MICROSOFT ACEHACKER

MARS COLONIZATION PROGRAM

PROJECT : ENTERTAIN THE CREW

CONTRIBUTORS

SBG BOTS

SBG Bots

Bhumika Bhatia
Shristi Rauniyar
Banavath Gayathri
Delhi Technological University

INTRODUCTION

We have chosen the project **“Entertain the Crew”** which includes the games **“Tic Tac Toe”** and **“Misere Tic Tac Toe”**. These are very simple, popular and 2-player games. The board size is *nxn*. We have used a 3x3 square.

For Tic Tac Toe, in order to win the game a player has to put their three of the marks in a row - horizontally, vertically or diagonally on a grid.

Misere Tic Tac Toe is just the opposite. The aim is to make sure to not make a line of three. We have added a twist to it - both AI and the crew member will be playing X.

OVERVIEW

In the 3x3 Tic-Tac-Toe game, we have two variations:

1. **Human vs. Human**

   Here, two crew members play against each other. We have functionalities to restart the game, display the winner as well as display whichever crew member’s turn it is at that time. On the creative side, we also display random space facts to increase general knowledge of the crew.

2. **Human vs. Computer**

   Here, a crew member plays against the computer. The crew member can choose the level of difficulty (Easy/Medium/Difficult/Very difficult and Impossible) and the starting player. Since the opponent is an intelligent agent, we have provided an option where the crew member can ask for hints for the next move.

In the 3x3 Misere Tic-Tac-Toe game, we have one variation:

1. **Human vs. Computer**

   Here, a crew member plays against the computer. We have the] same choices as above, i.e. choice of starting player, two levels of difficulty (easy and impossible/hard) and hints that suggest the next move. The variation of impossible/hard depends on the starting player. If the starting agent is AI, the game is deemed impossible and otherwise, there is still a chance for the human to win if it plays first.

HIGH LEVEL DIAGRAM

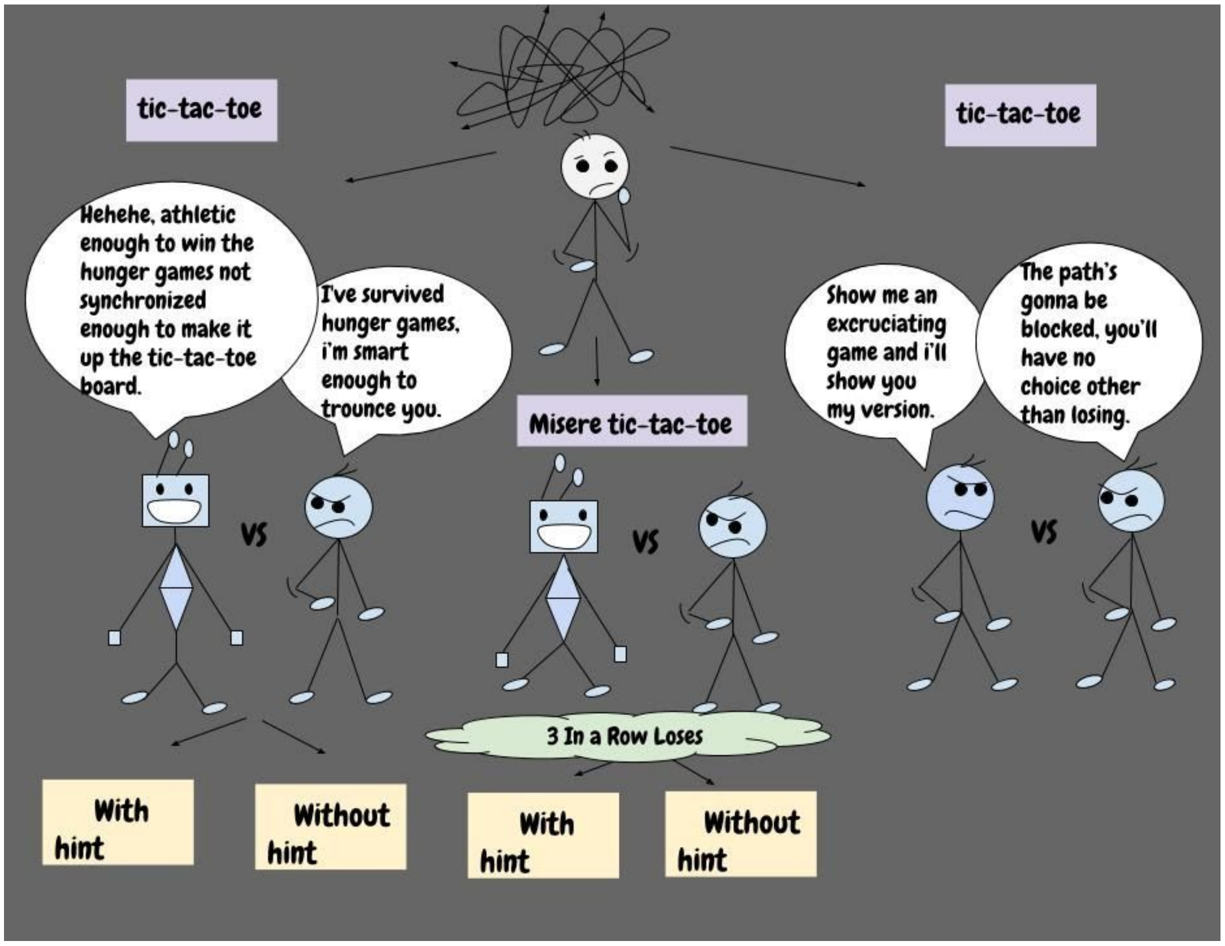

tic-tac-toe
tic-tac-toe
Hehehe, athletic enough to win the hunger games not synchronized enough to make it up the tic-tac-toe board.
I've survived hunger games, i'm smart enough to trounce you.
Misere tic-tac-toe
Show me an excruciating game and i'll show you my version.
The path's gonna be blocked, you'll have no choice other than losing.
VS
VS
VS
3 In a Row Loses
With hint
Without hint
With hint
Without hint

Fig: High Level Diagram of the WebApp

TECHNOLOGIES

BACK END

- JavaScript.
- Python for Web Scraping facts for The Fact Machine.

FRONT END

- Usage of p5.js which used the HTML Canvas to draw and render the board.
- Usage of Bulma, CSS for the user interface.
- Usage of Babel to generate compatible JavaScript code for browsers
- Usage of JQuery for functionality.
- Usage of Google Drawings to make the High Level Diagram, Low Level Diagram and Visual Representations of Minimax and Alpha Beta Algorithms

We have used **GitHub Pages** for the deployment of our Web Application.

The WebApp is available here.

PROGRAMMING APPROACH

We have made use of the **Object Oriented Programming** paradigm in all our algorithms across all the games. The **Agent Approach** has been implemented.

An example w.r.t. Human vs. AI Tic Tac Toe

Our **Software Agent** "***game***" is the **Agent** that acts upon the environment through **sensors** (for example: the action of a ***mouse click***) and **actuators** (for example: the ***find_move()*** function).

Game, i.e. the **Software Agent** receives sensory inputs from the click of the buttons (for example: selecting the ***Difficulty Level***, ***Starting Player*** & ***Choice of Hints***, and ***Start Game***) and clicking on the Game Board from the HTML Page. These sensory inputs invoke the **Agent Functions**.

The software agent invokes the ***make_board()*** **Agent Function**. ***make_board()*** invokes the **Class** ***TicTacToe()*** and **Agent Function** ***board()***.

TicTacToe() has many **Agent Functions** that have their respective **Agent Programs** as listed below:

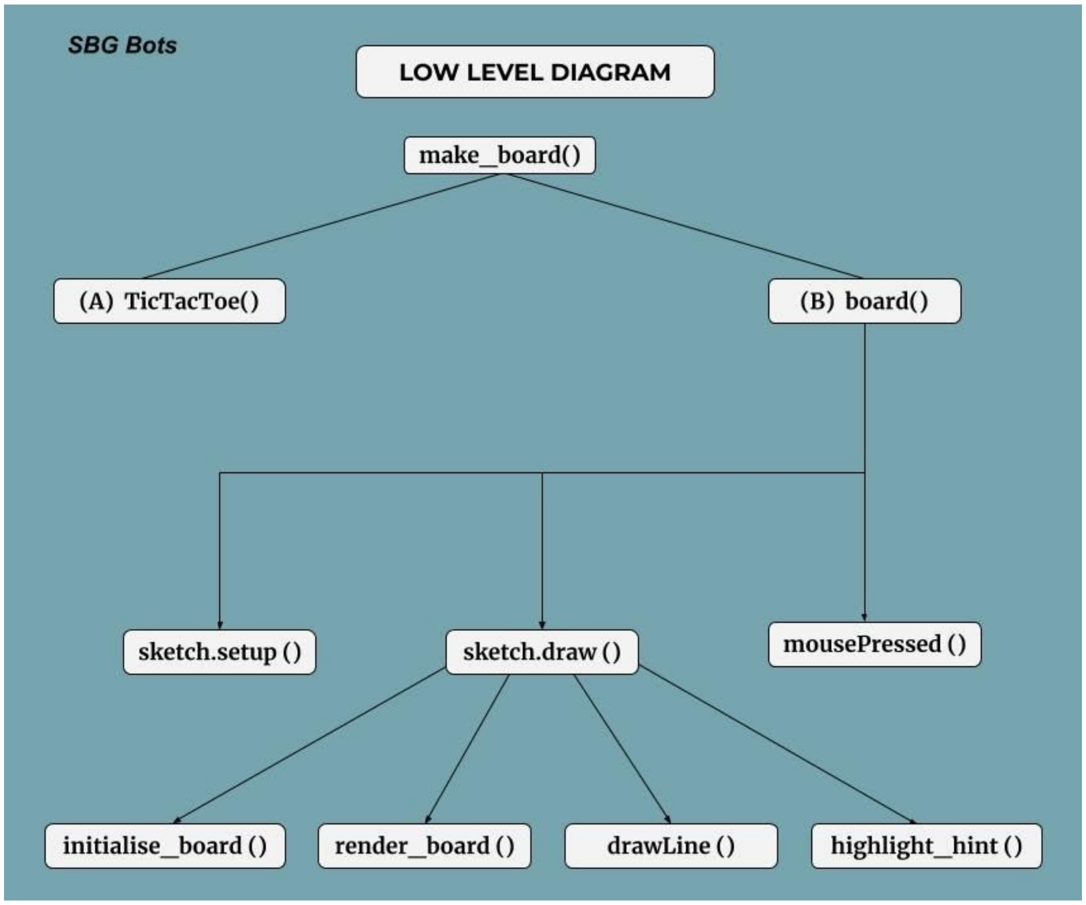
- ***Initialise_board()*** : draws the Tic Tac Toe Game Box
- ***render_board()*** : renders Naughts (O) and Crosses (X) on the game board
- ***find_move()*** : maps the given **percepts** to the ***minimax()*** **Agent Function** which is an **Adversarial Search Algorithm** optimised with **Alpha Beta Pruning** to decide the best move
- ***drawLine()*** : draws the winning line if a player manages to mark a row, column of diagonal
- ***check_end()*** : checks for winners
- ***mark_winner()*** : marks the winner or tie
- ***print_winner()*** : prints the winning player or tie
- ***is_empty()*** : checks if a cell is empty

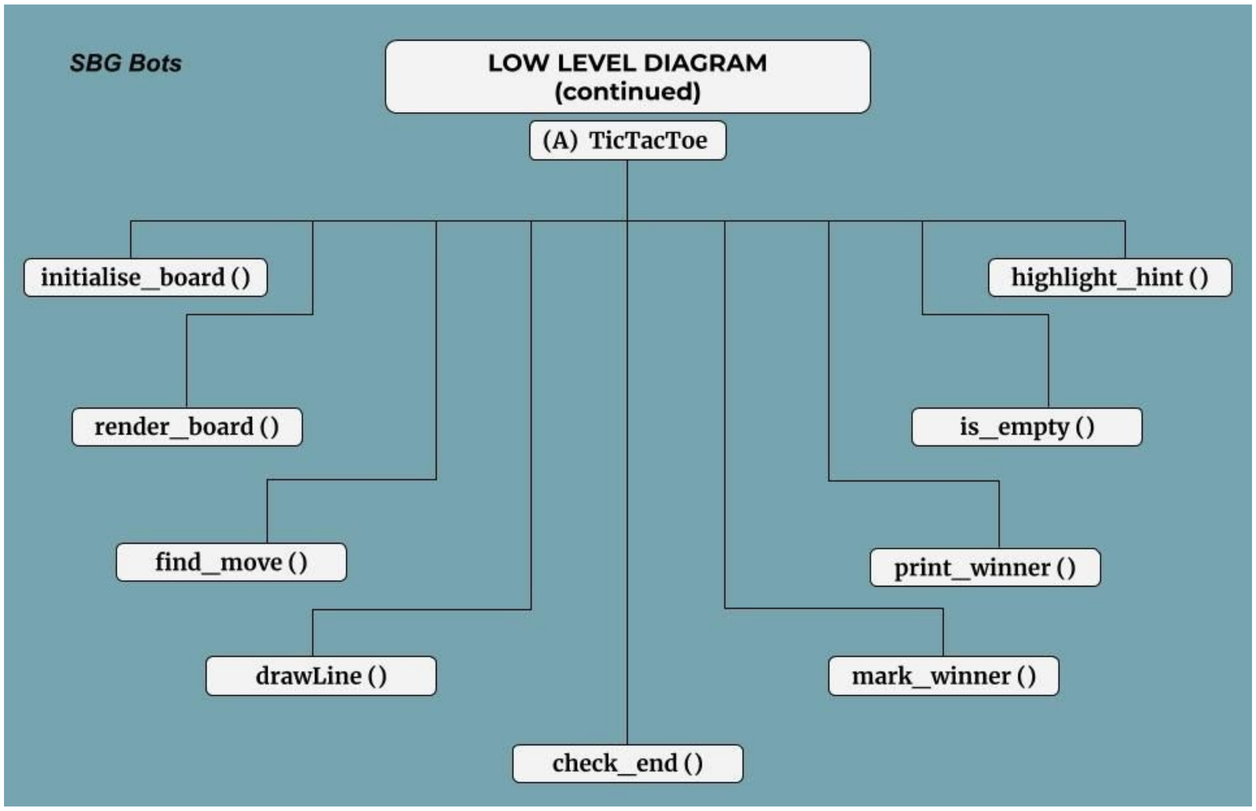
- ***highlight_hint()*** : receives the **percept** that the crew member needs assistance and highlights the next best possible move by mapping the percept to human_move_help() agent function to return the cell address of the next best possible move

board() **Agent Function** maps the **percepts** into action and invokes the following **Agent Functions** that also have their respective **Agent Programs** as listed below:

- ***sketch.setup()*** : It is a **p5.js Agent Program** that runs when the WebApp is started. We use it here to set the environment properties that are background, text, colors, etc.
- ***sketch.draw()*** : It is also a **p5.js Agent Program** that we have used to map percepts to act on :
    - Initialising the board using ***inititialise_board()*** agent function
    - Drawing Naughts and Crosses using ***render_board()*** agent function
    - Highlight hints using ***highlight_hint()*** agent function
    - Drawing the winning stroke using ***drawLine()*** agent function
- ***mousePressed()*** : An **Agent Function** to pass on percepts to render the environment and the game whenever the mouse clicks a valid area on the canvas by sending a **sensory input**.

LOW LEVEL DIAGRAM





HUMAN VS. AI TIC-TAC-TOE

The two players are the crew member and computer. We have the option of choosing the starting player, the difficulty level and hints if the crew member wants it.

- **Starting Player :** The Crew Member can choose the starting player between AI or Human, otherwise AI starts the game itself as a default.
- **Difficulty Level :** The Crew Member can choose the level between easy, medium, difficult, very difficult and impossible. Impossible is the level where the crew either can't win against AI or a tie happens.
- **Hints :** The Crew Member has the option of showing hints after every move.

Minimax Algorithm and Alpha Beta Pruning have been used for implementing Human vs. AI tic-tac-toe.

MINIMAX ALGORITHM

Minimax is an **Adversarial Search Algorithm** which is adopted when there is an "enemy" or "opponent" changing the state of the problem every step in a direction which we do not want.

Examples: Chess, business, trading, war, tic-tac-toe.

We change our game state, but we cannot control the next state. The opponent will change the next state in a way that is unpredictable.

Minimax algorithm chooses the best move for a player assuming that the opponent is also playing optimally. The goal is to maximize the profit or minimize the maximum loss just like the name suggests itself.

Minimax algorithm works in such a way that it goes through all the possible moves of the current game status and returns the best winning condition after each move to make sure that it scores foremost. So for a complicated game the amount of computation increases, therefore minimax must be optimized.

Minimax algorithm is widely used in the successful fields like Artificial Intelligence, statistics, game theory etc.

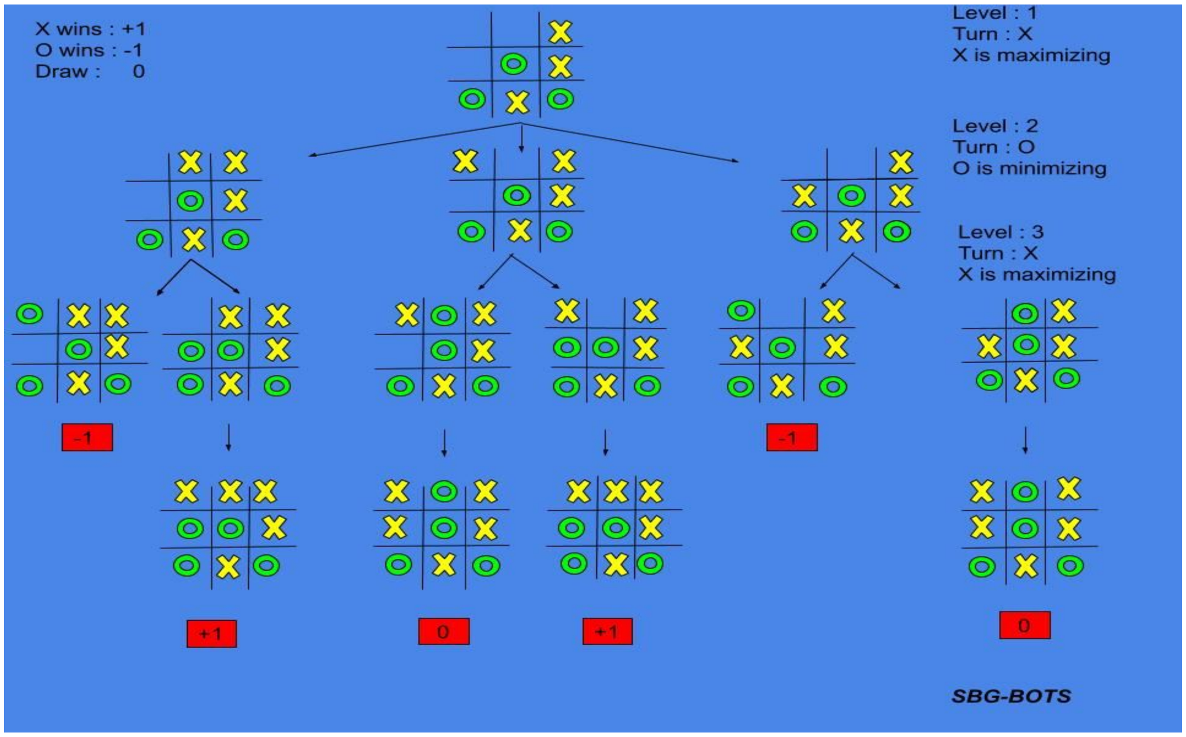


Fig. 1: Game State Tree

Fig. 1 represents the game tree which shows all the possible moves from the current state of the game till the end of the game.

As we can see from level 2 (where turn : O and O is minimizing) to level 3 where O wins the game with a score of -1 as O is minimizing.

Otherwise, from level 3 (where turn : X and X is maximizing) either X wins the game with a score of +1 or draw happens between O and X with a score of 0.

```
function minimax(position, depth, AI)
    if depth ==max_depth or game over in position
        return score evaluation of position

    if AI // maximising player
        maxEval = -Infinity
        for each position in board == empty
            eval = minimax(position, depth +1, false)
            maxEval = max(maxEval, eval)
        return maxEval

    Else
        minEval = +Infinity
        for each position in board == empty
            eval = minimax(position, depth+1, true)
            minEval = min(minEval, eval)
        return minEval

// initial call
minimax(board_position, 3, true)
```

- Note that only some game states are shown for illustration purposes

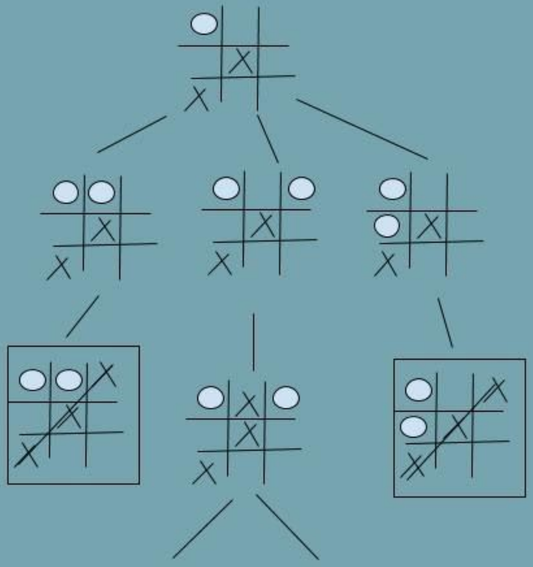


Fig. 2: Minimax Algorithm

In this example, we have shown a board at depth 3 (3 moves played already). Considering the human as O and AI as X, it is now O's turn. O is the minimizing player and should aim to find the best move amongst all possible game states.

It uses the **Depth First Search** by choosing a game state and calling the recursive function minimax until an end state (base case) is reached, which is when:

i) Game is over and a tie

ii) AI or human has made a row, column or diagonal of 3

The score for that player is then calculated. AI will return the maximum score when it wins 3 in a row. It returns that to the parent state that called it, and then another game state possibility is evaluated and the score is returned. AI, the maximising player, then chooses the next game state that returns the maximum score. The next player then plays.

The boxes represent the end states.

ALPHA-BETA PRUNING

Alpha-Beta Pruning is a search algorithm which is used for two-player games. It is the optimization technique for the minimax algorithm. The technique used by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning. This involves two specifications- alpha (the best highest value), Beta (the best lowest value), so called alpha-beta pruning.

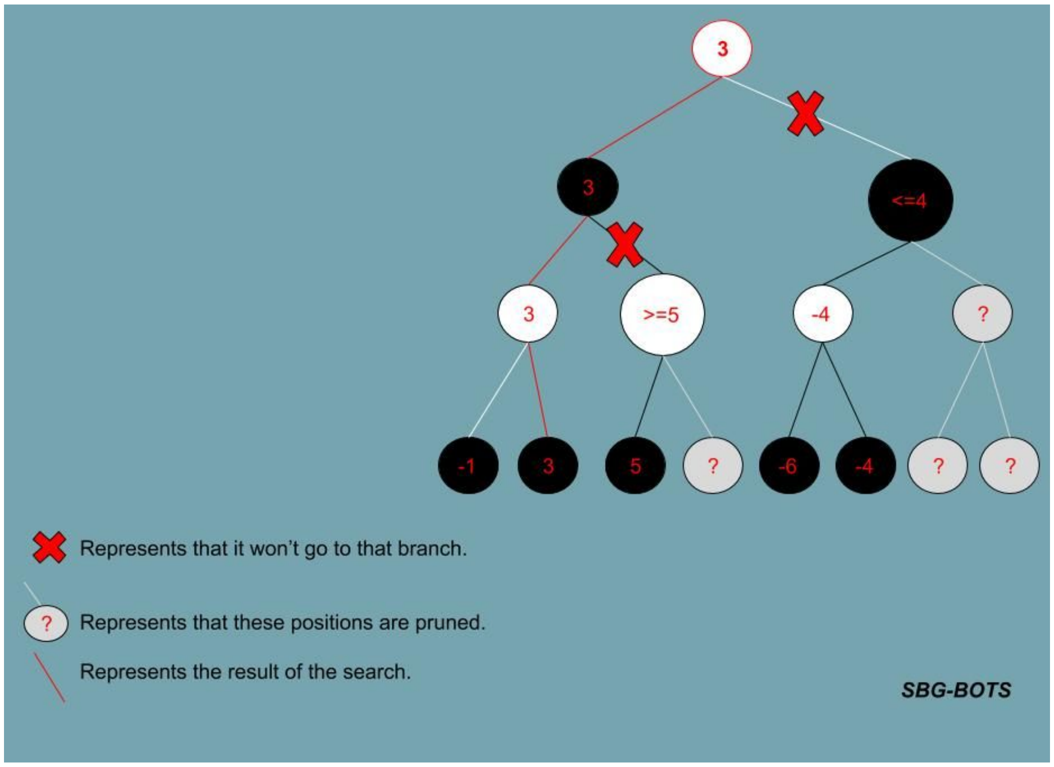
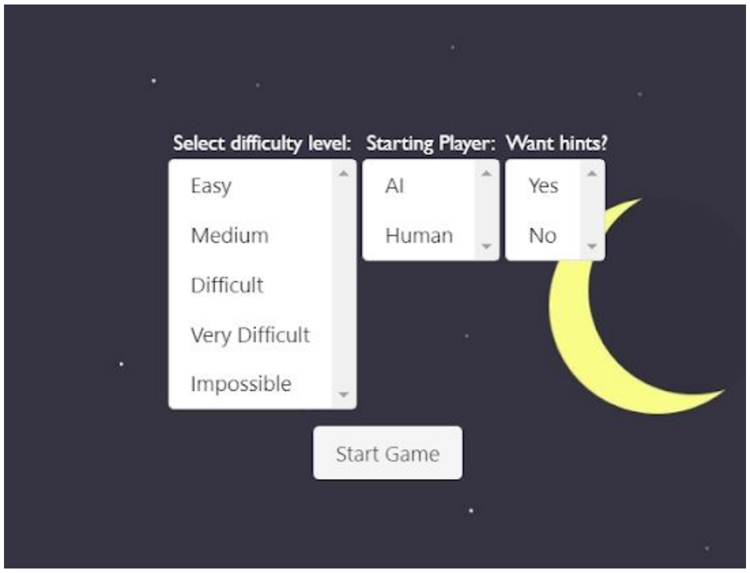

3
3
<=4
3
>=5
-4
?
-1
3
5
?
-6
-4
?
?
Represents that it won't go to that branch.
Represents that these positions are pruned.
Represents the result of the search.
SBG-BOTS

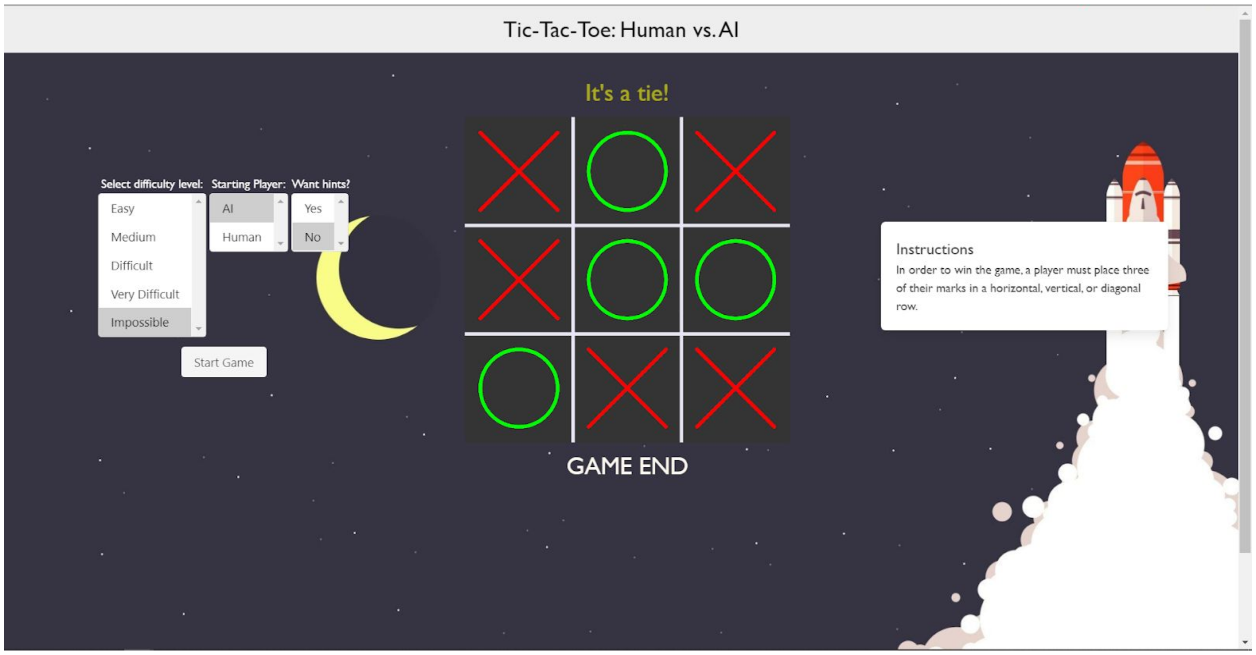
Fig. 3: Alpha-Beta Pruning

Let's run this example to look at how it can be sped up by using pruning. These first few steps are the same as before but consider the situation, we have after evaluating +5 position without evaluating the other position. White can atleast get a 5 from here so we can mark this position as being greater than or equal to 5. We can now see that black won't go down this branch because it already has a better option available. This means that we don't have to waste any computation on evaluating this final position, we can simply pretend that it doesn't exist, in other words we've pruned it from the tree. Things now will continue as normal again for a few steps until we get to -4 (in white). Black to play in this position will be choosing whichever of the two moves leads to the lowest evaluation, so we know that the evaluation here is going to be less than or equal to minus 4. We can now be sure that white won't go down this branch because it already has a better option available to it and so we can prune these positions. As we can see that the results of the search are exactly the same as before but here we've just saved some time by not considering the positions when they can't affect the outcome.



SS1: Human vs. AI Tic Tac Toe Page (features)

Fig. SS1 Is the Human vs. AI Tic Tac Toe Page. Impossible state in the difficulty level is the state where the player either loses or a draw happens. (refer SS2 and SS3)


Tic-Tac-Toe: Human vs. AI

It's a tie!

Select difficulty level: Impossible

Starting Player: AI

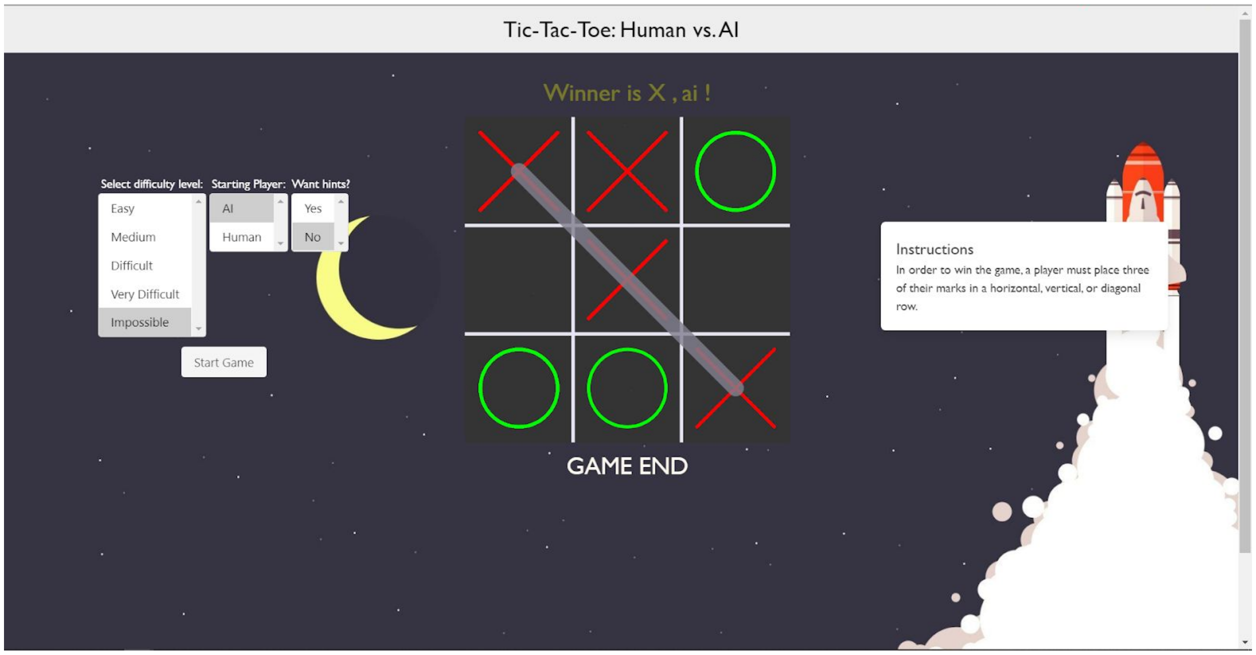
Want hints? No

Instructions

In order to win the game, a player must place three of their marks in a horizontal, vertical, or diagonal row.

GAME END

SS2: Human vs. AI Tic Tac Toe Page (Tie)


Tic-Tac-Toe: Human vs. AI

Winner is X , ai !

Select difficulty level: Impossible

Starting Player: AI

Want hints? No

Instructions

In order to win the game, a player must place three of their marks in a horizontal, vertical, or diagonal row.

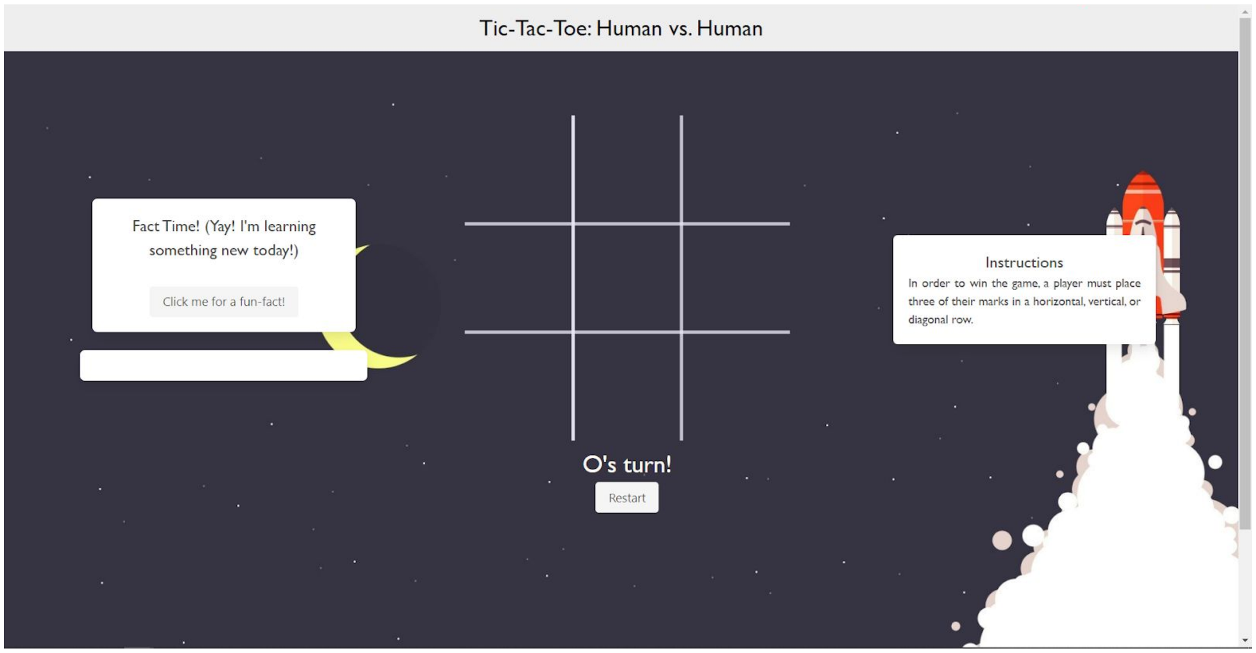
GAME END

SS3: Human vs. AI Tic Tac Toe Page (Winner is AI)

HUMAN VS. HUMAN TIC-TAC-TOE

The two players are two crew members. We have included a Fact Machine for the crew members' entertainment and enlightenment.

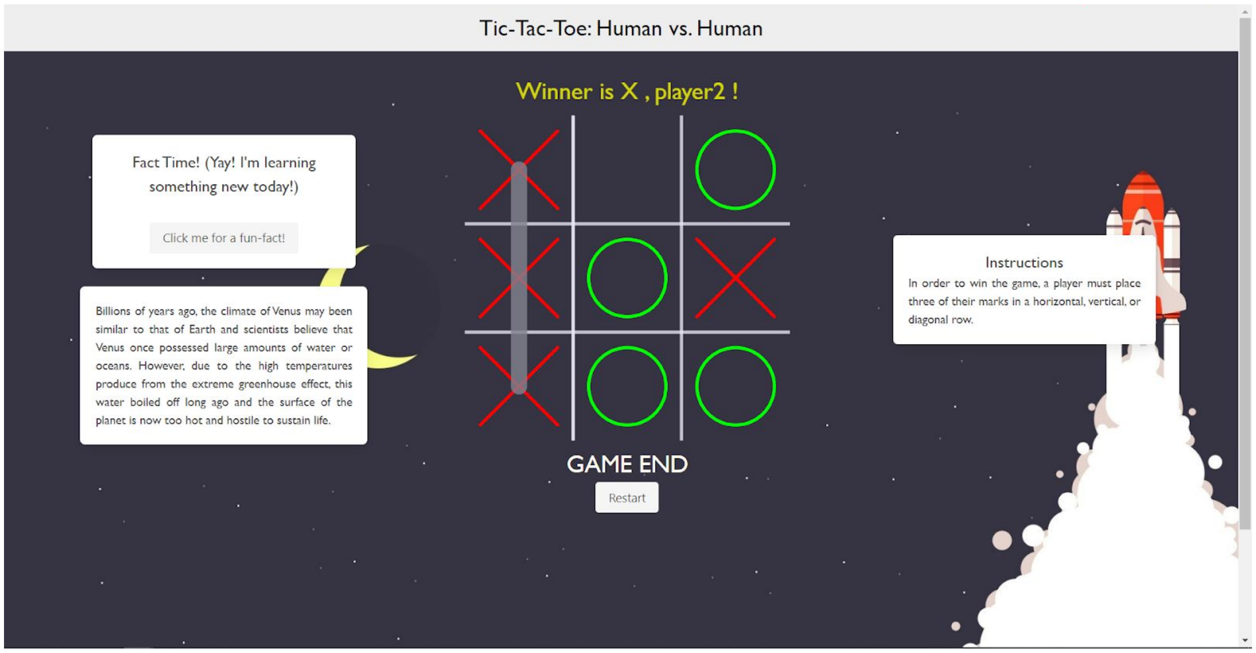
The programming here does not include Minimax and Pruning.


Tic-Tac-Toe: Human vs. Human
Fact Time! (Yay! I'm learning something new today!)
Click me for a fun-fact!
O's turn!
Restart
Instructions

In order to win the game, a player must place three of their marks in a horizontal, vertical, or diagonal row.

SS4: Human vs. Human Tic Tac Toe Page

The game is between player 1 and player 2. As the game ends the player can restart the game by clicking the restart button. We have also included a Fact Machine for the entertainment and enlightenment of the crew. For an example, refer to fig. SS5.


Tic-Tac-Toe: Human vs. Human
Winner is X , player2 !
Fact Time! (Yay! I'm learning something new today!)
Click me for a fun-fact!
GAME END
Restart
Instructions

Billions of years ago, the climate of Venus may been similar to that of Earth and scientists believe that Venus once possessed large amounts of water or oceans. However, due to the high temperatures produce from the extreme greenhouse effect, this water boiled off long ago and the surface of the planet is now too hot and hostile to sustain life.

In order to win the game, a player must place three of their marks in a horizontal, vertical, or diagonal row.

SS5: Human vs. Human Tic Tac Toe Page (facts)

HUMAN VS. AI MISERE TIC-TAC-TOE

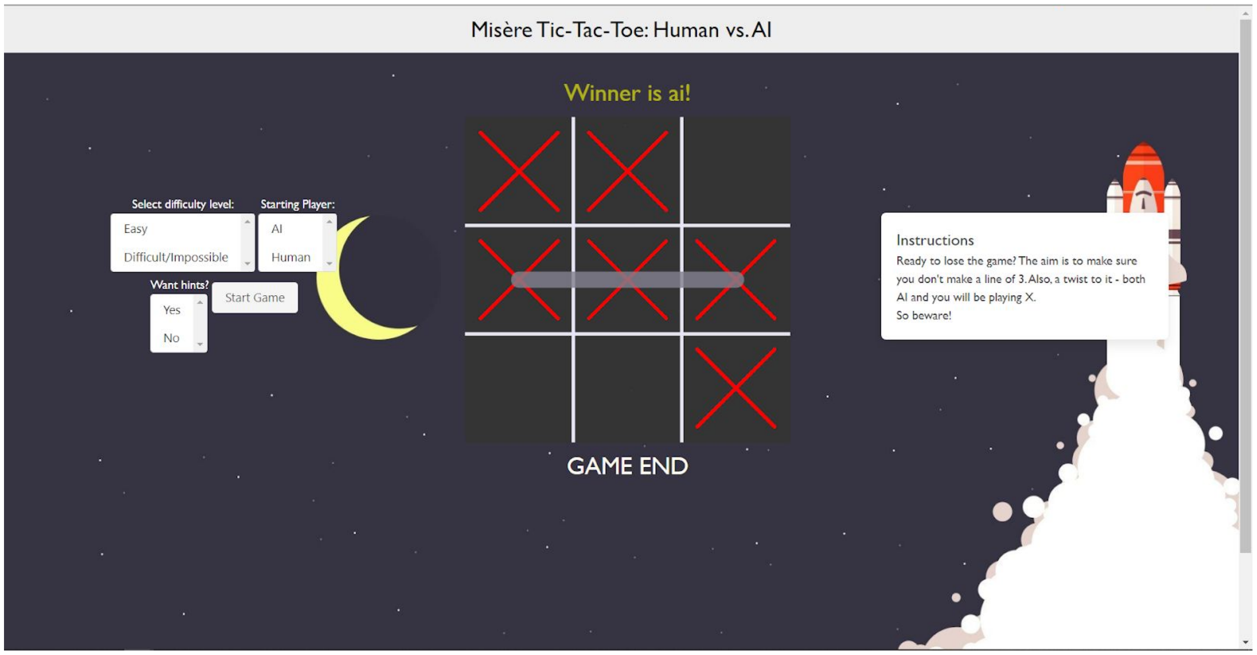
The two players are the crew member and computer, similar to Human vs. AI Tic Tac Toe. We have the same options of choosing the starting player, the difficulty level and hints if the crew member wants it.

- **Starting Player :** The Crew Member can choose the starting player between AI or Human, otherwise AI starts the game itself as a default.
- **Difficulty Level :** The Crew Member can choose the level between easy and difficult/impossible.
- **Hints :** The Crew Member has the option of seeing hints after every move.

This game is the opposite of a normal tic tac toe. The aim is to not make 3 in a row, column or diagonal, to win the game. This has been implemented by:

- Minimizing the score returned when a line of 3 is made by AI
- This penalizes the AI in the minimax function
- AI will aim to maximise score

Our variation players a disjunctive version of the game, where both players play ‘X’ symbol. This allows any one of them to complete the game, as both players have the symbol X.



Instructions
Ready to lose the game? The aim is to make sure you don't make a line of 3. Also, a twist to it - both AI and you will be playing X.
So beware!

SS6: Human vs. AI Misere Tic Tac Toe Page