deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Programming Elements

Lab work No. 3

Jodionísio Muachifi «97147»

Miguel Simões «118200»

Gustavo Reggio «118485»

Group No. 1

Mestrado em Engenharia de Computadores e Telemática

Mestrado em Engenharia de Automação Industrial

Professor: Armando J. Pinho

January 3, 2024

Link do repositório do Projeto 3

Conteúdo

Figuras

Abbreviations

MECT – Mestrado em Engenharia de Computadores e Telemática

MEAI – Mestrado em Engenharia de Automação Industrial

PD – Programação defensiva

TU – Teste Unitário

MSE – Mean Squared Error

NN – Neural Network

BP – Backpropagation

TTT – Tic-Tac-Toe

CLI – Command Line Interface

Chapter 1

1 Introduction

Tic-Tac-Toe (TTT), a classic game known for its simplicity and strategic depth, serves as the focal point of this project. The objective is to implement a command-line interface for playing TTT, incorporating features such as displaying the current game state, validating player moves, determining winners, and starting games from predefined states. Additionally, the project extends to include the option of playing against a random agent, with the seed for the pseudo-random generator provided as a command-line argument. To further elevate the challenge, the report details the integration of neural network functionalities to train an agent, providing an advanced opponent for TTT enthusiasts.

This report comprehensively outlines the steps and decisions made throughout the implementation of each part of this project, shedding light on each step choices and considerations during development. The integration of a neural network into the TTT playing agent is a notable highlight, showcasing the application of artificial intelligence (i.e., Machine Learning) in game development.

In addition to what was requested in the project statement, we have added an additional option that allows both players to compete randomly. At the end of the game, we present some statistics to see who has the most matches and the probability of each player winning. We have also reused some of the code developed in the previous project, mainly the ***base.h*** and ***base.c*** files.

We have developed the source code using ***C language***, and it's available in our GitHub repository with detailed instructions on how to execute it. This will provide easy access for reference and use. The graphics in this report were created using Python and Matlab programming languages. Additionally, we have included a ***TTT*** Python version in the project to compare its performance between the C and Python versions and test its functionality.

Chapter 2

2 Part I

The main purpose of the exercises developed in this part was to apply and integrate the material we studied in the theory classes, applying fundamental concepts about pointers, neural networks, backpropagation calculation and the concept of TTT itself.

2.1 Exercise No. 1 - Command-Line Interface Implementation

In this exercise, we delve into the specific steps taken to implement the CLI, addressing challenges encountered and decisions made during the process. The goal is to create a robust and efficient interface that seamlessly integrates with the subsequent components of the project, providing a solid basis for an interactive and engaging TTT experience.

The primary objectives include creating a user-friendly environment that allows players to engage in the game flawlessly. The implemented CLI could display the current state of the TTT board, soliciting and validating moves from the current player, determining if there is a winner, and facilitating the initiation of the game from a predefined state specified as a command-line argument as illustrated in Figure 1.

Just to recall, for this first version of the game, the rule was to play with two players, X and Y, both controlled by human players (we called it by ***dual***). However, we opted to use 'Y' instead of 'O' due to conflicts with the variables (O for output layer) defined in ***base.h*** and ***base.c*** when developing our neural network agent.

We implemented essential input validation during the game. For instance, if player X attempts to place their marker in a square that is already occupied, a message will be displayed: Square is already occupied. Please try again as illustrated in Figure 2.

To run this exercise (program), you need to follow the instructions below and make sure you are in the partI directory:

```
user@host: partI$ make
user@host: partI$ ./game_play ttt dual
```

```
cloudx@DESKTOP-ERKH3KR:~/EP/projects/work3/ttt-prod$ ./game_play ttt
 Usage: ./game_play ttt <mode>
Memo: <mode> can be  dual  or  rand  or  nnet  or  auto_rand
cloudx@DESKTOP-ERKH3KR:~/EP/projects/work3/ttt-prod$ ./game_play ttt dual
Received state: .........
Turn: X, Moves: 0
  1 2 3
A . . .
B . . .
C . . .

 It is X's turn. 0 moves have been made.
Enter a move (e.g. A1):
```

Figure 1 - Game Usage (dual playing)

Player Starts

```
Received state: .........
Turn: X, Moves: 0
  1 2 3
A . . .
B . . .
C . . .

 It is X's turn. 0 moves have been made.
Enter a move (e.g. A1): A1
  1 2 3
A X . .
B . . .
C . . .

 It is Y's turn. 1 moves have been made.
Enter a move (e.g. A1): C3
  1 2 3
A X . .
B . . .
C . . Y

 It is X's turn. 2 moves have been made.
Enter a move (e.g. A1): A2
  1 2 3
A X X .
B . . .
C . . Y
```

Player Ends (Wins)

```
 It is Y's turn. 3 moves have been made.
Enter a move (e.g. A1): C3
 Square is already occupied. Try again.
Enter a move (e.g. A1): C1
  1 2 3
A X X .
B . . .
C Y . Y

 It is X's turn. 4 moves have been made.
Enter a move (e.g. A1): A3
  1 2 3
A X X X
B . . .
C Y . Y

 It is Y's turn. 5 moves have been made.
 X is the winner.
```

Figure 2 - Game Play (dual players)

As depicted in Figure 2, the format TTT differs from the familiar convention. We have intentionally adopted this format to differentiate from the common standard used.

To implement this exercise, we created a function named ***play_dual(char *state)*** in our ***game_play.c*** file. Within this function, we saved all game states to a file. The file is referenced in red as the second argument of the ***save_game_numbers(&game, "states_dsets.ttt")*** function. Later on, we shall describe in detail about this last function, when we talk about neural network agent.

2.2 Exercise No. 2 - Playing Against a Random Agent

The challenge lies in adapting the existing CLI to accommodate interactions with both human players and the random agent seamlessly. The development involves enhancements to the game logic to support automated moves and a strategic design to make the interactions engaging for the human player. By the end of this exercise, the CLI offered us a versatile gaming experience, allowing users to choose between human-human, human-random agent.

In order to perform this exercise, we developed a function called ***play_rand(char *state)*** in our ***game_play.c*** file. Within this function, we stored all game states to a file, which is referred to in red as the second argument of the ***save_game_numbers(&game, "states_dsets.ttt")*** function. We will describe this last function in detail later on, when we talk about neural network agent.

The most important feature of this function (***play_rand***) is illustrated in Figure 3, where the movements across columns and rows of the board are obtained randomly, i.e., the computer works as player Y in this case.

```
// Check if it is the human player's turn
if (game.turn == X)
{
    // Get the move from the human player
    get_move(&game, &row, &col);
}
else
{
    // Get a random move from the computer player
    do
    {
        row = rand() % SIZE;
        col = rand() % SIZE;
    } while (game.board[row][col] != EMPTY);
    printf("\033[34m The computer plays %c%d. \033[0m\n", 'A' + row, col + 1);
}

// Make the move
make_move(&game, row, col);
```

Figure 3 - Function play_rand (important feature)

To run this exercise (program), you need to follow the instructions below and make sure you are in the partI directory:

```
user@host: partI$ make
user@host: partI$ ./game_play ttt rand
```

Player Starts

```
Received state: .........
Turn: X, Moves: 0
  1 2 3
A . . .
B . . .
C . . .

 It is X's turn. 0 moves have been made.
Enter a move (e.g. A1): A1
  1 2 3
A X . .
B . . .
C . . .

 It is Y's turn. 1 moves have been made.
 The computer plays C3.
  1 2 3
A X . .
B . . .
C . . Y
```

Player Ends (Wins)

```
 It is X's turn. 2 moves have been made.
Enter a move (e.g. A1): C1
  1 2 3
A X . .
B . . .
C X . Y

 It is Y's turn. 3 moves have been made.
 The computer plays C2.
  1 2 3
A X . .
B . . .
C X Y Y

 It is X's turn. 4 moves have been made.
Enter a move (e.g. A1): B1
  1 2 3
A X . .
B X . .
C X Y Y

 It is Y's turn. 5 moves have been made.
 X is the winner.
```

Figure 4 - Game Play (player X and rand player Y)

In summary, in this section, we delve into the considerations and decisions made while integrating the random agent functionality to ensure a balanced and enjoyable gaming experience for users. Furthermore, we observed insights into the impact of the random agent player (PC) when interacting with a human player. As depicted in Figure 4, the human player typically emerges as the winner in such scenario.

We have tried this scenario several times and found that the player X consistently wins or has a higher probability of winning when playing against the random player Y. As mentioned in this section, we also added a feature to calculate the probability of each player, which is illustrated in Figure 5.

```
 It is X's turn. 4 moves have been made.
Enter a move (e.g. A1): A3
  1 2 3
A X . X
B . . Y
C Y . X

 It is Y's turn. 5 moves have been made.
 The computer plays C2.
  1 2 3
A X . X
B . . Y
C Y Y X

 It is X's turn. 6 moves have been made.
Enter a move (e.g. A1): B2
  1 2 3
A X . X
B . X Y
C Y Y X

 It is Y's turn. 7 moves have been made.
 X is the winner.

                --- Statistics Table ---
| Total Games | Human Player X Wins | Rand Player Y Wins     | Draws |
|-------------|---------------------|------------------------|-------|
| 7           | 4        P = 57.14% | 3          P = 42.86% | 0     |
-------------------------------------------------------------------
```

Figure 5 - Total Game Play (probability of wins for each player)

2.3 Exercise No. 3 - Training an Intelligent Agent with Neural Networks

This section of the project marks a significant advancement by leveraging neural network functionalities to train an intelligent agent for tic-tac-toe. Building upon the groundwork laid in the previous exercises, this section explores the intersection of machine learning and game-playing strategies. The goal is to create an agent capable of making informed and strategic moves, challenging human players with a higher level of sophistication.

To implement this part of the project, we had to create the functions described below:

1. Function ***nn_play()***: here is where we initialize the neural network, train it, save it to a file, and call a function to play the game, ***play_agent(nn)***. Also, to make it work properly, we had to create a separate agent file called ***nn_agent.h*** and ***nn_agent.c***.
2. Function ***save_game_numbers(game_t *game, char *filename)***: in section 2.1 and 2.2, we referred to an important function that allows us to map the board with X representing 1, Y representing -1, and EMPTY representing 0. We have also stored the probability into the file states_dsets.ttt, the datasets where all states are saved from dual and rand player.
3. Function ***calculate_probability(game_t *game)***: this function allows to calculate the probability basing in number of moves.
4. Function ***update_statistics(game_t *game, double duration)***: this function permits to calculate some statistics like ***average duration***, ***longest game duration***, ***fastest win duration***.

While we have developed various functions, this report focuses on describing key features within select important functions. These features are integral to the functions discussed earlier. In Figures 6 and 7, bar charts and graphs depict the probability per game and time during the game.

To run this exercise (program), you need to follow the instructions below and make sure you are in the partI directory:

```
user@host: partI$ make
user@host: partI$ ./game_play ttt nnet
```

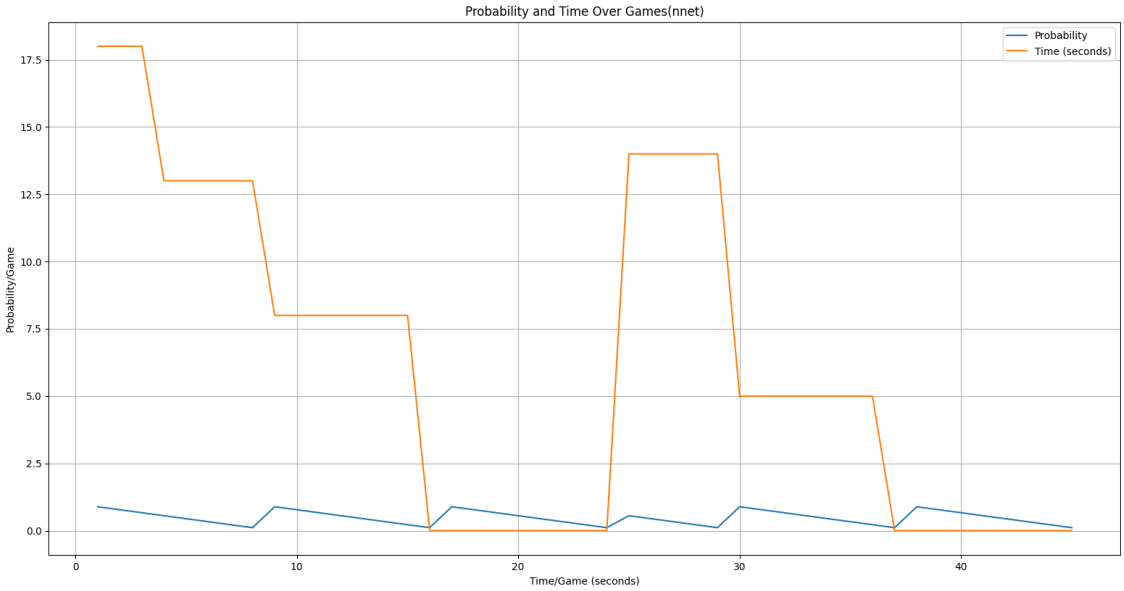

Figure 6 - Probability and Time Over Games (nnet)

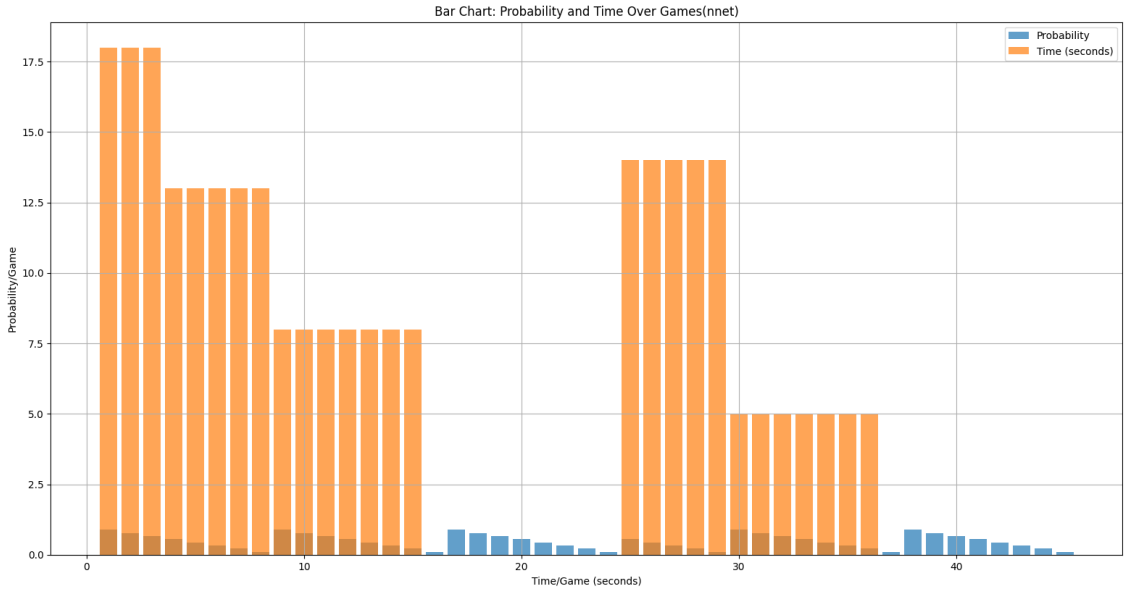

Figure 7 - Bar Chart for Probability and Time Over Games(nnet)

In Figure 8, we illustrate the playing steps of the NN agent (Player X and Player Y). We adjusted crucial features such as Seed, I, H, O, and learning rate to ensure seamless functioning.

Player Starts

```
.........Received state: .........
Turn: X, Moves: 0
  1 2 3
A . . .
B . . .
C . . .

 It is X's turn. 0 moves have been made.
  1 2 3
A . . .
B . . .
C . X .

 It is Y's turn. 1 moves have been made.
Enter a move (e.g. A1): C2
 Square is already occupied. Try again.
Enter a move (e.g. A1): C1
  1 2 3
A . . .
B . . .
C Y X .

 It is X's turn. 2 moves have been made.
  1 2 3
A . . .
B . . .
C Y X X

 It is Y's turn. 3 moves have been made.
Enter a move (e.g. A1): A1
  1 2 3
A Y . .
B . . .
C Y X X
```

Player Ends (Wins)

```
 It is X's turn. 4 moves have been made.
  1 2 3
A Y . .
B X . .
C Y X X

 It is Y's turn. 5 moves have been made.
Enter a move (e.g. A1): B2
  1 2 3
A Y . .
B X Y .
C Y X X

 It is X's turn. 6 moves have been made.
  1 2 3
A Y . .
B X Y X
C Y X X

 It is Y's turn. 7 moves have been made.
Enter a move (e.g. A1): A2
  1 2 3
A Y Y .
B X Y X
C Y X X

 It is X's turn. 8 moves have been made.
  1 2 3
A Y Y X
B X Y X
C Y X X

 It is Y's turn. 9 moves have been made.
 Player X wins with a probability of 0.888889!
```

Figure 8 - Game Play (Player X nnet and Player Y Human)

To summarize, this exercise involves implementing the training agent, selecting appropriate neural network architectures, and fine-tuning (hyper)parameters. It was essential to balance the agent's learning progress to avoid overfitting or underfitting to achieve optimal performance. In this section, we provide in-depth insights into the technical aspects of integrating neural networks into the TTT game. We discuss the methodology for training the agent, the challenges faced during the process, and the decisions made to enhance the agent's learning capabilities.

As shown in Figure 8, the chance of Player X winning is currently at 0.888889. This is an essential aspect that we have previously discussed in this report. The probability calculation is in line with the methodology we have explained in the preceding section. It gives a complete insight into how these probabilities are calculated from the neural network functions that we have implemented.

2.4 Exercise No. 4 - Playing Using a Random Agent for Both Players

This section introduces a distinctive gameplay scenario by pitting two random agents against each other in a tic-tac-toe match. Unlike previous exercises involving human players or a human player against a random agent or human players against NN agent, this exercise explores the dynamics of a fully automated game where both players' moves are determined randomly.

In this exercise, we have added a new feature that allows players to compete against each other and calculate their probability of winning at the end. We have created a function called ***auto_rand()*** which enables players to compete randomly against each other, as shown in Figure 9. The player who starts first has a chance of winning. Moreover, we have added some additional statistics, such as ***Average Game Duration***, ***Longest Game Duration***, and ***Average Moves Per Game***, and we have plotted them graphically as presented in Figures 10 and 11, respectively.

Player Starts

```
Received state: .........
Turn: X, Moves: 0
  1 2 3
A . . .
B . . .
C . . .

 It is X's turn. 0 moves have been made.
 Player X plays A2.
  1 2 3
A . X .
B . . .
C . . .

 It is Y's turn. 1 moves have been made.
 Player Y plays B1.
  1 2 3
A . X .
B Y . .
C . . .

 It is X's turn. 2 moves have been made.
 Player X plays B2.
  1 2 3
A . X .
B Y X .
C . . .

 It is Y's turn. 3 moves have been made.
 Player Y plays B3.
  1 2 3
A . X .
B Y X Y
C . . .
```

Player Ends (Wins)

```
 It is X's turn. 4 moves have been made.
 Player X plays C3.
  1 2 3
A . X Y
B Y . .
C . X X

 It is Y's turn. 5 moves have been made.
 Player Y plays A1.
  1 2 3
A Y X Y
B Y . .
C . X X

 It is X's turn. 6 moves have been made.
 Player X plays B2.
  1 2 3
A Y X Y
B Y X .
C . X X

 It is Y's turn. 7 moves have been made.
 X is the winner.

                --- Statistics Table ---
| Total Games | Rand Player X Wins   | Rand Player Y Wins     | Draws |
|-------------|----------------------|------------------------|-------|
| 50          | 33       P = 66.00%  | 12          P = 24.00% | 5     |
-----------------------------------------------------------------------

      === Aditional Statistics ===
Fastest Win Duration: No wins yet
Average Game Duration: 24.50 seconds
Longest Game Duration: 49.00 seconds (in 7 moves)
Average Moves Per Game: 3.00
=======================================================================
```

Figure 9 - Game Play Auto Random (Player X vs Player Y)

To run this exercise (program), you need to follow the instructions below and make sure you are in the partI directory:

```
user@host: partI$ make
user@host: partI$ ./game_play ttt auto_rand
```

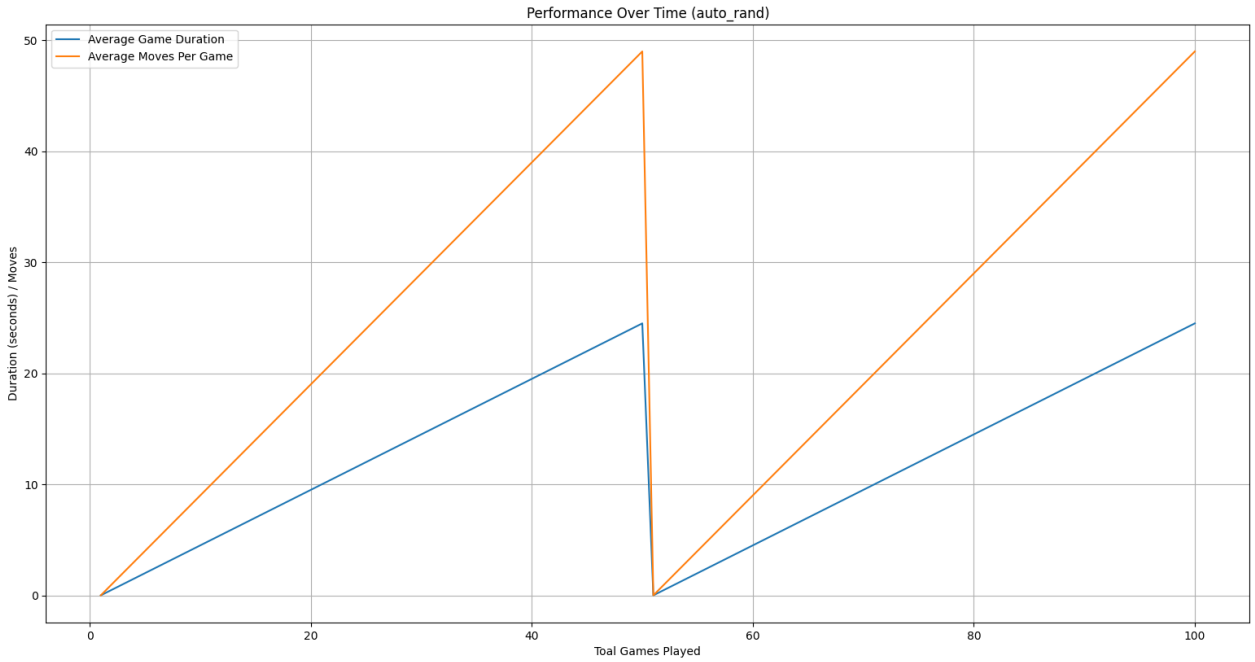

Figure 10 - Performance of Two Players Over Time (auto_rand)

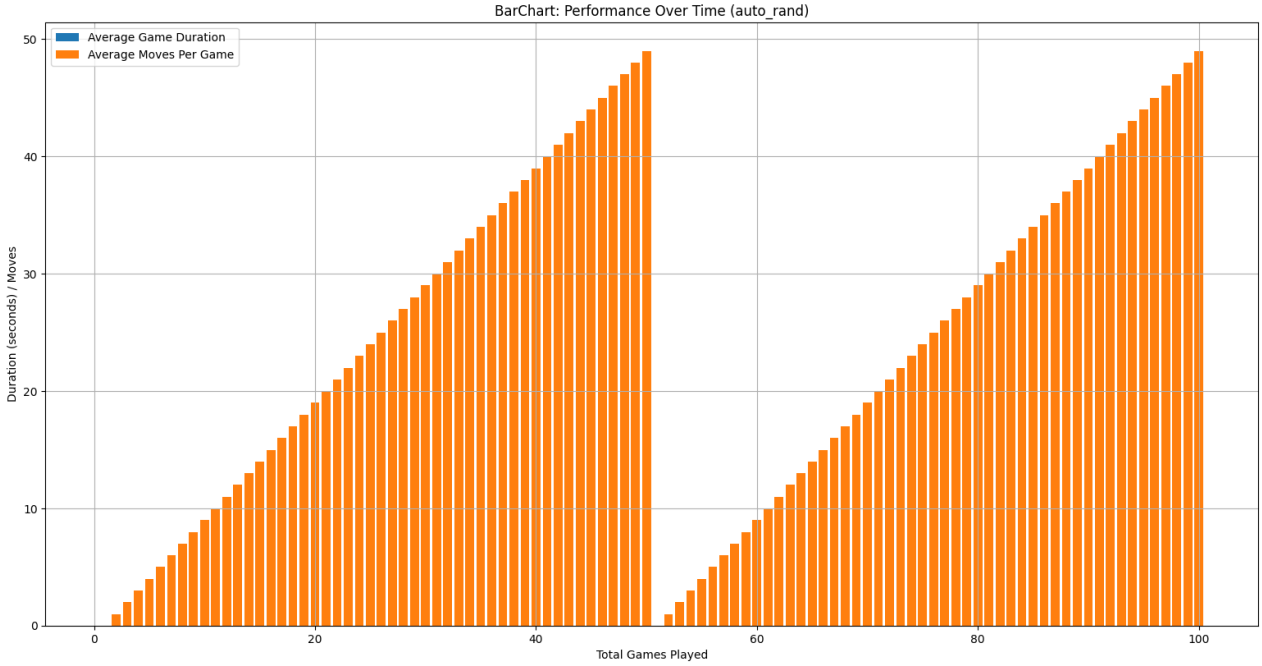

Figure 11 - BarChart Performance of Two Players Over Time (auto_rand)

In summary, this section introduces a dynamic aspect to the project by allowing for gameplay between random agents. Statistical metrics and graphical representations are included to help us better understand the game's performance during the competition. This is crucial to ensure a higher probability of winning for the player who starts first.

Chapter 3

3 Authors' contribution

The participation of each author was excellent with regard to the research, discussion and writing of this work, highlighting the effort of each member of the group with regard to the development of the code. The percentage contribution for each student is as follows:

- Jodionísio Muachifi – 40%
- Miguel Simões – 32%
- Gustavo Reggio – 28%

Chapter 4

4 Conclusion

In conclusion, the collaborative exercises not only demonstrate the flexibility of the implemented Command Line Interface (CLI) for different gameplay scenarios but also showcase the project's evolution from traditional gameplay to advanced machine learning applications. The integration of neural networks and experimentation with automated matchups provide a comprehensive understanding of the dynamics of tic-tac-toe, paving the way to extend the features implemented on this project.

During the development of our neural network agent, we encountered some issues. The agent was filling the board line by line, causing problems when the other player placed their mark. We resolved this issue by randomly going through the rows and columns, calculating the ***bestscore*** and comparing it to the probabilities found in the dataset. Ultimately, the player wins with the highest probability.

Referências bibliográficas

- https://www.w3schools.com/c/c_structs.php
- https://en.wikipedia.org/wiki/Pointer_(computer_programming)
- https://www.youtube.com/watch?v=Ilg3gGewQ5U
- https://elearning.ua.pt/mod/url/view.php?id=1336755
- https://en.wikipedia.org/wiki/Backpropagation
- https://en.wikipedia.org/wiki/Neural_network
- https://en.wikipedia.org/wiki/GNU_Debugger
- https://en.wikipedia.org/wiki/Valgrind