Tic-Tac-Toe AI Documentation

# Overview

This is a simple Tic-Tac-Toe game implemented in Python using the Streamlit library for a web interface. The game allows the player to compete against an AI, which uses the Minimax algorithm to calculate its optimal moves. The player plays as 'O', while the AI plays as 'X'. The game is played on a 3x3 grid, and the objective is to get three marks in a row, either horizontally, vertically, or diagonally.

# Features

- Interactive web interface for playing Tic-Tac-Toe.  
- Player plays as 'O', AI plays as 'X'.  
- AI calculates the best possible move using the Minimax algorithm.  
- Game ends when there is a winner or a draw.  
- Restart functionality to start a new game.

# Technologies Used

- Streamlit: Used for creating the interactive UI for the game.  
- NumPy: Used for handling the 3x3 game board and performing operations efficiently.

# Code Structure and Functionality

## 1. Board Initialization

def initialize\_board():  
 return np.full((3, 3), " ")

Purpose: Initializes the game board as a 3x3 grid filled with spaces (' '), representing empty spots.  
Returns: A 3x3 NumPy array representing the empty board.

## 2. Winner Check

def check\_winner(board):  
 for row in board:  
 if len(set(row)) == 1 and row[0] != " ":  
 return row[0]  
  
 for col in board.T:  
 if len(set(col)) == 1 and col[0] != " ":  
 return col[0]  
  
 if len(set([board[i, i] for i in range(3)])) == 1 and board[0, 0] != " ":  
 return board[0, 0]  
  
 if len(set([board[i, 2 - i] for i in range(3)])) == 1 and board[0, 2] != " ":  
 return board[0, 2]  
  
 return None

Purpose: Checks if there is a winner in the game by evaluating rows, columns, and diagonals.  
Returns: The winner ('X' or 'O') if there is one, otherwise None if no winner is found.

## 3. Full Board Check

def is\_full(board):  
 return " " not in board

Purpose: Checks if the board is full (i.e., there are no empty spaces).  
Returns: True if the board is full, otherwise False.

## 4. Evaluate Board for Minimax Algorithm

def evaluate(board):  
 winner = check\_winner(board)  
 if winner == "X":  
 return 1 # AI win  
 elif winner == "O":  
 return -1 # Player win  
 else:  
 return 0 # Draw

Purpose: Evaluates the current board state for use in the Minimax algorithm.  
Returns: 1 for AI win, -1 for player win, or 0 for a draw.

## 5. Minimax Algorithm

def minimax(board, depth, is\_maximizing):  
 score = evaluate(board)  
 if score != 0: # If the game is over, return the score  
 return score  
  
 if is\_full(board):  
 return 0 # Draw  
  
 if is\_maximizing:  
 best = -float("inf")  
 for i in range(3):  
 for j in range(3):  
 if board[i, j] == " ":  
 board[i, j] = "X"  
 best = max(best, minimax(board, depth + 1, False))  
 board[i, j] = " "  
 return best  
 else:  
 best = float("inf")  
 for i in range(3):  
 for j in range(3):  
 if board[i, j] == " ":  
 board[i, j] = "O"  
 best = min(best, minimax(board, depth + 1, True))  
 board[i, j] = " "  
 return best

Purpose: The Minimax algorithm simulates all possible moves to find the best move for the AI.  
Parameters: board (current game state), depth (current depth of recursion), is\_maximizing (True for AI, False for player).  
Returns: The best score for AI or player based on the current game state.

## 6. AI Move Calculation

def ai\_move(board):  
 best\_val = -float("inf")  
 best\_move = (-1, -1)  
  
 for i in range(3):  
 for j in range(3):  
 if board[i, j] == " ":  
 board[i, j] = "X"  
 move\_val = minimax(board, 0, False)  
 board[i, j] = " "  
 if move\_val > best\_val:  
 best\_val = move\_val  
 best\_move = (i, j)  
  
 return best\_move

Purpose: Calculates the AI’s next best move by using the Minimax algorithm.  
Returns: The coordinates of the best move (a tuple with row and column).

## 7. Streamlit Setup and UI

st.title("Tic-Tac-Toe")  
st.write("You are 'O' and the AI is 'X'. Try to beat the AI!")

Purpose: Sets the title and game instructions on the web interface using Streamlit.

## 8. Session State Management

if "board" not in st.session\_state:  
 st.session\_state.board = initialize\_board()  
if "game\_over" not in st.session\_state:  
 st.session\_state.game\_over = False  
if "winner" not in st.session\_state:  
 st.session\_state.winner = None  
if "turn" not in st.session\_state:  
 st.session\_state.turn = "O" # Player's turn starts first

Purpose: Initializes session state variables for the board, game status (game\_over), winner, and player’s turn.

## 9. Displaying the Game Board

def display\_board():  
 board = st.session\_state.board  
 for i in range(3):  
 cols = st.columns(3)  
 for j in range(3):  
 key = f"{i}-{j}" # Unique key for each button  
 if board[i, j] == " " and not st.session\_state.game\_over:  
 if st.session\_state.turn == "O": # Human player's turn  
 if cols[j].button(" ", key=key):  
 board[i, j] = "O"  
 winner = check\_winner(board)  
 if winner:  
 st.session\_state.winner = winner  
 st.session\_state.game\_over = True  
 elif is\_full(board):  
 st.session\_state.winner = "Draw"  
 st.session\_state.game\_over = True  
 else:  
 st.session\_state.turn = "X" # Switch to AI's turn  
 else: # AI's turn  
 cols[j].button(" ", key=key, disabled=True) # Disable button during AI's move  
 else:  
 # Display existing marks (O, X, or empty)  
 cols[j].button(board[i, j], key=key, disabled=True)

Purpose: Displays the 3x3 grid of buttons (each representing a board spot) for the player to interact with.

## 10. AI's Move Execution

if st.session\_state.turn == "X" and not st.session\_state.game\_over:  
 move = ai\_move(board)  
 board[move] = "X"  
 winner = check\_winner(board)  
 if winner:  
 st.session\_state.winner = winner  
 st.session\_state.game\_over = True  
 elif is\_full(board):  
 st.session\_state.winner = "Draw"  
 st.session\_state.game\_over = True  
 else:  
 st.session\_state.turn = "O" # Switch to player's turn

Purpose: Executes the AI's move. After the player makes their move, the AI uses the Minimax algorithm to determine and make the next move.

## 11. Restart Button

if st.button("Restart"):  
 st.session\_state.board = initialize\_board()  
 st.session\_state.game\_over = False  
 st.session\_state.winner = None  
 st.session\_state.turn = "O" # Reset to player's turn

Purpose: Allows the player to restart the game at any time by resetting the board, game over status, winner, and player’s turn.

# How to Run the Game

1. Install Streamlit and NumPy:  
 pip install streamlit numpy  
  
2. Save the Python code to a file, for example `tic\_tac\_toe.py`.  
  
3. Run the Streamlit app:  
 streamlit run tic\_tac\_toe.py  
  
4. Open the provided URL in your browser to play the game.

# Conclusion

This simple implementation of Tic-Tac-Toe provides an interactive experience where the player competes against an AI that calculates its moves using the Minimax algorithm. The game is designed to be easy to play, with features like automatic AI play, winner detection, and a restart option. The use of Streamlit makes it easy to build and deploy the game as a web app.