In [1]:

```
#######################################################
#     LAB 6 - Tic Tac Toe + Minmax + TKinter UI       #
#     Author: SHIKHAR ASTHANA                         #
#     Roll No. : 2K22/AFI/24                          #
#     Subject: Principles of AI (Gull Kaur Ma'am)     #
#######################################################
```

In [2]:

```
#Importing header files
import pandas as pd
import numpy as np
from queue import Queue
import matplotlib.pyplot as plt
import networkx as nx
import time

import tkinter as tk
from tkinter import messagebox
from tkinter import *

from queue import Queue
from queue import LifoQueue
```

In [5]:

```
root = Tk()
root.title("DTU 2K22_AFI_24 Shikhar TicTacToe - MinMax")

global_counter = 1

button_variables = ["b1","b2","b3","b4","b5","b6","b7","b8","b9"]


#Board for the computer to keep track of which player moves where
board = {1: ' ', 2: ' ', 3: ' ',
         4: ' ', 5: ' ', 6: ' ',
         7: ' ', 8: ' ', 9: ' '}


#Function to check if the position at which player is wanting to play is empty or not
def spaceIsFree(position):
    if board[position] == ' ':
        return True
    else:
        return False



#Function to check whether the game is finished or not
def checkForWin():
    if (board[1] == board[2] and board[1] == board[3] and board[1] != ' '):
        return True
    elif (board[4] == board[5] and board[4] == board[6] and board[4] != ' '):
        return True
    elif (board[7] == board[8] and board[7] == board[9] and board[7] != ' '):
        return True
    elif (board[1] == board[4] and board[1] == board[7] and board[1] != ' '):
        return True
    elif (board[2] == board[5] and board[2] == board[8] and board[2] != ' '):
        return True
    elif (board[3] == board[6] and board[3] == board[9] and board[3] != ' '):
        return True
    elif (board[1] == board[5] and board[1] == board[9] and board[1] != ' '):
        return True
    elif (board[7] == board[5] and board[7] == board[3] and board[7] != ' '):
        return True
    else:
        return False


#Function to check if this is a draw
def checkDraw():
    for key in board.keys():
        if (board[key] == ' '):
            return False
    return True


#Function to assign X to the tile which player plays
def button_click(b,position):
    global global_counter

    #check if the tile is empty
    if b["text"] == " ":
        b["text"] = "X"
        insertLetter("X",position)

        #b.config(state="disabled")
    else:
        messagebox.showerror("InvalidMove","That tile has already been assigned a value. Please choose another tile")



#Function to Disable all buttons in the even of a win,lose or draw
def disable_buttons():
    global button_variables
    for item in button_variables:
        globals()[item]["state"] = "disabled"

    #update the UI
    root.update()


#Function to update the board for computer turn
def update_board():
    counter = 1
    global button_variables, board

    for item in button_variables:
        globals()[item]["text"] = board[counter]
        counter += 1

    #Update the UI
    root.update()


#Function to decide the move of the computer - Driver function for the MinMax Algorithm
def compMove():
    bestScore = -800
    bestMove = 0
    for key in board.keys():
        if (board[key] == ' '):

            board[key] = "O"
            score = minimax(board, False)
            board[key] = ' '

            if (score > bestScore):
                bestScore = score
                bestMove = key

    insertLetter("O", bestMove)
    return


#Function to help the MinMax Function to decide who won
def checkWhichMarkWon(mark):
    if board[1] == board[2] and board[1] == board[3] and board[1] == mark:
        return True
    elif (board[4] == board[5] and board[4] == board[6] and board[4] == mark):
        return True
    elif (board[7] == board[8] and board[7] == board[9] and board[7] == mark):
        return True
    elif (board[1] == board[4] and board[1] == board[7] and board[1] == mark):
        return True
    elif (board[2] == board[5] and board[2] == board[8] and board[2] == mark):
        return True
    elif (board[3] == board[6] and board[3] == board[9] and board[3] == mark):
        return True
    elif (board[1] == board[5] and board[1] == board[9] and board[1] == mark):
        return True
    elif (board[7] == board[5] and board[7] == board[3] and board[7] == mark):
        return True
    else:
        return False


#Function to implement the MinMax Algorithm - Recursion Implementation
def minimax(board, isMaximizing):
    #Terminal Conditions to end the recursion
    #Check if the computer won - send back a positive score
    if (checkWhichMarkWon("O")):
        return 1
    #check if the player won - send back a negative score
    elif (checkWhichMarkWon("X")):
        return -1
    #check if there is a draw - send back a neutral score
    elif (checkDraw()):
        return 0

    #First the bot will try to maximise its own score
    if (isMaximizing):
        bestScore = -800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = "O"
                score = minimax(board, False)
                board[key] = ' '
                if (score > bestScore):
                    bestScore = score
        return bestScore

    #The bot will try to minimise the score because it is estimating what the player might move
    else:
        bestScore = 800
        for key in board.keys():
            if (board[key] == ' '):
                board[key] = "X"
                score = minimax(board, True)
                board[key] = ' '
                if (score < bestScore):
                    bestScore = score
        return bestScore



#Function which converts UI inputs to moves in board variable and updates UIs with updated moves
def insertLetter(letter, position):
    global board
    if spaceIsFree(position):
        board[position] = letter

        if letter == 'O':
            update_board()

        if (checkDraw()):
            print("Draw!")
            messagebox.showerror("Draw - Game Over","It was a Draw.")
            disable_buttons()
            return


        if checkForWin():
            if letter == 'O':
                print("AI wins!")
                messagebox.showerror("You Lost - Game Over","The AI has won.")
                disable_buttons()
                return
            else:
                print("Player wins!")
                messagebox.showerror("You Won - Game Over","Congratulations! You have won. Hurray!")
                disable_buttons()
                return

        if letter == 'X':
            compMove()

        return
    else:
        messagebox.showerror("InvalidMove","This is not an acceptable move. Please choose another tile")


#Function to quit the Tkinter GUI
def quit_grid():
    root.destroy()


#Function to Reset the Tkinter GUI
def reset_grid():

    #Reseting the gobal counter variable to 1
    global board


    #Reseting the values to empty of the input grid
    #Reseting the colour of the input grid
    #Enabling the tiles of input grid
    global button_variables

    for item in button_variables:
        globals()[item]["text"] = " "
        globals()[item].config(bg="Silver")
        globals()[item]["state"] = "normal"

    board = {1: ' ', 2: ' ', 3: ' ',
             4: ' ', 5: ' ', 6: ' ',
             7: ' ', 8: ' ', 9: ' '}



#Building different frames for proper alignment
frame_title = LabelFrame(root, width = 200)
frame_input = LabelFrame(root, padx=50, pady=50)

#Assigning frames to proper grid positions
frame_title.grid(row=0, sticky = tk.W+tk.E)
frame_input.grid(row=1)



#Label for the title Frame
label_title = Label(frame_title, text= """SHIKHAR ASTHANA'S TIC-TAC-TOE SOLVER""", font=("Helvetica", 20))
label_title.pack()


#Label for the first frame
label_input = Label(frame_input, text= "Tic-Tac-Toe", font=("Helvetica",20))
label_input2 = Label(frame_input, text= "Your Turn. Choose the tile you want to play.", font=("Helvetica",10))
label_input.grid(row = 0, columnspan=3, sticky = tk.W+tk.E)
label_input2.grid(row = 1, columnspan=3, sticky = tk.W+tk.E)

#Build the grid for Tic-Tac-Toe
b1 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b1,1))
b2 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b2,2))
b3 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b3,3))

b4 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b4,4))
b5 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b5,5))
b6 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b6,6))

b7 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b7,7))
b8 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b8,8))
b9 = Button(frame_input, text= " ",  font=("Helvetica",20), height = 4, width = 6, bg = "Silver", command=lambda: button_click(b9,9))

#Assigning the buttons to the proper grid position
b1.grid(row=2, column=0)
b2.grid(row=2, column=1)
b3.grid(row=2, column=2)

b4.grid(row=3, column=0)
b5.grid(row=3, column=1)
b6.grid(row=3, column=2)

b7.grid(row=4, column=0)
b8.grid(row=4, column=1)
b9.grid(row=4, column=2)

#A Button to reset the whole UI
b_reset = Button(root, text = "Reset", font=("Helvetica", 20), height = 3, width = 6, bg = "Silver", command = lambda: reset_grid())
b_reset.grid(row = 3, column=0, sticky=tk.N+tk.S+tk.E+tk.W)

#A Button to quit
b_quit = Button(root, text = "Quit", font=("Helvetica", 20), height = 3, width = 6, bg = "Silver", command = lambda: quit_grid())
b_quit.grid(row = 4, sticky=tk.N+tk.S+tk.E+tk.W)



root.mainloop()
```

```
AI wins!
Draw!
AI wins!
```

In [ ]: