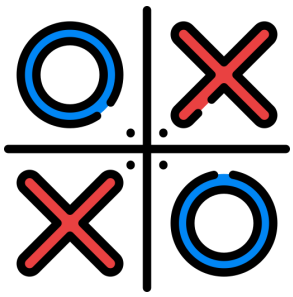


THE MARS COLONIZATION MISSION

TIC TAC TOE

-BY TEAM MANGAL- YATRI

OUR TEAM

NIKHIL NISCHAL 2018EEB1169 IIT ROPAR	PREETESH VERMA 2018EEB1171 IIT ROPAR	GARIMA SONI 2018CSB1089 IIT ROPAR

As a part of the Microsoft Engage 2020 program we were asked to prepare an **AI Powered Tic-Tac-Toe Game** using the Minimax Algorithm and to present it with a web interface.The Algorithm has been written in JavaScript and the web interface has been developed using HTMl5,CSS and Javascript.Mongo database has been used as Database service to store the scores of various users who play the game.The Game has two parts: Human vs Ai and Human Vs Human. The first part has several difficulty levels which range from being completely random to being unbeatable. We have also provided the game with a HINT Feature where the user when unable to decide upon a move could refer to for help. In the following follow up we explain the minimax algorithm which powers the game along with the rules of the game and the scoring pattern.

TIC TAC TOE

It's a very popular game. We might have played at some point of our childhood but still we would like to explain the rules of the game -

1. The game is to be played by two players (We have both functionalities for Human vs Human and Human vs Computer).
2. One of the players chooses 'O' and the other 'X' to mark their respective cells.
3. One player starts the game and the game ends when one of the players has one complete row/ column/ diagonal filled with their respective symbol.
4. If no one wins, then the game ends in a tie.

IMPORTANT RESULTS

- The game is designed in such a way that if the player optimally plays with the computer at the toughest level then the game will always be atmost a draw.
- At the very easy, easy, etc levels other than the toughest level, the chances of winning against the computer decreases as the levels are increased.

BACKTRACKING

It is an algorithm that explores the space of all possible answers to help in finding the solution of a computational problem which satisfies some constraints by building possible cases with respect to the solutions and abandones each partial case which possibly can not be completed to validate the solution.

It develops answers by identifying a set of successive decisions.

It will not necessarily give the optimal solution.

MINIMAX ALGORITHM

It is a kind of backtracking algorithm that is used in decision making and game theory to find the best move of a player, assuming that the opponent plays optimally.

It is widely used in two player games like Tic Tac Toe, Mancala, Chess, etc.

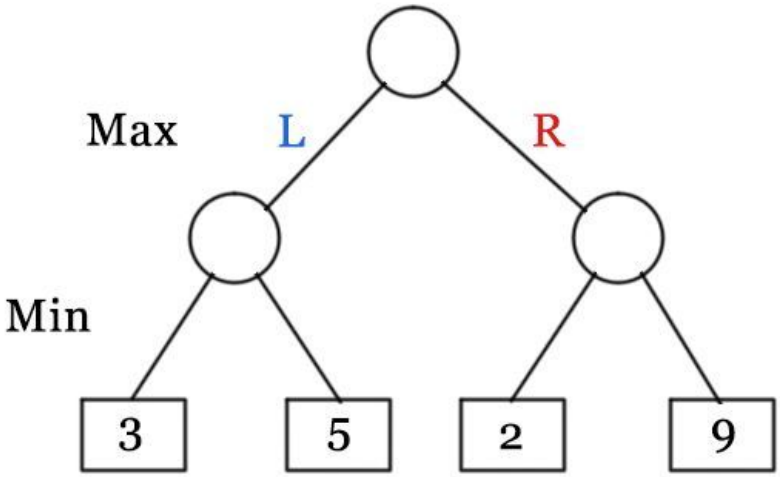
This algorithm majorly has 2 parts-

Maximizer and Minimizer

In the Minimax algorithm we set two players as maximizer and minimizer.The **maximizer** tries to get the highest score possible at each step of the game 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.

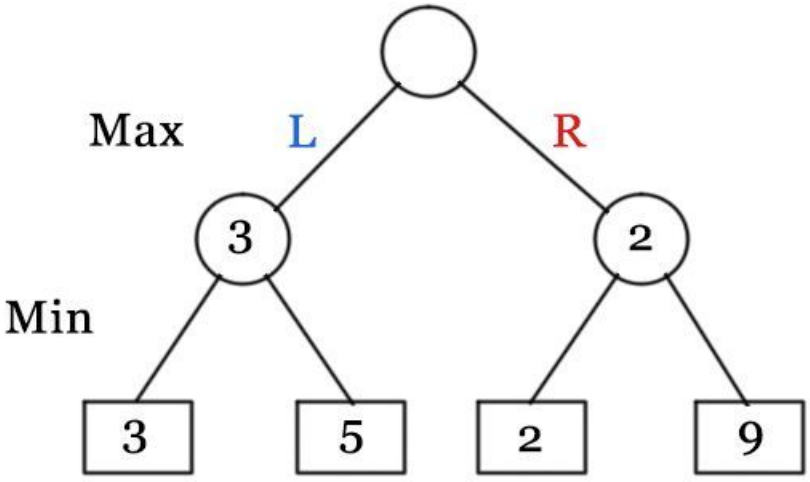
If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.



In the above example

● 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.

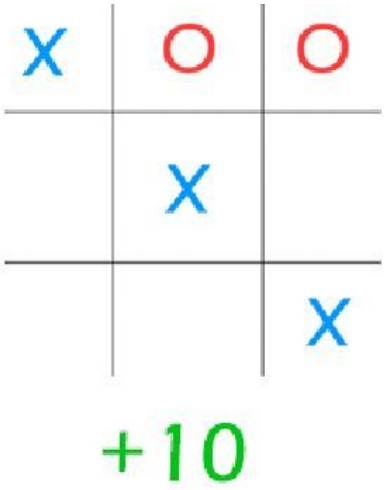


In order to keep the count of maximizer and minimizer we need to implement a function that calculates the value of the board depending on the placement of pieces on the board. This function is often known as Evaluation Function. It is sometimes also called **Heuristic Function.**

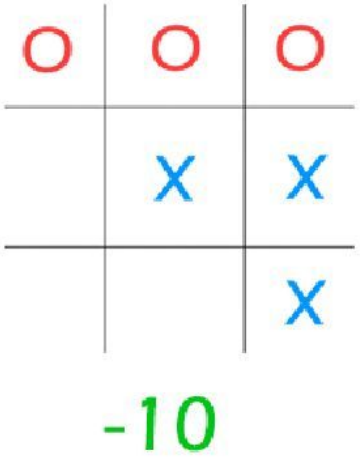
The evaluation function is unique for every type of game. The basic idea behind the evaluation function is to give a high value for a board if maximiser's turn or a low value for the board if minimizer's turn.

For this scenario let us consider X as the maximizer and O as the minimizer.This is what our evaluation function looks like :

1) If X wins on the board we give it a positive value of +10 as per shown in the image.

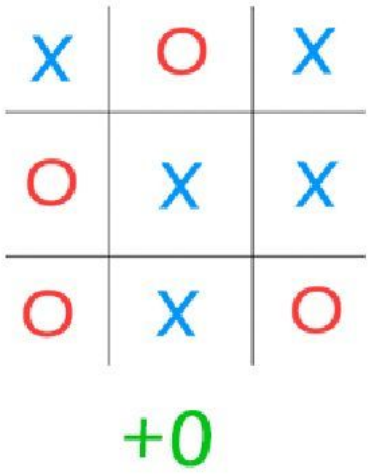


2)If O wins on the board we give it a negative value of-10 as per shown in the image.



3)If no one has won or the game results in a draw then

we give a value of +0 as per shown in the image.



We could have chosen any positive / negative value other than 10. For the sake of simplicity we chose 10.

● Finding optimal move / best move With the help of min max we can assure that the computer will never lose but we want the system to play optimally means to win in minimum number of steps when it can.

Let's assume that there are 2 possible ways for X to win the game from a given 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 gives a faster victory, although the system may choose B sometimes by min max algorithm.

To overcome this problem we subtract the depth value from the calculated score. It means that in case of a Win it will choose a the Win 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 system will choose move A over move B. The same thing is applied to the minimizer. Rather than 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 ,it will not create much of a difference.

We can write the pseudocode simply as

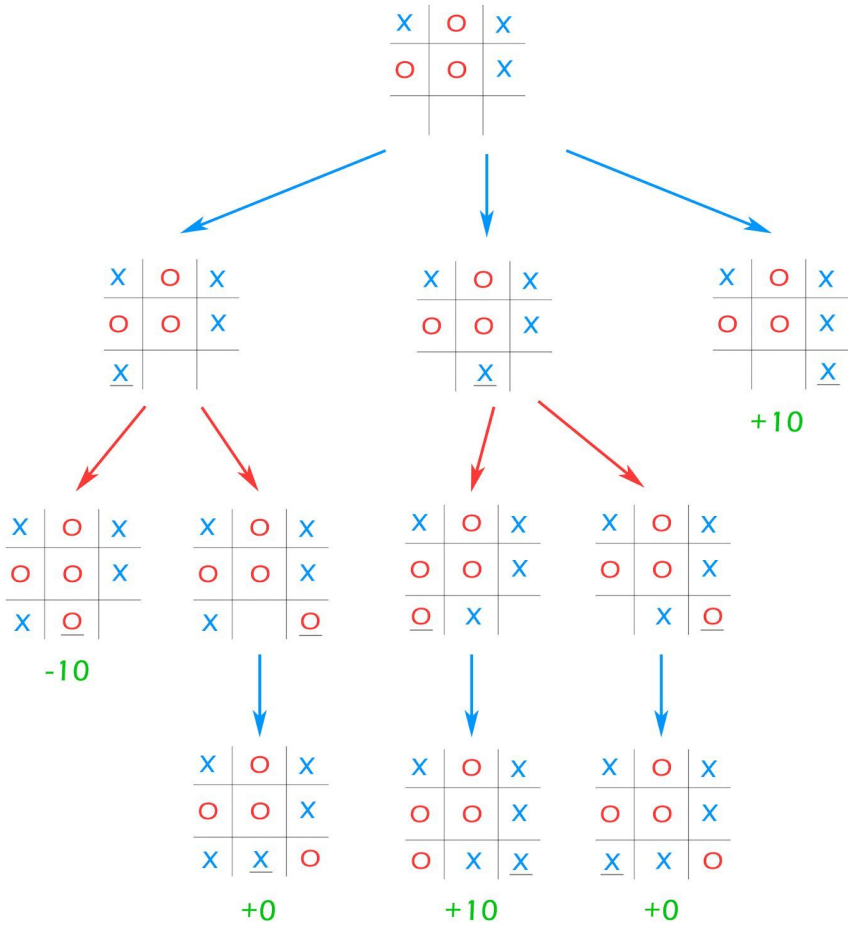
if minimizer has won;

return WIN_SCORE - depth

if maximizer has won;

return LOSE_SCORE + depth

In the following section we try to demonstrate working of



The 3 possible cases in the above example are :

{here (a,b) represent simple matrix representation}

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

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

3. 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.As we have consider that O will play optimally because if it didn't then we don't have worry about that case as it will be automatically be taken care of.

Therefore the best choice for X, is to play [2,2], which will guarantee a victory for him.

In this way we can play tic-tac-toe optimally to compete and make the game toughest.

A FLOW CHART DESCRIBING THE WALK THROUGH OF THE APP