✅ Project: Shared-Tac-Toe — Two-Process Tic-Tac-Toe Using Shared Memory and Semaphores

🎯 Objective:

Implement a **two-player Tic-Tac-Toe game** where each player is a **separate process**. The game state is stored in **shared memory**, and synchronization is handled with **a named semaphore or a POSIX mutex with a condition variable** to coordinate turns.

No threads. No signals. Just pure **inter-process synchronization** and **safe shared memory use**.

📐 High-Level Design:

- Two **independent processes**, each launched with a command-line argument: `"player1"` or `"player2"`
- Both connect to a **shared memory segment** that contains the game board and synchronization variables
- Turns are coordinated using either:
    - A **POSIX named semaphore**, or
    - A **mutex + condition variable** pair inside the shared memory
- The game ends when one player wins or the board is full
- Each player draws the board and interacts via the terminal

📦 Shared Memory Structure:

```
#define BOARD_SIZE 3

typedef struct {
    char board[BOARD_SIZE][BOARD_SIZE]; // Game board: 'X', 'O', or ''
    int current_turn; // 1 for player1, 2 for player2
    int move_count;
    int game_over; // 1 when game ends
    char winner;   // 'X', 'O', or '-' for draw
    // synchronization primitives
} SharedGame;
```

🕹️ Player Behavior:

Each process does the following:

1. **Attach to shared memory**
2. **Wait until it's their turn**
3. **Lock the mutex**
4. **Read and draw the current board**
5. **Prompt user for a move**
6. **Validate and apply move**
7. **Update the move count, check for win/draw**
8. **Switch turn to the other player**
9. **Signal the condition variable and unlock**
10. **If the game is over**, display the winner and exit

📋 Required Features:

- User plays from **terminal input** (row, column)
- The board is redrawn after every move
- The game ends with a **clear message**: "Player X won", "Draw", etc.
- Shared memory is created by the first player and destroyed by the last one
- Proper cleanup of **mutexes, condition variables, and shared memory**

🧠 What Students Will Learn:

- How to safely share memory across processes
- How to synchronize without busy-waiting
- How to use **mutexes and condition variables across processes** (`PTHREAD_PROCESS_SHARED`)
- Safe initialization and destruction of shared memory objects
- Clean process coordination

🧪 Testing Notes:

Students should test by launching two terminals:

```
./tic_tac_toe player1
# and in another terminal:
./tic_tac_toe player2
```

They should be able to see their turns alternate and play interactively.