



TIC TAC TOE



Prepared By,
Arnav Barman &
Tejasvi Jain





Table of Content

01

Project Details

02

Introduction



03

What and how of
Minimax?

04

Understanding the
Code!

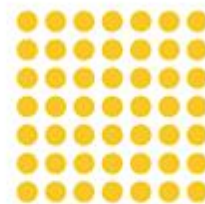
06

Game Tree

07

The complete
project





Project Details

Our project is an implementation of the Minimax AI Algorithm on the Tic-Tac-Toe (or Naughts and Crosses) game using the Minimax algorithm in AI.



Team

Arnav Barman

102053038

Tejasvi Jain

102003367


Introduction

To solve games using AI, we will introduce the concept of a game tree followed by the minimax algorithm. The different states of the game are represented by nodes in the game tree, very similar to the above planning problems. The idea is just slightly different. In the game tree, the nodes are arranged in levels that correspond to each player's turns in the game so that the "root" node of the tree (usually depicted at the top of the diagram) is the beginning position in the game. In tic-tac-toe, this would be the empty grid with no Xs or Os played yet. Under root, on the second level, there are the possible states that can result from the first player's moves, be it X or O. We call these nodes the "children" of the root node.

Each node on the second level would further have as its children nodes the states that can be reached from it by the opposing player's moves. This is continued, level by level, until reaching states where the game is over. In tic-tac-toe, this means that either one of the players gets a line of three and wins, or the board is full and the game ends in a tie.




What and how of Minimax?



Minimax Algorithm is a decision rule formulated for 2 player zero-sum games (Tic-Tac-Toe, Chess, Go, etc.). This algorithm sees a few steps ahead and puts itself in the shoes of its opponent. It keeps playing and exploring subsequent possible states until it reaches a terminal state resulting in a draw, a win, or a loss. Being in any of these possible terminal states has some utility for the AI — such as being in a 'Win' state is good (utility is positive), being in a 'Loss' state is bad (utility is negative), and being in a draw in neither good nor bad (utility is neutral). These games are known as zero-sum games, because in a mathematical representation: one player wins (+1) and other player loses (-1) or both of them draw (0).

The algorithm search, recursively, the best move that leads the Max player to win or not lose (draw). It considers the current state of the game and the available moves at that state, then for each valid move it plays (alternating min and max) until it finds a terminal state (win, draw or lose).



Understanding the Code!



The MAX(+1) may be X or O and the MIN(-1) may be O or X, whatever. The board is 3x3.

```
def minimax(state, depth, player):
```

state: the current board in tic-tac-toe (node)

depth: index of the node in the game tree

player: may be a MAX player or MIN player

```
    if player == MAX:  
        return [-1, -1, -infinity]  
    else:  
        return [-1, -1, +infinity]
```



Both players start with their worst score. If the player is MAX, its score is -infinity. Else if the player is MIN, its score is +infinity. The best move on the board is [-1, -1] (row and column) for all.

```
    if depth == 0 or game_over(state):  
        score = evaluate(state)  
        return score
```


If the depth is equal zero, then the board hasn't new empty cells to play. Or, if a player wins, then the game ended for MAX or MIN. So the score for that state will be returned.

If MAX won: return +1


If MIN won: return -1

Else: return 0 (draw)





```
for cell in empty_cells(state):
    x, y = cell[0], cell[1]
    state[x][y] = player
    score = minimax(state, depth - 1, -player)
    state[x][y] = 0
    score[0], score[1] = x, y
```




For each valid moves (empty cells):

- **x**: receives cell row index
- **y**: receives cell column index
- **state[x][y]**: it's like board[available_row][available_col] receives MAX or MIN player
- **score = minimax(state, depth - 1, -player)**:
 - state: is the current board in recursion;
 - depth - 1: index of the next state;
 - -player: if a player is MAX (+1) will be MIN (-1) and vice versa.

The move (+1 or -1) on the board is undo and the row, column are collected.

The next step is compare the score with best.

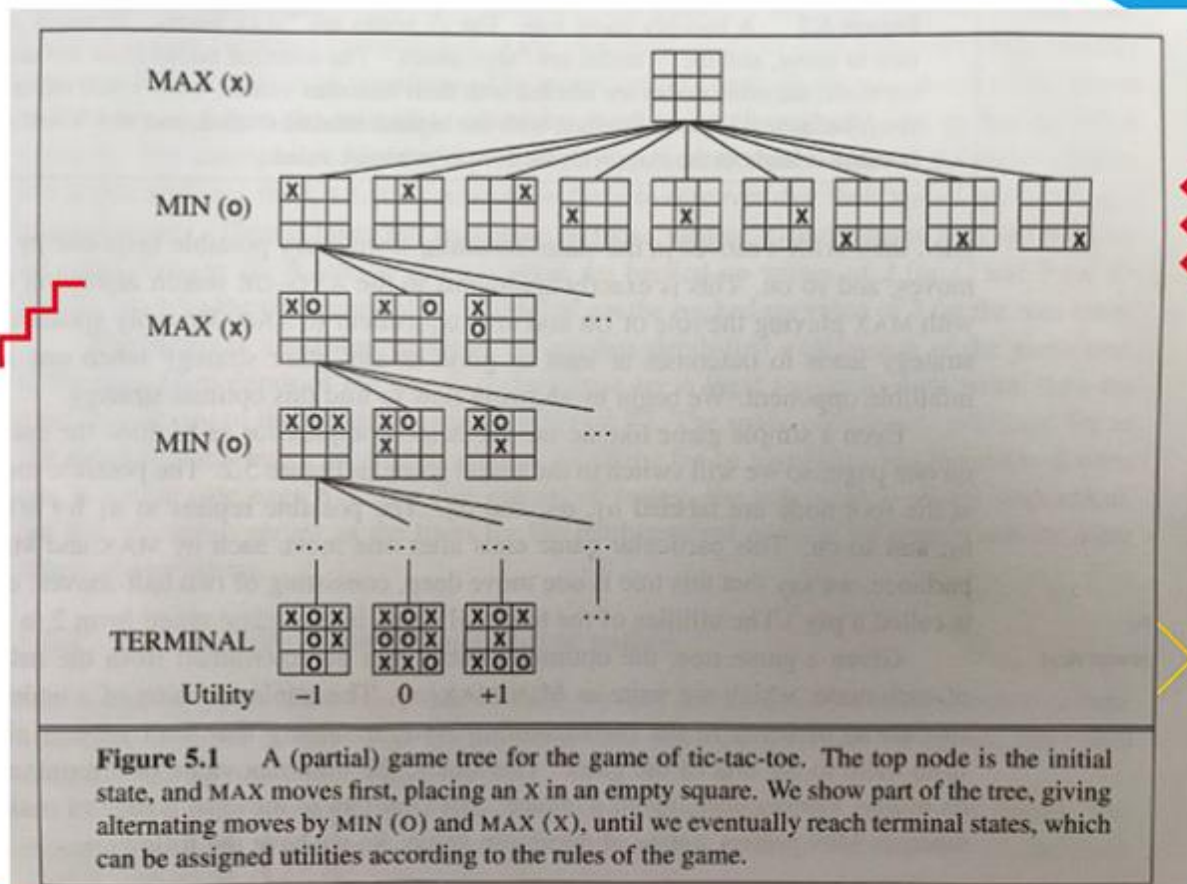


```
if player == MAX:
    if score[2] > best[2]:
        best = score
else:
    if score[2] < best[2]:
        best = score
```

For MAX player, a bigger score will be received. For a MIN player, a lower score will be received. And in the end, the best move is returned.



Game Tree



The full game tree has 5,49,946 nodes!



The Complete Project

https://github.com/Arnav-Barman/TicTacToe-AI_Project-Minimax/blob/main/TicTacToe.ipynb

