Project 3

Pedro Henrique Batista Nunes
cc22147@g.unicamp.br

Victor Hugo Barbosa
cc22152@g.unicamp.br

TI326 - Artificial Intelligence
COTUCA - Unicamp

19 de outubro de 2024

1 Introduction

Tic-Tac-Toe originated in Egypt, with 3x3 game boards found around 1300 BCE. This two-player game involves marking cells on a grid to form a line vertically, horizontally, or diagonally. If no player wins and all cells are filled, the game is a draw.

We will solve this game using the Minimax algorithm, a technique for decision-making in adversarial environments, maximizing gains while minimizing the opponent's advantages.

1.1 Computational Experiments

Experiments were run on a macOS Sonoma 14.3 with an Apple M1 chip and 16 GB RAM, using Python 3.11 in Visual Studio Code.

2 Test Instances

The moves sequences will be described taking into account the following mapping of the 3x3 board:

1	2	3
4	5	6
7	8	9

2.1 Results

The results of our computational experiments are summarized in the table of computational results bellow.

- **Instances:** The identification number of the test
- **Status:** Refers to the current situation of the algorithm
- **X:** Refers to who (AI or Player) plays as "X", that is, who plays first, as X always makes the initial move
- **Plays:** The sequence of moves made by the Player
- **Responses:** AI's reactions and responses
- **Result:** The result of the game ("Player"for the player wins, "AI"for the AI wins, and "draw"for a draw)

Instance	*Status*	*X*	*Plays*	*Responses*	*Result*
01	Initial Code	Player	5-3	1	-
02	Initial Code	AI	7	1	-
03	First Minimax Algorithm	Player	1-5	9-8-7-6-4-3-2	AI
04	Minimax Almost Done	Player	7-5	9-8	Player
05	Minimax finished	Player	5-2-6-7-9	1-8-4-3	Draw
06	Minimax finished	Player	3-9-4-7-2	5-6-1-8	Draw
07	Minimax finished	Player	7-4-9-2-6	5-1-8-3	Draw
08	Minimax finished	IA	3-7-9	1-4-5-6	AI
09	Minimax finished	IA	5-3-4-8	1-2-7-6-9	Draw
10	Minimax finished	IA	9-2-5	1-3-7-4	AI

2.2 Results Explanation

- **Instances 01 and 02:** The first tests of the initial code came to nothing, as the minimax algorithm had not yet been implemented, it only had initial variables

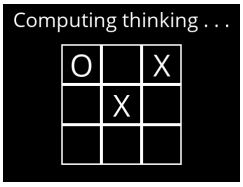

Figura 1: Test 01

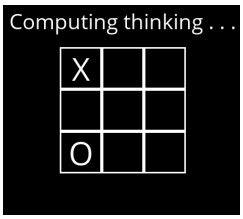

Figura 2: Test 02

- **Instance 03:** The first version of the minimax algorithm did not switch between Min and Max, it chose only one of the two, causing it to return all movements

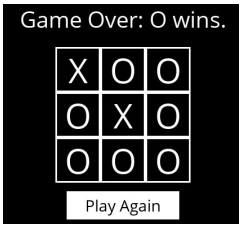

Figura 3: Test 03

- **Instance 04:** In the second version of minimax, where it was almost ready, it was not making the best moves, as an error in the algorithm did not store the best score, preventing the best moves from being calculated.

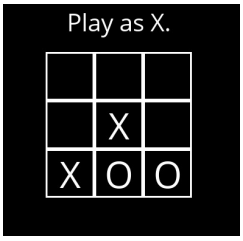

Figura 4: Test 04

- **Instances 05, 06 and 07:** The third and final version of the algorithm worked perfectly, the best moves were always chosen. It is also possible to observe that when the player

makes the first move (that is, chooses to play as X) it is more likely that the game will end in a draw.

Figura 5: Test 05

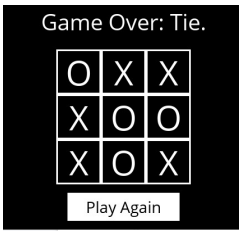

Figura 6: Test 06

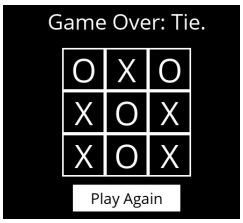

Figura 7: Test 07

- **Instances 08, 09 and 10:** Still demonstrating that the algorithm works, it is clear that when the player chooses to play as O, the AI's chances of winning increase

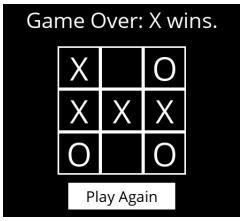

Figura 8: Test 08

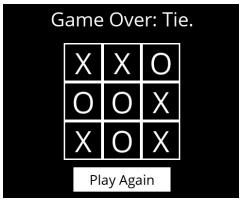

Figura 9: Test 09

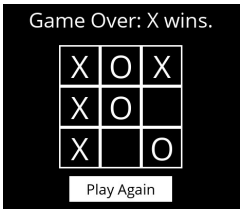

Figura 10: Test 10

3 The Algorithm

3.1 Minimax

As mentioned earlier, the Minimax algorithm was implemented to solve tic-tac-toe. Minimax is a specific algorithm within the domain of Adversarial Search and cannot be used in every game, as an Adversarial environment is required, that is, with 2 players, in turns, with perfect information (each player has complete knowledge of the possible moves) and zero sum (one player's victory is his opponent's defeat). Therefore, Minimax is designed to determine the best possible play for a player, assuming the opponent is also playing optimally to minimize the original player's gain. The algorithm works in conjunction with a simple evaluation function that evaluates possibilities based on states, building a decision tree that represents all possible future moves, alternating between players' turns.

- **Step 1 - Game Tree Construction:** Each node in the tree represents a possible state of the game. The alternating levels of the tree represent the players' turns (Max and Min).
- **Step 2 - Assignment of Values:** At the terminal nodes (leaves of the tree), a value is assigned based on an evaluation function that determines how favorable that state is for the player Max.
- **Step 3 - Spread of Values:**
  - **Max Levels:** The algorithm chooses the maximum value among the children, representing the best move for Max.
  - **Min Levels:** The algorithm chooses the minimum value among the children, representing the best move for Min (minimizing the gain for Max).
- **Step 4 - Final Decision:** From the root of the tree (current state of the game), Minimax selects the move that leads to the best possible value, considering the opponent's optimized response.