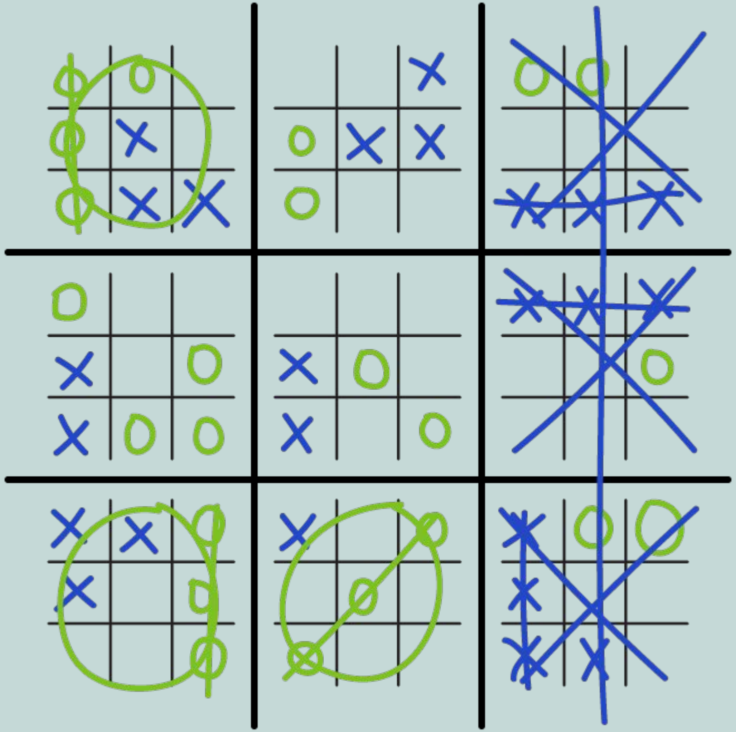
Artificial Intelligence
1st Project Checkpoint

Bruno Drumond - up201202666
Tomás Sucena Lopes - up202108701

Ultimate Tic-Tac-Toe



Overview

A variant of the popular Tic-Tac-Toe game consisting of a 3x3 grid of Tic-Tac-Toe boards. Players take turns playing on the smaller boards until one wins in the larger board.

Goal

Implement the game, as well as **computer players** that can play it with varying levels of expertise.

Problem Formulation

State

Since the board is a grid of smaller boards, it can be represented as a multidimensional array.
We opted for a 2D string array with three values: ‘X’, ‘O’, or the empty string.

Initial State

The 2D array is filled with empty strings.

Objective Test

Verify if any of the players has won the big board (**win**) or if there are no legal moves remaining (**draw**).

Operators

Preconditions

The rules of Tic-Tac-Toe apply. As such, the players:

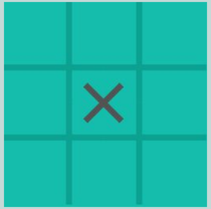
- Take turns placing their marks on empty tiles.
- Can only play in the small board dictated by their opponent’s last move.
  - Unless it has already been won, in which case the player can choose.

Cost

All moves have the same cost.

Heuristics

The standard heuristics from Tic-Tac-Toe apply to both the big and small boards.

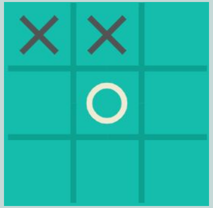


Middle

Capturing the middle board presents the most victory opportunities.

Blocking

Preventing the opponent from making advantageous moves should be rewarded.



2 out of 3

Capturing two boards in a winning pattern that is not blocked creates a strong threat.

Traps

Forcing the opponent to play in boards where they have a weak presence is crucial.

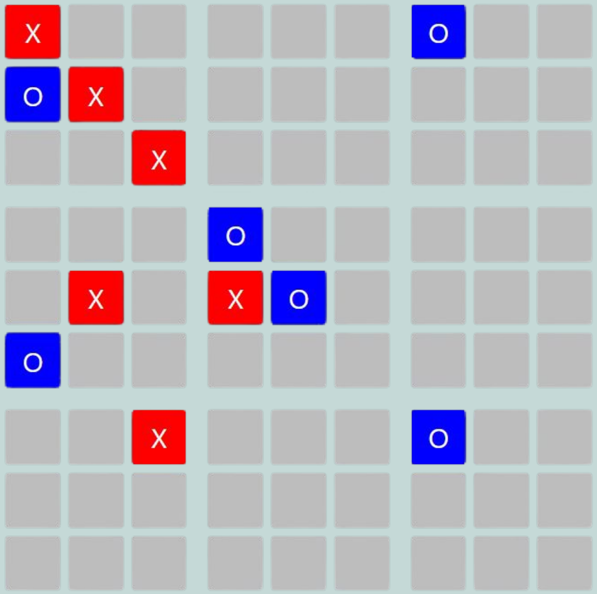
Checkpoint Progress

Technologies

- Opted for React with TypeScript.

Game

- Implemented the game state.
    - Used a 2D string array to represent the tiles, as well as a string array for the big board.
    - Implemented valid moves, turns, win conditions, etc.
- Designed the gameplay UI.
    - Designed the big board, small boards, and tiles.
    - Added visual hints to aid the players in understanding the game state.



Updated Progress

Game

- Add two game menus.
- Allow customization of the board size.
- Improved the game GUI.

Gameplay

- Add the possibility to play against the AI through the implementation of minimax algorithm with alpha-beta pruning.
    - Deterministic decision-making suitable for perfect-information games.
    - Good performance through efficient pruning of unnecessary search paths
    - Scalable difficulty through depth adjustment
- Player can choose from 4 possible difficulties (very easy, easy, medium and hard).
- The game state is saved through each play.

Heuristics

The AI uses a composite evaluation function that considers:

Board Value Assessment

- Win/Loss detection.
- Best potential winning moves.
  - Center and corner control

Strategic Position Evaluation

- Limiting opponent options by choosing the best moves.
- Avoiding moves that send the opponent to advantageous boards.

Move Depth Consideration

- Quicker wins are prioritized over delayed wins.
- Delayed losses are preferred over immediate losses.

Experimental Results

Difficulty	Algorithm	Depth	Heuristics	Player Wins	CPU Wins	Draws	CPU Win Rate	Average Decision Time
Very Easy	Randomizer	NA	None	5	0	0	0%	2 ms
Easy	Minimax with alpha-beta pruning	2	All	1	3	1	60%	85 ms
Medium	Minimax with alpha-beta pruning	5	All	0	4	1	80%	455 ms
Hard	Minimax with alpha-beta pruning	7	All	0	5	0	100%	1790 ms

These results were obtained by playing 5 games in each difficulty in a default 3x3 size board.

Conclusion

Minimax with Alpha-Beta Pruning successfully powers the AI

- Provides competitive play across varying difficulty levels.
- Enables real-time responsiveness up to a particular depth.

Heuristic Evaluation Function is the key for good AI gameplay

- Captures strategic elements like board control and threats.
- Balances offense and defense.

⚠️ Limitations

- Exponential growth of the search tree makes a full-depth search option (perfect play) computationally infeasible.
- Fixed heuristics with manually tuned weights may not generalize to all game situations and cannot adapt or learn from gameplay history.

Bibliography

The following are hyperlinks to the resources we consulted throughout this project:

Technologies

- React
  - Documentation
  - Tutorial
  - Tic-Tac-Toe Tutorial
- TypeScript
  - Documentation
  - Documentation for React

Game

- Ultimate Tic-Tac-Toe
  - Rules
  - Implementation
  - Heuristics

Algorithms

- Minimax (w/ Alpha-Beta Pruning)
  - Video