Overview

We have created a browser based web application that involves the minimax algorithm to create an unbeatable AI powered Tic-Tac-Toe game. The game is built using HTML and CSS for the User Interface designing and JavaScript to code our algorithm on which our AI player works. Some of the salient features of our version of the Tic-Tac-Toe are:-

- The game provides an option to choose which players we want to play with ie. do we want to play against another human friend of ours or the unbeatable AI (computer).

- We have delayed animation between two moves of the players to give a real hand drawing effect and the board UI has been designed to give the real board effect (the details and the libraries used are mentioned afterwards).
- The human players have an option to choose what sign they would play with, zeros or crosses.
- There is a 'Reset' option if we wish to abandon the game in the middle and start again or if we wish to start a new game after we have ended the game.
- If any of the players wins or makes a winning pattern then that would be displayed using an animation that duplicates the real drawing of a line to mark the winning pattern.
- In the end the game displays the player who wins or whether or not it is a tie.
- We have also provided the option on the top right corner to directly view our Github code for the game.

In the human vs human match, we just input the moves on the board and then check if it creates a winning pattern/ terminal state of the board which marks whether it is a tie and none of the players has won or if any one has won. We display the sign of the player at the position it has clicked and then change the turn by changing the boolean.

The soul of the game is the logic that is used to create our AI player and we use the minimax algorithm to choose the next optimal move of the computer player.

MINIMAX ALGORITHM
(IN GENERAL)

Minimax is a kind of backtracking algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally. It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

In Minimax the two players are called maximizer and minimizer. The maximizer tries to get the highest score possible while the minimizer tries to do the opposite and get the lowest score possible.

Every board state has a value associated with it. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value.

MINIMAX ALGORITHM IN TIC TAC TOE GAME THEORY

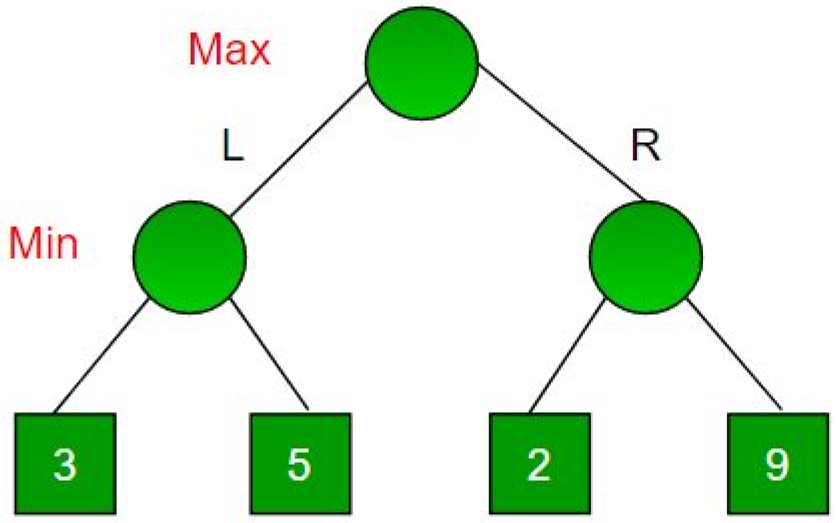
(AI VS HUMAN)

FINDING THE BEST MOVE

We shall be introducing a new function which evaluates all the available moves and then returns the best move the maximizer can make.

To check whether or not the current move is better than the best move we take the help of a function which will consider all the possible ways the game can go and returns the best value for that move, assuming the opponent also plays optimally

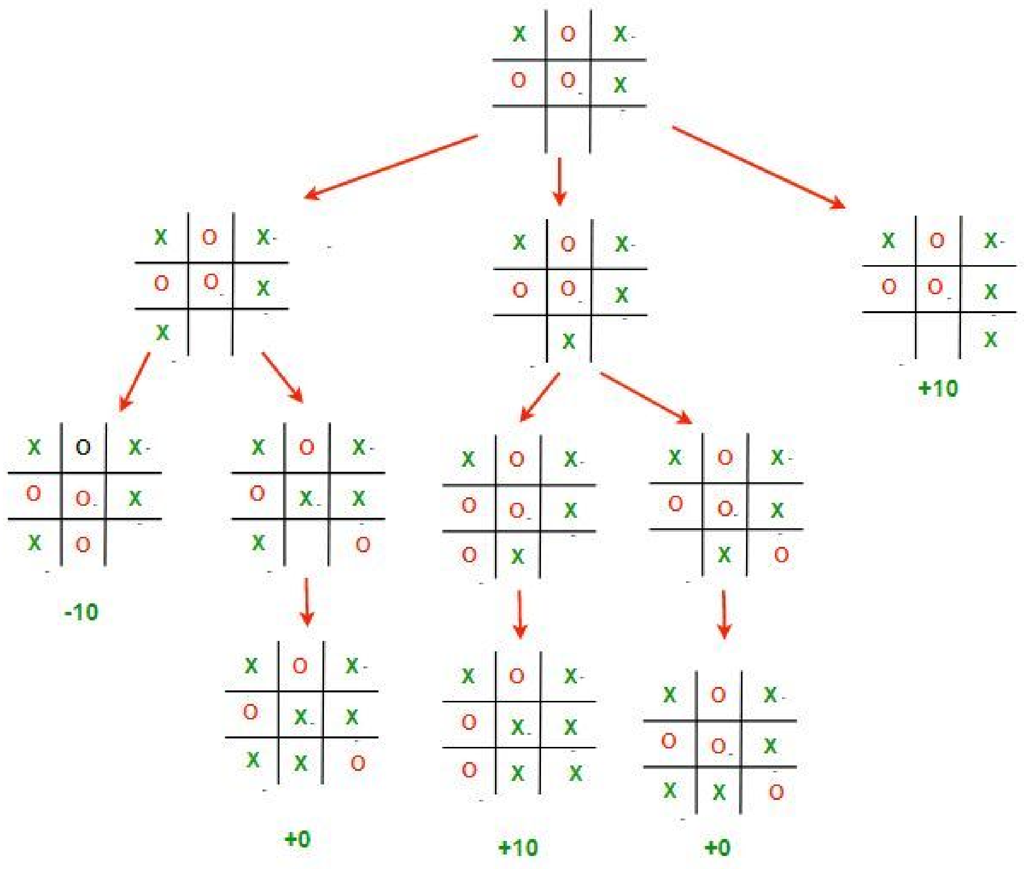
AN EXAMPLE:-
Lets consider a game which has 4 final states and paths to reach final state are from root to 4 leaves of a perfect binary tree as shown below. Assume you are the maximizing player and you get the first chance to move, i.e., you are at the root and your opponent at next level. **Which move you would make as a maximizing player considering that your opponent also plays optimally?**



Since this is a backtracking based algorithm, it tries all possible moves, then backtracks and makes a decision.

- Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will definitely choose the least among both, that is 3
- Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

GAME TREE EXPLANATION



This image depicts all the possible paths that the game can take from the root board state. It is often called the Game Tree.

The 3 possible scenarios in the above example are :

- Left Move : If X plays [2,0]. Then O will play [2,1] and win the game. The value of this move is -10
- Middle Move : If X plays [2,1]. Then O will play [2,2] which draws the game. The value of this move is 0

- Right Move : If X plays [2,2]. Then he will win the game. The value of this move is +10;

Even though X has a possibility of winning if he plays the middle move, O will never let that happen and will choose to draw instead.

Assume that there are 2 possible ways for X to win the game from a give board state.

- Move **A** : X can win in 2 move
- Move **B** : X can win in 4 moves

Our evaluation function will return a value of +10 for both moves **A** and **B**. Even though the move **A** is better because it ensures a faster victory, our AI may choose **B** sometimes. To overcome this problem we subtract the depth value from the evaluated score. This means that in case of a victory it will choose a the victory which takes least number of moves and in case of a loss it will try to prolong the game and play as many moves as possible. So the new evaluated value will be

- Move **A** will have a value of +10 – 2 = 8
- Move **B** will have a value of +10 – 4 = 6

Now since move **A** has a higher score compared to move **B** our AI will choose move **A** over move **B**. The same thing must be applied to the minimizer. Instead of subtracting the depth we add the depth value as the minimizer always tries to get, as negative a value as possible. We can subtract the depth either inside the evaluation function or outside it. Anywhere is fine. I have chosen to do it outside the function.

Use of Javascript

JavaScript can be executed in time-intervals.That we used to give a time gap between two moves to create real effect of hand drawing.

`The setTimeout(`*`function, milliseconds`*) - executes a function, after waiting a specified number of milliseconds.

The first parameter is the function to be executed.

The second parameter indicates the number of milliseconds before execution.

We have also embedded Scalable Vector Graphics elements to add the background patterns and real drawing effects in the board.

The rest of the features and the code has been explained using comments in the code.