**TIC-TAC-TOE**

**GAME**



#### Submitted by

**PRAKASH Y (RA2011003010044)**

**HARISH S (RA2011003010046)**

**ABIISHEK G (RA2011003010063)**

*Under the guidance of*

**G MANOJ KUMAR**

(Guide Affiliation)

#### in partial fulfillment for the award of the degree of

### BACHELOR OF TECHNOLOGY

in

**COMPUTER SCIENCE & ENGINEERING**

of

### FACULTY OF ENGINEERING AND TECHNOLOGY



##### 

##### S.R.M.Nagar, Kattankulathur, Chengalpattu District

### APRIL 2023

# ABSTRACT

This project aims to implement a Tic Tac Toe game using the Minimax algorithm. The Minimax algorithm is a decision-making algorithm that is commonly used in two-player games to make optimal decisions. The game will be developed using a programming language and will be designed to be played against the computer. The user will be able to choose the level of difficulty, and the game will have an AI opponent that will use the Minimax algorithm to make its moves. The project will involve implementing the game's logic and interface, as well as testing and debugging to ensure its functionality. The game will be implemented using the Model-View-Controller architecture and a graphical user interface. The Minimax algorithm will be used to determine the computer's moves, providing a competitive and intelligent opponent for players. The resulting game will be modular and extensible, allowing for future improvements and customization options. This project demonstrates the potential of artificial intelligence in game development and highlights the importance of careful design and implementation.

**TABLE OF CONTENTS**

**Sr.no. Title Page No.**

[ABSTRACT](#_bookmark0) 2

[TABLE OF CONTENTS](#_TOC_250001)  3

[LIST OF FIGURES](#_bookmark1) 4

|  |  |  |
| --- | --- | --- |
| **1** | [**INTRODUCTION**](#_bookmark3) | **5** |
| **2** | **ALGORITHM USED** | **6,7** |
| **3** | **IMPLEMENTATION** | **7,8,9** |
| **4** | **OUTPUT SCREEN** | **10** |
| **5** | **RESULTS** | **11** |
| **6** | **GAME TREE** | **11,12** |
| **7** | **CONCLUSION AND FUTURE ENHANCEMENT** | **13** |
| **8** | **REFERENCES** | **13** |

**LIST OF FIGURES**

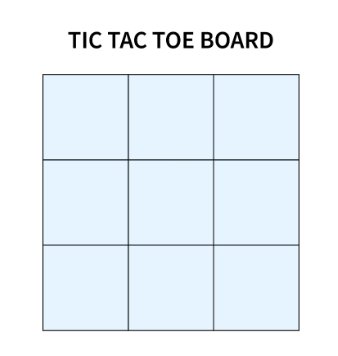
**Figure No. Figure Name Page No.**

|  |  |  |
| --- | --- | --- |
| 1 | **Tic tac toe** | 5 |
| 2 | **Game screen** | 10 |
| 3 | **Output screen** | 10 |
| 4 | **Game tree** | 11 |
| 5 | **Simplified game tree** | 12 |

**Introduction:**

To solve games using AI, we will introduce the concept of a game tree followed by 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.



**ALGORITHM USED:**

Minimax algorithm used in artificial intelligence to build a two player games, such as tic-tac-toe, checkers, chess and go. This games are known as zero-sum games, because in a mathematical representation: one player wins (+1) and other player loses (-1) or both of anyone not to win (0).

The Minimax algorithm is a decision-making algorithm commonly used in game theory and artificial intelligence to find the optimal move for a player, assuming that the opponent also plays optimally. The algorithm works by generating a game tree that represents all possible moves and outcomes for both players. The algorithm then evaluates the game tree by assigning a score to each leaf node of the tree, representing the final outcome of the game. The score is assigned based on some heuristic function that evaluates the desirability of a particular game outcome for a given player. The algorithm then recursively backpropagates the scores up the tree, alternating between minimizing and maximizing the score at each level of the tree, representing the turns of the two players. This process continues until the root of the tree is reached, which represents the optimal move for the player.

The Minimax algorithm is commonly used in games such as chess, checkers, and tic-tac-toe, where the game tree is small enough to be generated and evaluated in a reasonable amount of time. However, for larger games such as Go or real-time games such as Starcraft, more sophisticated algorithms such as Monte Carlo Tree Search are used instead.

**How does it works?**

The algorithm search, recursively, the best move that leads the Max player to win or not lose (draw). It consider 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 Algorithm**

The algorithm was studied by the book Algorithms in a Nutshell (George Heineman; Gary Pollice; Stanley Selkow, 2009). Pseudocode (adapted):

minimax(state, depth, player)

if (player = max) then

best = [null, -infinity]

else

best = [null, +infinity]

if (depth = 0 or gameover) then

score = evaluate this state for player

return [null, score]

for each valid move m for player in state s do

execute move m on s

[move, score] = minimax(s, depth - 1, -player)

undo move m on s

if (player = max) then

if score > best.score then best = [move, score]

else

if score < best.score then best = [move, score]

return best

end

**IMPLEMENTATION:**

Now we'll see each part of this pseudocode with Python implementation. The Python implementation is available at this repository. First of all, consider it:

board = [ [0, 0, 0], [0, 0, 0], [0, 0, 0] ]

MAX = +1

MIN = -1

The MAX may be X or O and the MIN 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 your worst score. If player is MAX, its score is -infinity. Else if player is MIN, its score is +infinity. Note: infinity is an alias for inf (from math module, in Python).

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)

Now we'll see the main part of this code that contains recursion.

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. Final algorithm:

def minimax(state, depth, player):

if player == MAX:

best = [-1, -1, -infinity]

else:

best = [-1, -1, +infinity]

if depth == 0 or game\_over(state):

score = evaluate(state)

return [-1, -1, score]

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

if player == MAX:

if score[2] > best[2]:

best = score

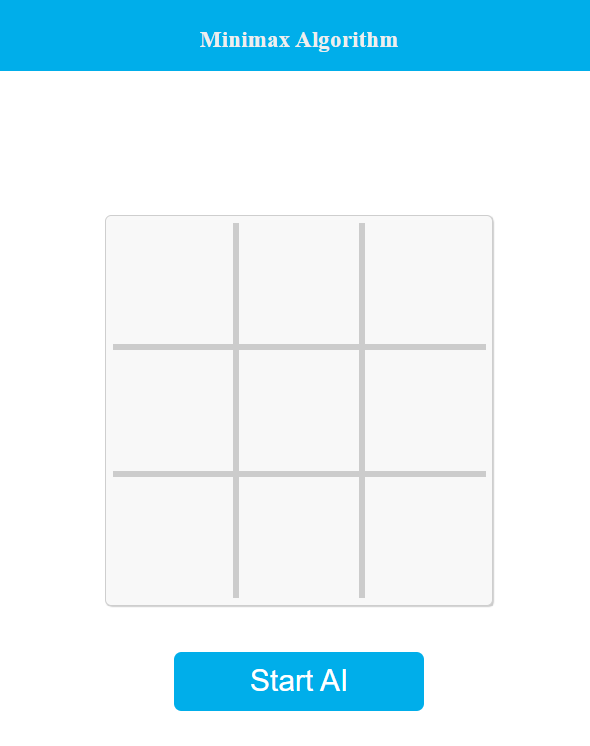
else:

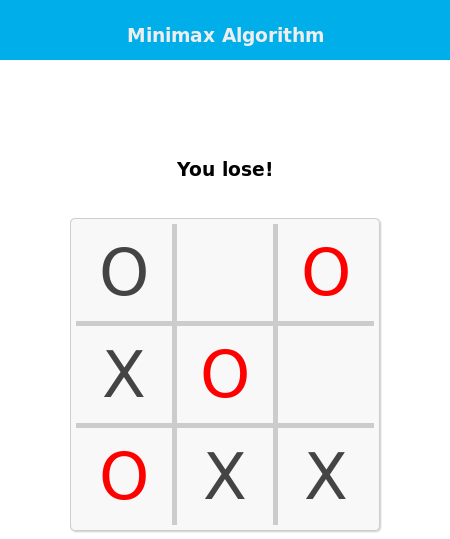
if score[2] < best[2]:

best = score

return best

**OUTPUT SCREEN:**

****



**RESULT:**

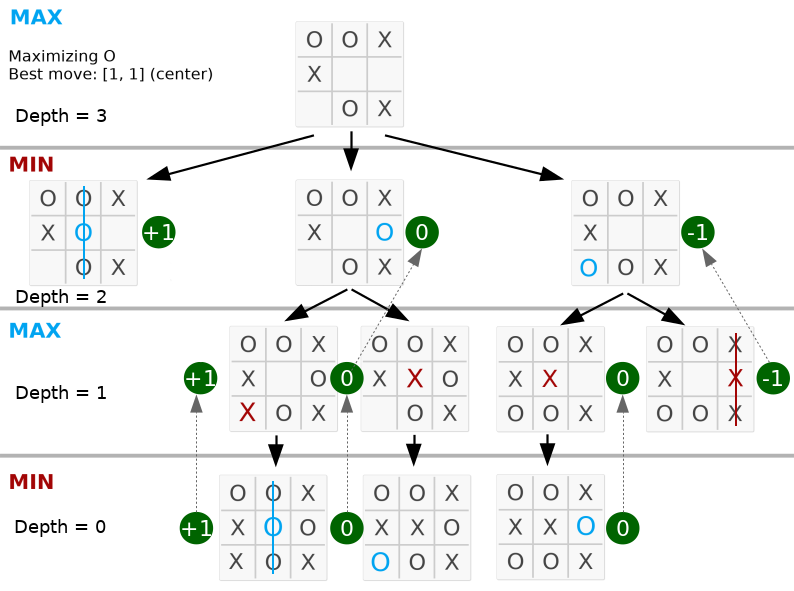
The computer opponent would use the Minimax algorithm to determine its moves, making it a challenging opponent for the human player.

The result of the project would also depend on the specific goals and requirements of the project, such as the user interface, the level of difficulty, the performance of the algorithm, and the overall user experience. If the project meets all the requirements and goals, it can be considered a success.

In addition, implementing the Minimax algorithm in a Tic-Tac-Toe game can be a good way to learn and practice the principles of artificial intelligence and game theory, and can serve as a foundation for more advanced AI applications in other games and domains.

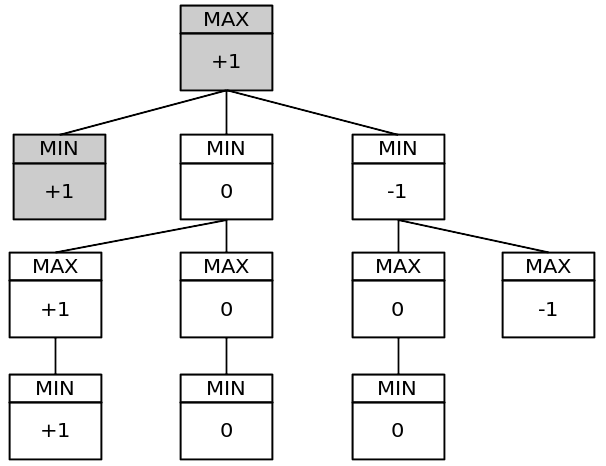
**Game Tree:**

Below, the best move is on the middle because the max value is on 2nd node on left.

****

Take a look that the depth is equal the valid moves on the board. The complete code is available in py\_version/.

**Simplified game tree:**



That tree has 11 nodes. The full game tree has 549.946 nodes! You can test it putting a static global variable in your program and incrementing it for each minimax function call per turn.

In a more complex game, such as chess, it's hard to search whole game tree. However, Alpha–beta Pruning is an optimization method to the minimax algorithm that allows us to disregard some branches in the search tree, because he cuts irrelevant nodes (subtrees) in search.

**CONCLUSION:**

Using the Minimax algorithm in a Tic-Tac-Toe game can result in a challenging opponent for human players, as the algorithm is designed to find the optimal move that maximizes the computer's chances of winning, assuming that the opponent also plays optimally. The project can be considered a success if it meets all the goals and requirements, such as providing a good user experience, having an intuitive user interface, and demonstrating good performance.

Furthermore, implementing the Minimax algorithm in a Tic-Tac-Toe game can be a valuable learning experience, as it allows for the practical application of principles of artificial intelligence and game theory. This can serve as a foundation for more advanced AI applications in other games and domains, providing an opportunity for further exploration and experimentation in the field.

**FUTURE ENHANCEMENT:**

There are several ways in which a Tic-Tac-Toe game using the Minimax algorithm can be enhanced in the future, including:

1. Adding more advanced AI techniques: While the Minimax algorithm is a strong starting point, there are more advanced AI techniques, such as Monte Carlo Tree Search, that can be used to further improve the computer player's performance.
2. Adding more difficulty levels: To provide a more challenging game for players of varying skill levels, additional difficulty levels can be added to adjust the depth of the search tree or other parameters used by the algorithm.
3. Adding additional game modes: To provide more variety and options for players, additional game modes, such as a "best of three" or "tournament" mode, can be added.
4. Improving the user interface: Improvements to the user interface can enhance the overall user experience and make the game more enjoyable to play.
5. Multiplayer support: Adding multiplayer support can allow for players to compete against each other online, providing an additional layer of challenge and engagement.

Overall, there are several ways to enhance a Tic-Tac-Toe game using the Minimax algorithm, and these enhancements can provide additional value and appeal to players.

**REFERENCES:**

1. Book: George T. Heineman; Gary Pollice; Stanley Selkow. Algorithms in a nutshell. O'Reilly, 2009.
2. Wikipédia: <https://en.wikipedia.org/wiki/Minimax>
3. Nanyang Technological University:

<https://www.ntu.edu.sg/home/ehchua/programming/java/JavaGame_TicTacToe_AI.html>