TIC-TAC-TOE

ClassicTicTacToe Class Implements the classical game : Defined functions for make_move, check_winner, check_draw, etc.

In [17]:
```
import random
from random import randint

class ClassicTicTacToe:
    def __init__(self):
        #Create empty board (3x3)
        self.board = [[' ' for _ in range(3)] for _ in range(3)]
        self.current_player = 'X'
        # Assign current player to 'X' initially (X always starts first)

    # Assign next player move at [row][col]
    def make_move(self, row, col):
        if self.board[row][col] == ' ':
            self.board[row][col] = self.current_player
            #TO switch to next player (O if X is current_player)
            self.current_player = 'O' if self.current_player == 'X' else 'X'
            return True
        return False

    def check_winner(self):   #Function to return if there is a check in current_move
        # Implement the logic to check for a win
        for i in range(3):
            # For Rows:
            if self.board[i][0] == self.board[i][1] == self.board[i][2] != ' ':
                return self.board[i][0]
            # For Columns:
            if self.board[0][i] == self.board[1][i] == self.board[2][i] != ' ':
                return self.board[0][i]

            #Diagonals:
        if self.board[0][0] == self.board[1][1] == self.board[2][2] != ' ':
            return self.board[0][0]
        if self.board[0][2] == self.board[1][1] == self.board[2][0] != ' ':
            return self.board[0][2]
        # returns current_player if check
        return None

    def check_draw(self):
        return all(self.board[row][col] != ' ' for row in range(3) for col in range(3))

    def display_board(self):
        for row in self.board:
            print('|'.join(row))
            print('-' * 5)
    def play(self):
        while True:
            self.display_board()
            print(f"Player {self.current_player}, it's your turn.")
            row = int(input("Enter row (0-2): "))
            col = int(input("Enter column (0-2): "))

            if self.make_move(row, col):
                winner = self.check_winner()
                if winner:
                    self.display_board()
                    print(f"Player {winner} wins!")
                    break
```

```
                elif self.check_draw():
                    self.display_board()
                    print("It's a draw!")
                    break
```

In [16]:
```
#  2 Player Game :
if __name__ == "__main__":
    game = ClassicTicTacToe()
    game.play()
```

```
 | |
-----
 | |
-----
 | |
-----
Player X, it's your turn.
Enter row (0-2): 0
Enter column (0-2): 2
 | |X
-----
 | |
-----
 | |
-----
Player O, it's your turn.
Enter row (0-2): 2
Enter column (0-2): 2
 | |X
-----
 | |
-----
 | |O
-----
Player X, it's your turn.
Enter row (0-2): 1
Enter column (0-2): 2
 | |X
-----
 | |X
-----
 | |O
-----
Player O, it's your turn.
Enter row (0-2): 2
Enter column (0-2): 1
 | |X
-----
 | |X
-----
 |O|O
-----
Player X, it's your turn.
Enter row (0-2): 0
Enter column (0-2): 1
 |X|X
-----
 | |X
-----
 |O|O
-----
Player O, it's your turn.
Enter row (0-2): 2
Enter column (0-2): 0
 |X|X
```

```
-----
 | |X
-----
O|O|O
-----
Player O wins!
```

SImulation for n games:

We use randint() method to generate random set of moves in the board. And simulate this for n=1000 games, updating win probabilities.

In [13]:
```python
#Simulation for n games:

def simulate_games(n):
    x_wins = 0
    o_wins = 0
    draws = 0

    for _ in range(n):
        # Game object from ClassicTicTacToe Class
        game = ClassicTicTacToe()
        while True:
            #Randomly choosing [row][col] for next move
            row = randint(0, 2)
            col = randint(0, 2)

            if game.make_move(row, col):
                winner = game.check_winner()
                if winner:
                    if winner == 'X':
                        x_wins += 1
                    else:
                        o_wins += 1
                    break
                elif game.check_draw():
                    draws += 1
                    break

#RESULTS :

    print(f"Simulated {n} games (X starts first):")
    print(f"X wins: {x_wins}")
    print(f"O wins: {o_wins}")
    print(f"Draws: {draws}")
    print()

    print(f"Probabilities in {n} games (X starts first):")
    print(f"X wins: {x_wins/n}")
    print(f"O wins: {o_wins/n}")
    print(f"Draw: {draws/n}")


n_games = 1000
simulate_games(n_games)
```

```
Simulated 1000 games (X starts first):
X wins: 586
O wins: 290
Draws: 124

Probabilities in 1000 games (X starts first):
X wins: 0.586
```

```
O wins: 0.29
Draw: 0.124
```