

Name :Talha Usman

Class:SU92-BSAIM-F24-064

Subject:AI-Lab

**Tic-Tac-Toe**

**Introduction**:  
This project is a Python implementation of a customizable Tic-Tac-Toe game using Object-Oriented Programming (OOP).   
It allows players to select the board size and play turn by turn until a win or draw occurs.   
The code is written in a class-based format which keeps the logic organized, makes the program easier to read,   
and helps in reusability for future improvements like AI or multiple modes.  
  
**Class and Initialization**:  
The program begins by defining a class named TicTacToe, which stores all the data and logic required to play the game.   
The \_\_init\_\_ method is a special function called a constructor. It initializes the game by creating a list called board,   
filled with empty spaces to represent available moves. It also stores the board size provided by the user and sets the   
current player to "X" by default, which means Player X always goes first.  
  
**Game Board Display and Moves**:  
The display board method prints the board in a clear format, with rows separated by lines and cells separated by vertical bars.   
This makes it easy for players to see the game state after each move. The make\_move method checks if a chosen position   
is empty before marking it with the current player's symbol. If the spot is already taken, it prints a warning message   
and asks for a new input. Once a move is successfully made, the switch\_player method changes the turn to the other player.  
  
**Winner and Draw Checking:**  
The check winner method checks all possible winning combinations. It first checks rows, then columns, and finally the two diagonals.   
If all cells in a row, column, or diagonal have the same symbol (X or O), the game declares that player as the winner.   
If no winner is found and there are no empty spaces left, the is\_draw method declares the game a draw.  
  
**Game Loop**:  
The play method controls the game flow. It runs inside a while loop which keeps going until a winner is found or the game ends in a draw.   
Inside this loop, players are asked to enter their move. A try-except block ensures that only valid numbers are entered   
and prevents the game from crashing due to invalid inputs. After each move, the board is displayed, and the program checks   
for a winner or draw. If neither occurs, the turn switches to the other player and the loop continues.  
  
**Conclusion:**  
This program is a simple but powerful example of how Object-Oriented Programming can be used to create interactive games.   
It uses key OOP concepts like classes, methods, and data encapsulation to make the game well-structured.   
It also demonstrates how Python lists, loops, and conditionals work together to create game logic.   
With small modifications, this code can be extended to include features like AI opponents, scoring systems, or larger board sizes.

**Flowchart of the code:**

