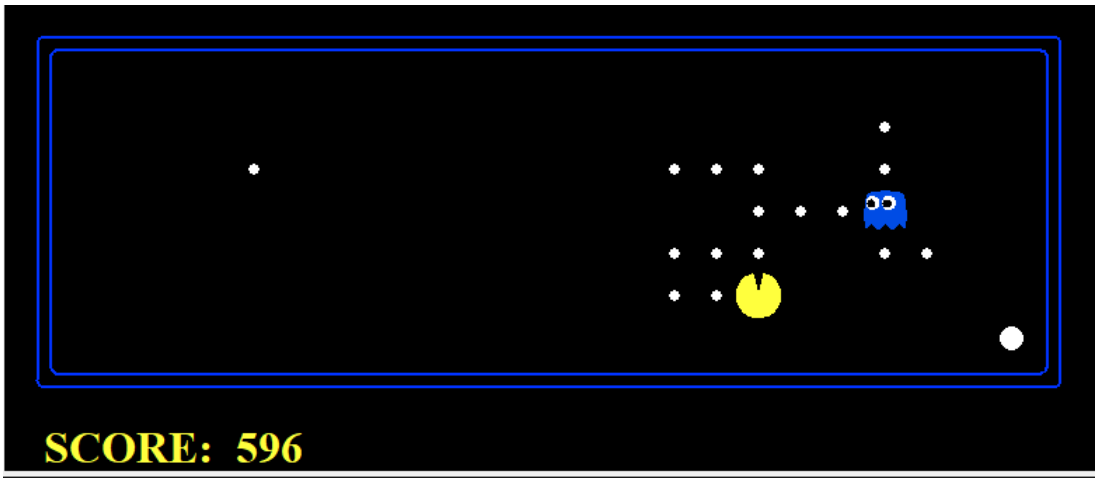

Figure 1: The Reflex Agent's behaviour according to the Evaluation Function

Figure 2: Pacman won with a score of 841

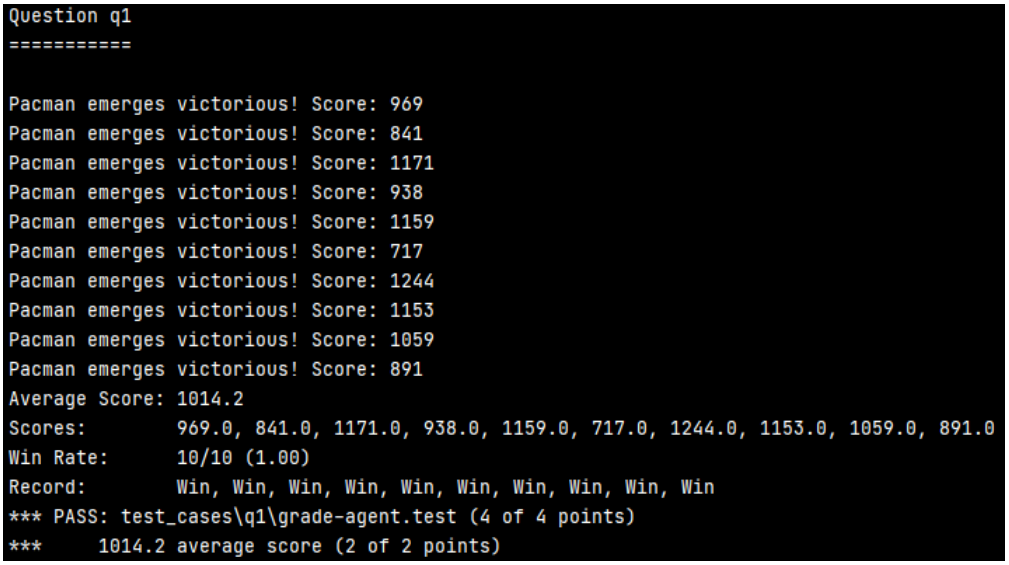

Figure 3: Results of Question one's autograder

2 Multi-Agent Searchers

2.1 Minimax

The *minimax algorithm* considers that the opponent will always behave in an optimal way. The agent always assumes the worst. However, it does not win, since the ghosts are *random*, not optimal. In this project, Pacman is considered **Max agent**, while each one of the ghosts is considered a **Min agent**.

2.1.1 Algorithm approach

The *minimax* function is a recursive function that simulates adversarial agents taking turns. It calls the **"max_value"** function (Pacman) which in turn calls the **"min_value"** function (first ghost). The **"min_value"** function keeps calling itself a number of times equal to the number of ghosts (ghosts taking turns). Then, the last ghost calls the **"max_value"** function. The same process continues until we get to the base case, which is winning, losing, or reaching the maximum depth.**"max_value"** returns the maximum possible evaluation score, while the **"min_value"** returns the minimum one. The **"minimax"** function returns the best possible action for Pacman.

2.1.2 Results

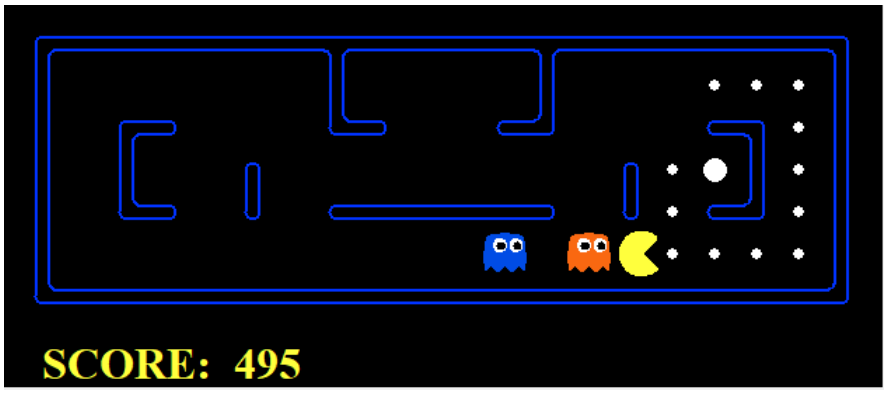

Figure 4: Pacman's behaviour according to minimax algorithm

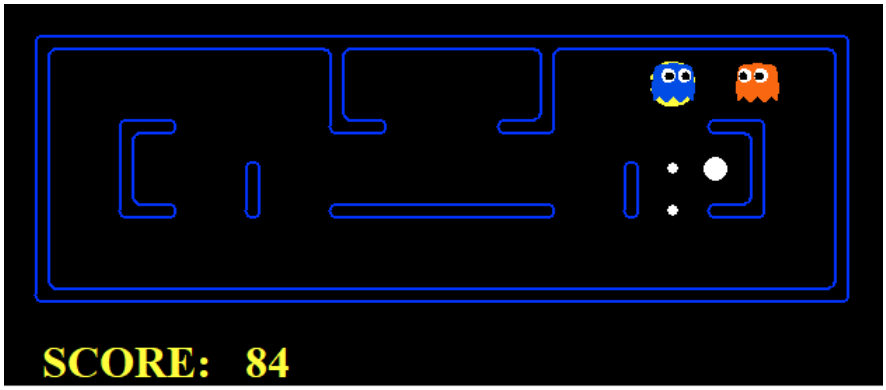

Figure 5: Pacman lost the game

```
*** 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 18 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###
```

Figure 6: Results of Question two's autograder

2.2 Alpha-Beta Pruning

The *Alpha-Beta pruning algorithm* is like the minimax algorithm. But it tries to prune the search tree by determining the paths that will never be taken by Pacman or the ghosts. In the end, Pacman takes the same actions' decisions using both algorithms, they give the same results. However, *Alpha-Beta pruning algorithm* is **much faster**. Likewise, It does not win, as it considers the ghosts as optimal while in fact they are *random*. The algorithm stores two variables, **Alpha (α)** and **Beta (β)**. α is the best score for Pacman along the path from the root to the end, β is the best score for a ghost along the path from the root to the end (lowest score for Pacman).

2.2.1 Algorithm approach

The*Alpha-Beta* algorithm has the same approach as the minimax algorithm. What is different is that α gets updated in the **"max_value"** function to be $max(alpha, maxi)$, where $maxi$ is the highest score found by Pacman along the path. While β gets updated in the **"min_value"** function to be $min(beta, mini)$, where $mini$ is the lowest score found by Pacman along the path (best option for a ghost).

2.2.2 Results

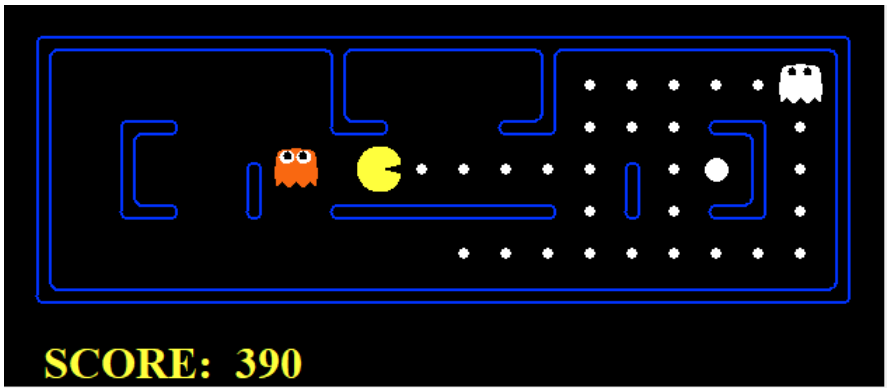

Figure 7: Pacman's behaviour according to alpha-beta pruning algorithm

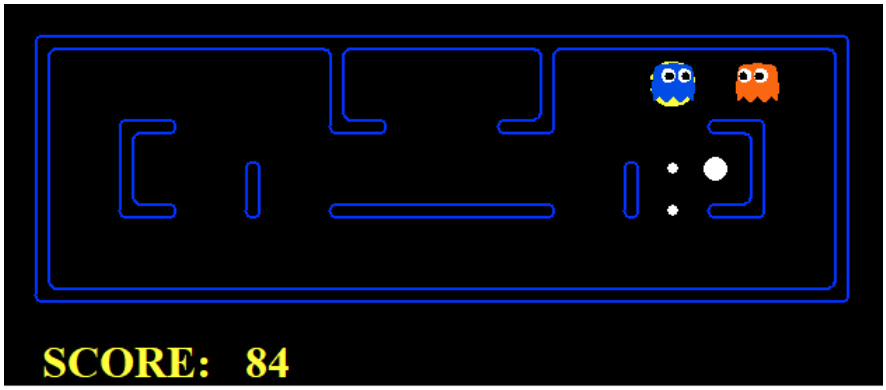

Figure 8: Pacman lost the game

```
*** 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 18 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q3\8-pacman-game.test

### Question q3: 5/5 ###
```

Figure 9: Results of Question three's autograder

It is clear that the Alpha-Beta pruning gave the same results as the minimax algorithm.

2.3 Expectimax

The *Expectimax algorithm*, unlike the minimax algorithm considers that the opponent is not optimal. It assumes that the agent will not always choose the best action. Since the case here is that the ghosts are *random*, the Expectimax won about half of the games. Pacman is considered **Max**, while each one of the ghosts is considered as a **chance node**.

2.3.1 Algorithm approach

In the *expectimax* function, we have the same **"max_value"** function as in the minimax algorithm. However, instead of the **"min_value"** function, we have a **"chance_node"** function that computes the expected values of evaluation scores rather than the minimum value. The **"expectimax"** function returns the best possible action for Pacman.

2.3.2 Results

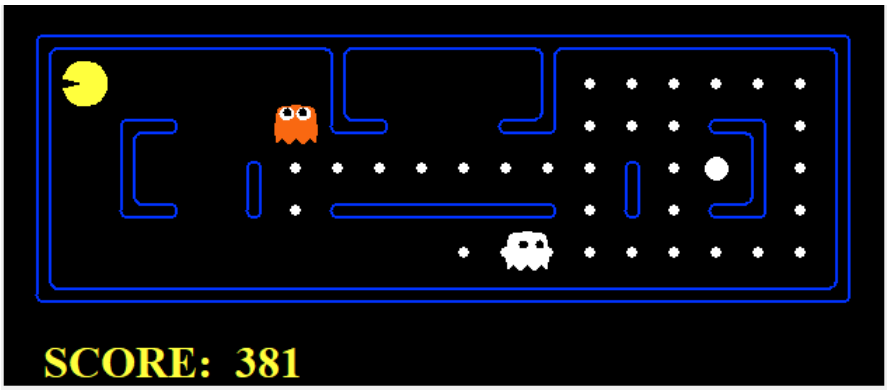

Figure 10: Pacman's behaviour according to expectimax algorithm

```
Pacman died! Score: -502
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Pacman emerges victorious! Score: 532
Pacman died! Score: -502
Pacman emerges victorious! Score: 532
Average Score: 15.0
Scores:        -502.0, -502.0, 532.0, -502.0, 532.0, -502.0, 532.0, 532.0, -502.0, 532.0
Win Rate:      5/10 (0.50)
Record:        Loss, Loss, Win, Loss, Win, Loss, Win, Win, Loss, Win
```

Figure 11: Trapped Classic game with depth = 3

Pacman won five games out of ten. Since expectimax algorithm assumes the ghosts to be random, which is true.

```
*** 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 18 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q4\7-pacman-game.test

### Question q4: 5/5 ###
```

Figure 12: Results of Question four's autograder

3 Evaluation Function

3.1 Approach

Here, it is required to implement an Evaluation Function that evaluates a certain state, unlike the reflex agent. Here, Pacman's goal is to eat all the food, without being eaten from ghosts, and get an average score of 1000. As a result, the evaluation function was was the same as for the reflex agent, with some changes.

The evaluation score was boosted as the distance to the nearest ghost increased, only if it was not scared and reduced as the distance to the nearest food increased. However, if a ghost was scared, the evaluation score was reduced as the distance to this ghost increased. Moreover, the evaluation score was reduced as the distance between Pacman and a capsule increased. Finally, again trial and error approach was used to get the weights. The function returned the summation of the score and evaluation score.

To sum up, the Evaluation Function was designed to make Pacman run away from non-scared ghosts and chase scared ghosts, food, and capsules.

```
if newScaredTimes == 0:  # Get away from ghosts only if they're not scared
    eval_score -= 40 * 1 / min(pacman_ghosts_distances)
else:
    eval_score += 40 * 1 / min(pacman_ghosts_distances)

eval_score += 10 * 1 / min(pacman_food_distances)

if len(newCapsules) != 0:
    eval_score += 10 * 1 / min(pacman_capsules_distances)
```

3.2 Results

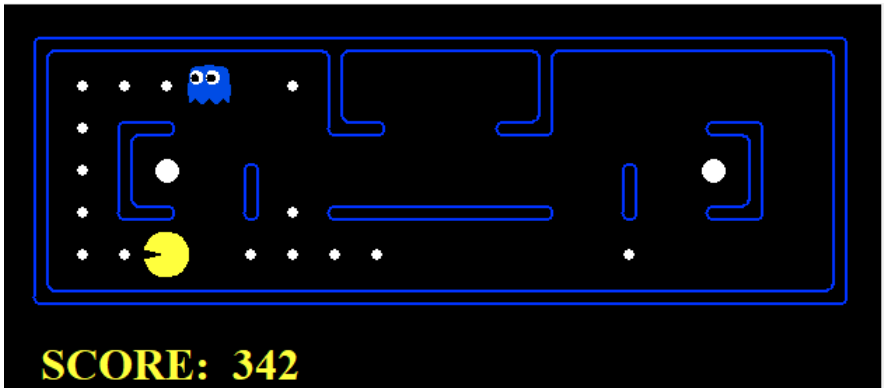

Figure 13: Pacman's behaviour according to the Evaluation Function

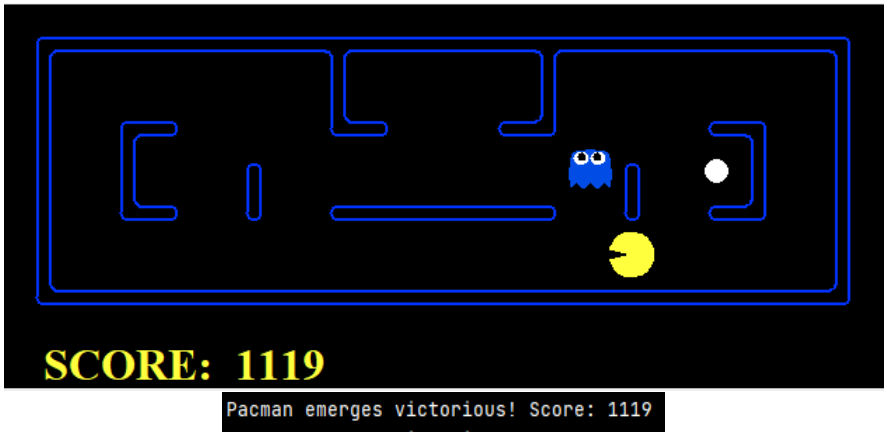

Figure 14: Pacman won with a score of 1119

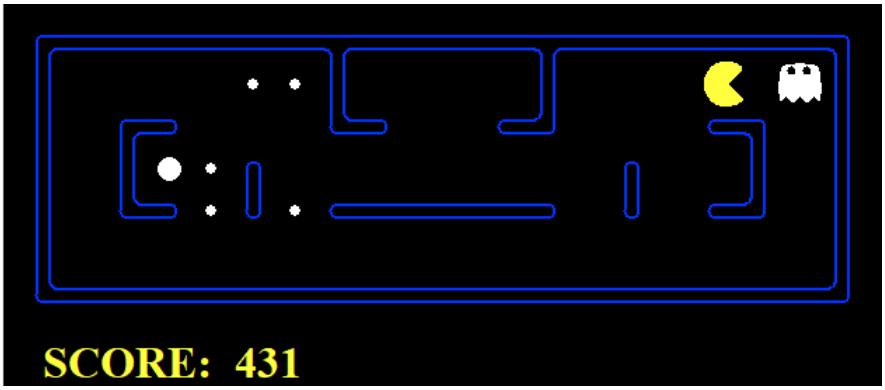

Figure 15: Pacman chasing a scared ghost

```
Question q5
===========

Pacman emerges victorious! Score: 1305
Pacman emerges victorious! Score: 1275
Pacman emerges victorious! Score: 1119
Pacman emerges victorious! Score: 977
Pacman emerges victorious! Score: 1156
Pacman emerges victorious! Score: 1127
Pacman emerges victorious! Score: 1172
Pacman emerges victorious! Score: 1013
Pacman emerges victorious! Score: 975
Pacman emerges victorious! Score: 1334
Average Score: 1145.3
Scores:        1305.0, 1275.0, 1119.0, 977.0, 1156.0, 1127.0, 1172.0, 1013.0, 975.0, 1334.0
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** PASS: test_cases\q5\grade-agent.test (6 of 6 points)
***     1145.3 average score (2 of 2 points)
```

Figure 16: Results of Question five's autograder

References

[1] "Project 2 - Multi-Agent Search - CS 188: Introduction to Artificial Intelligence, Spring 2020", Inst.eecs.berkeley.edu, 2020. [Online]. Available: https://inst.eecs.berkeley.edu/ cs188/sp20/project2/#question-2-5-points-minimax. [Accessed: 22- Dec- 2020].