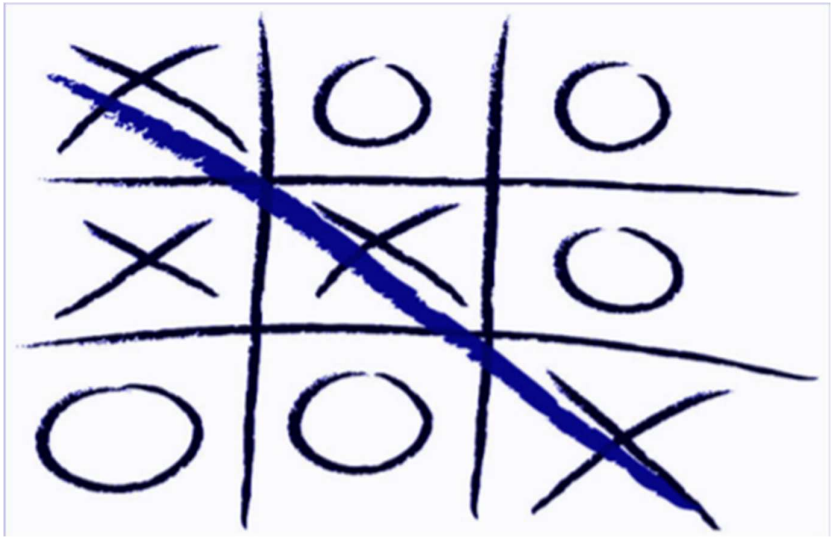
SYNOPSIS

PROJECT TITLE – TIC-TAC-TOE



BY - ROUNIK MONDAL AND VINAY NEGI

CERTIFICATE

ACKNOWLEDGEMENT

We would like to express my special thanks of gratitude to our teacher, Ms. Pooja Sharma as well as our principal Mrs. Rachana Kamath who gave us the golden opportunity to do this wonderful project on the topic Tic-Tac-Toe game, which also made us do a lot of research and we came to know about so many new things, We are really thankful to them.

Secondly, we would also like to thank our parents and friends who helped us a lot in finalizing this project within the limited time frame.

TABLE OF CONTENTS

- *Abstract*
- *Introduction*
- *Module/Algorithm Used*
- *Description of Module/Algo*
- *Limitation*
- *Reference*

ABSTRACT

Tic-tac-toe is a fun game played by two players. Before starting the game, a 3x3 square grid is formed using two vertical and two horizontal lines. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. In this study, we develop a computer program where a player can challenge artificial intelligent heuristic approach using minimax algorithm to win the game. We want to prove that artificial intelligent heuristic approach enables in giving a great performance for development of Tic-Tac-Toe game.

INTRODUCTION

WHAT IS TIC-TAC-TOE?

Tic-tac-toe is a simple, two-player game that, if played optimally by both players, will always result in a tie. The game is also called noughts and crosses or X's and O's.

Tic-tac-toe is a game that is traditionally played by being drawn on paper, and it can be played on a computer or on a variety of media. Other games, such as Connect 4, are based on this classic.

ORIGIN OF TIC-TAC-TOE

An early variation of tic-tac-toe was played in the Roman Empire, around the first century BC. "Tic-Tac-Toe" may also derive from "tick-tack", the name of an old version of backgammon first described in 1558. The US renaming of "noughts and crosses" to "tic-tac-toe" occurred in the 20th century.

MODULES/ALGORITHM USED

- Tkinter

- Minimax Algorithm

DESCRIPTION OF MODULES/ALGORITM

TKINTER

Tkinter is a standard library in python used for creating Graphical User Interface (GUI) for Desktop Applications. With the help of Tkinter developing desktop applications is not a tough task.

The primary GUI toolkit we will be using is Tk, which is Python's default GUI library. We'll access Tk from its Python interface called Tkinter.

MINIMAX ALGORITHM

WHAT IS MINIMAX?

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

HOW DOES IT WORK?

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 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
```

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 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).*

```
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.

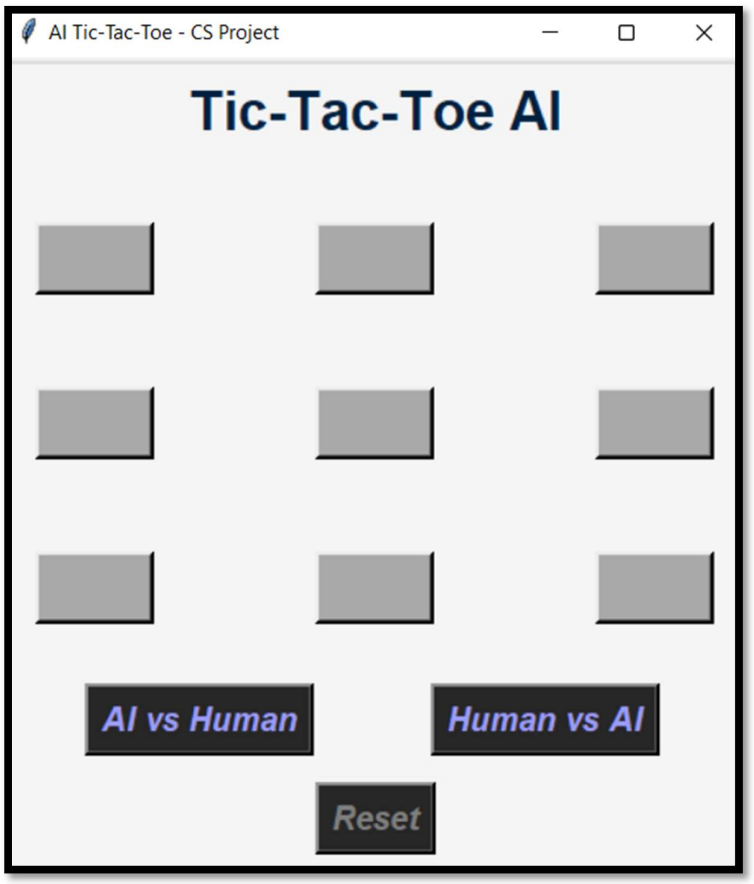
The next step is compare the score with best.

```
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

---

USER INTERFACE

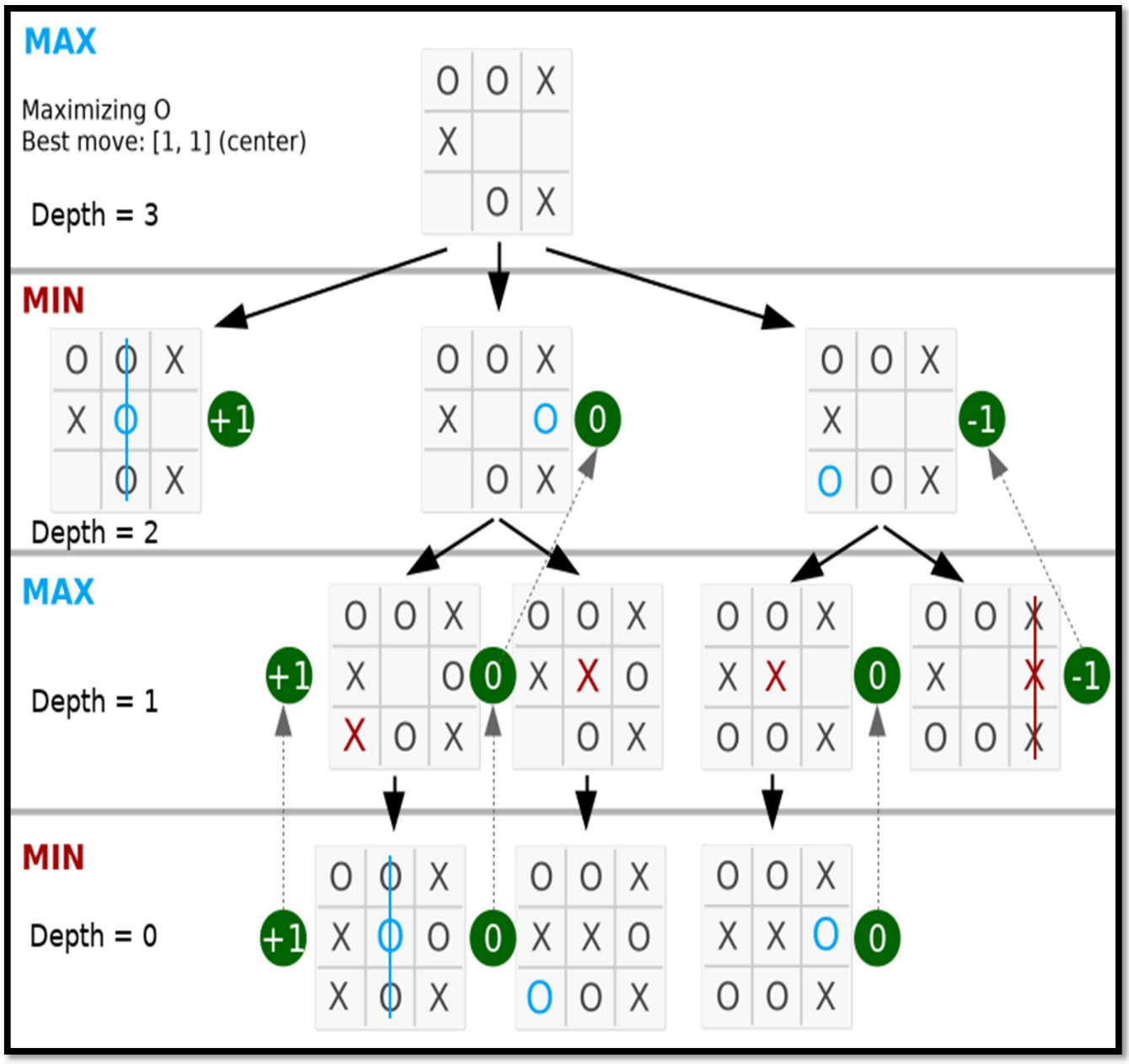


FINAL ALGORITHM

```
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
```

GAME TREE

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



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.

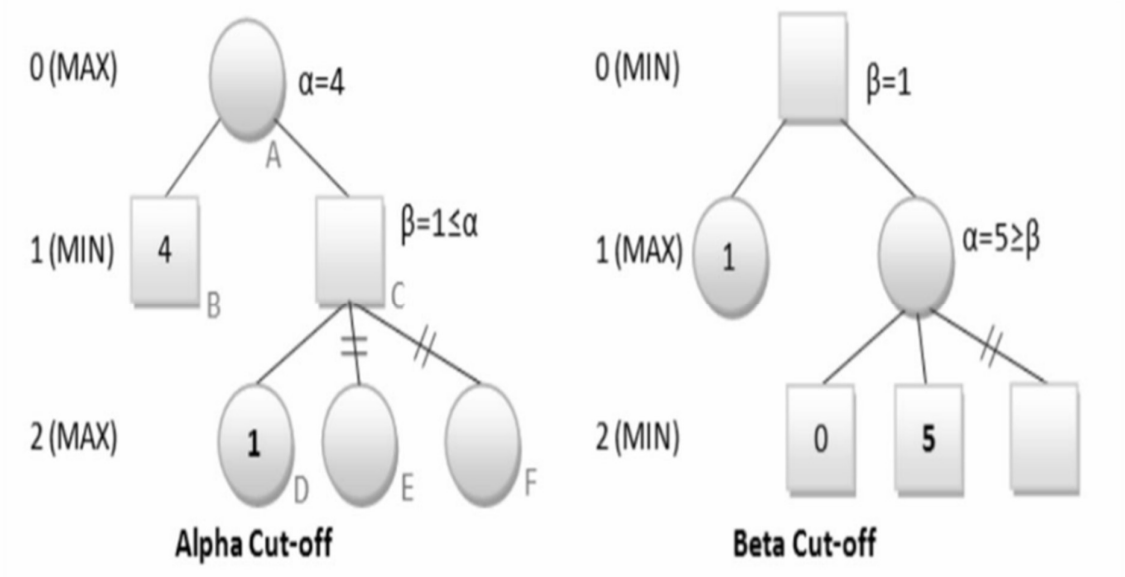
LIMITATION

IMPLEMENTING ALPHA-BETA PRUNING

We could have implemented *alpha-beta pruning* which is an optimization algorithm for the minimax algorithm. It reduces the computational time by a huge factor. It allows faster search and even goes into deeper levels in the game tree. It will cut off branches of game trees that does not require to be searched when there is a better movement exists

The steps of the Alpha-Beta Pruning algorithm fundamentally is summarized as below.

- **Search down the tree to the given depth.**
- **Once reaching the bottom, calculate the evaluation for this node. (i.e. it's utility)**
- **Backtrack, propagating values and paths**
- **Attain the minimum score of the Alpha.**



REFERENCE

- For Minimax Algorithm –
  - Book: George T. Heineman; Gary Police; Stanley Selkow. Algorithms in a nutshell. O'Reilly, 2009.

  - Wikipedia: https://en.wikipedia.org/wiki/Minimax

- For information about Tic-tac-Toe- https://en.wikipedia.org/wiki/Tic-tac-toe

---

EXTRAS

- Web-Version of this game has also been implemented
  - Follow this link: https://rounik-tictactoe.herokuapp.com

- To Download the project
  - Follow this link: Tic - Tac - Toe Game