IMPLEMENTING TIC-TAC-TOE AI WITH MINIMAX ALGORITHM WITH ALPHA- BETA PRUNING



Presented by-

Abhey Sehrawat

002EC15

**INTRODUCTION-**

**Tic-tac-toe** (also known as **noughts and crosses** or **Xs and Os**) is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

Because of the simplicity of tic-tac-toe, it is often used in the branch of artificial intelligence that deals with the searching of game trees. It is straightforward to write a computer program to play tic-tac-toe perfectly or to enumerate the 7665 essentially different positions (the state space complexity) or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space.

**ALGORITHM DETAILS-**

A **minimax** algorithm is a recursive algorithm for choosing the next move in an n-player game, usually a two-player, back and forth game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of his legal moves.

A simple version of the minimax algorithm, stated below, deals with games such as tic-tac-toe, where each player can win, lose, or draw. If player A can win in one move, his best move is that winning move. If player B knows that one move will lead to the situation where player A can win in one move, while another move will lead to the situation where player A can, at best, draw, then player B's best move is the one leading to a draw. Late in the game, it's easy to see what the "best" move is. The Minimax algorithm helps find the best move, by working backwards from the end of the game. At each step it assumes that player A is trying to maximize the chances of A winning, while on the next turn player B is trying to minimize the chances of A winning (i.e., to maximize B's own chances of winning).

**Alpha–beta** pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. This allows us to search much faster and even go into deeper levels in the game tree. It cuts off branches in the game tree which need not be searched because there already exists a better move available. The algorithm maintains two values, alpha and beta, which represent the maximum score that the maximizing player is assured of and the minimum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, i.e. both players start with their lowest possible score. It can happen that when choosing a certain branch of a certain node the minimum score that the minimizing player is assured of becomes less than the maximum score that the maximizing player is assured of (beta <= alpha). If this is the case, the parent node should not choose this node, because it will make the score for the parent node worse. Therefore, the other branches of the node do not have to be explored.

**PSEUDOCODE-**

function minimax(node, depth, isMaximizingPlayer, alpha, beta):

if node is a leaf node :

return value of the node

if isMaximizingPlayer :

bestVal = -INFINITY

for each child node :

value = minimax(node, depth+1, false, alpha, beta)

bestVal = max( bestVal, value)

alpha = max( alpha, bestVal)

if beta <= alpha:

break

return bestVal

else :

bestVal = +INFINITY

for each child node :

value = minimax(node, depth+1, true, alpha, beta)

bestVal = min( bestVal, value)

beta = min( beta, bestVal)

if beta <= alpha:

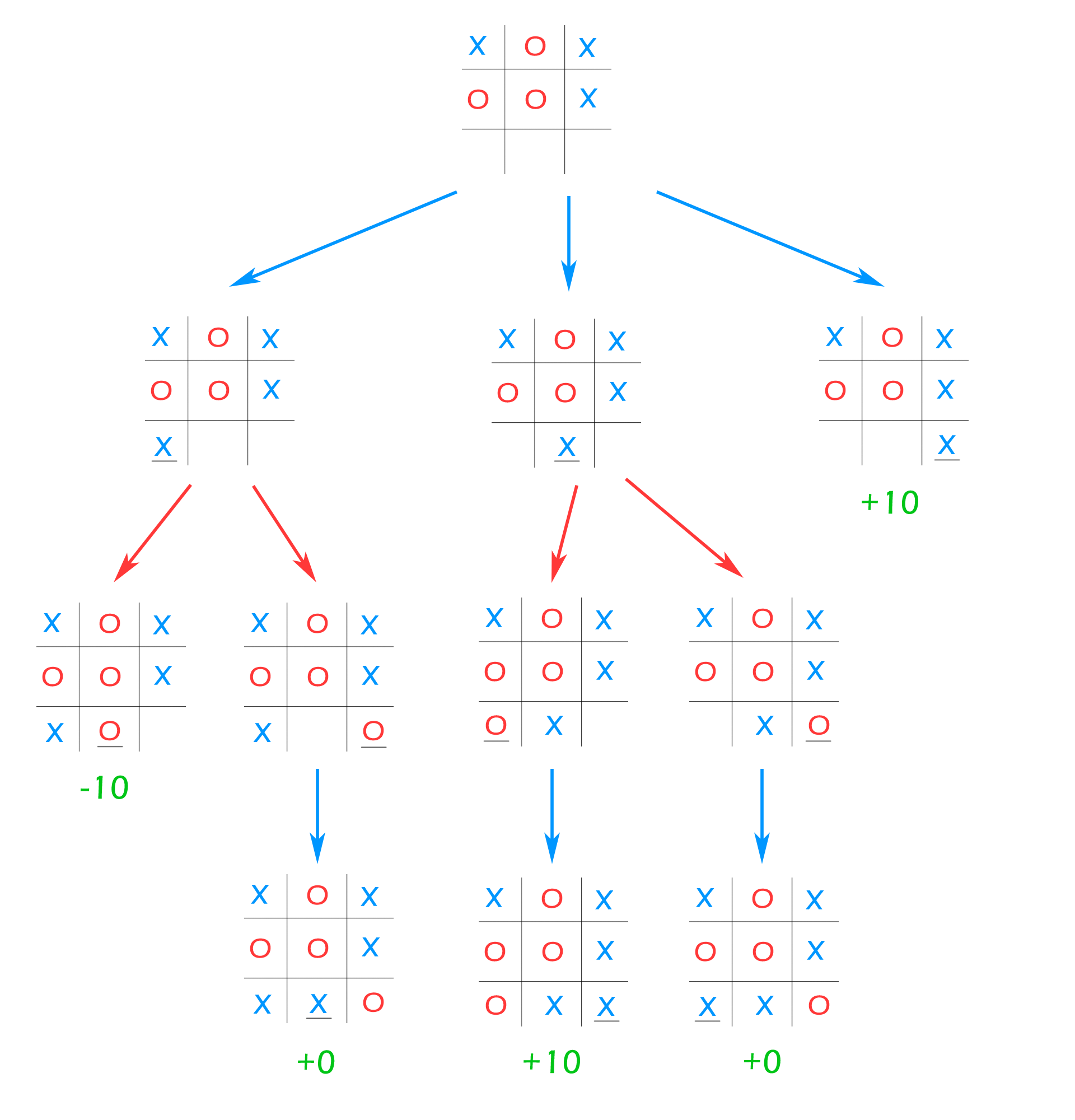
break

return bestVal

// Calling the function for the first time.

minimax(0, 0, true, -INFINITY, +INFINITY)

MINIMAX ALGORITHM VISUALISATION-



CODE-

IMPLEMENTATION OF MINIMAX-

std::pair<int, std::pair<int, int>> minimax\_optimization(char board[3][3], char marker, int depth, int alpha, int beta)

{

// Initialize best move

std::pair<int, int> best\_move = std::make\_pair(-1, -1);

int best\_score = (marker == AI\_MARKER) ? LOSS : WIN;

// If we hit a terminal state (leaf node), return the best score and move

if (board\_is\_full(board) || DRAW != get\_board\_state(board, AI\_MARKER))

{

best\_score = get\_board\_state(board, AI\_MARKER);

return std::make\_pair(best\_score, best\_move);

}

std::vector<std::pair<int, int>> legal\_moves = get\_legal\_moves(board);

for (int i = 0; i < legal\_moves.size(); i++)

{

std::pair<int, int> curr\_move = legal\_moves[i];

board[curr\_move.first][curr\_move.second] = marker;

// Maximizing player's turn

if (marker == AI\_MARKER)

{

int score = minimax\_optimization(board, PLAYER\_MARKER, depth + 1, alpha, beta).first;

// Get the best scoring move

if (best\_score < score)

{

best\_score = score - depth \* 10;

best\_move = curr\_move;

// Check if this branch's best move is worse than the best

// option of a previously search branch. If it is, skip it

alpha = std::max(alpha, best\_score);

board[curr\_move.first][curr\_move.second] = EMPTY\_SPACE;

if (beta <= alpha)

{

break;

}

}

} // Minimizing opponent's turn

else

{

int score = minimax\_optimization(board, AI\_MARKER, depth + 1, alpha, beta).first;

if (best\_score > score)

{

best\_score = score + depth \* 10;

best\_move = curr\_move;

// Check if this branch's best move is worse than the best

// option of a previously search branch. If it is, skip it

beta = std::min(beta, best\_score);

board[curr\_move.first][curr\_move.second] = EMPTY\_SPACE;

if (beta <= alpha)

{

break;

}

}

}

board[curr\_move.first][curr\_move.second] = EMPTY\_SPACE; // Undo move

}

return std::make\_pair(best\_score, best\_move);

}

**REFERENCES-**

1. [**https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/**](https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/)
2. [**https://en.wikipedia.org/wiki/Minimax**](https://en.wikipedia.org/wiki/Minimax)