Blockchain Engineer – Round 2 Integration Challenge

Objective

This round is designed to test your **critical thinking**, **blockchain integration skills**, and **ability to work with real game source code**.

You will integrate the APIs and smart contracts you built in **Round 1** into a real **two-player game** with live matchmaking, token staking, and winner payouts on-chain.

Instructions & Rules

1. Task Overview

- Find any 2-player game with open-source code (we have provided a GitHub repo for reference, but you are strongly encouraged to find your own unique game).
- Integrate your **TriX system APIs and smart contracts** into the game flow:
 - 1. Buy Game Tokens (GT) using USDT.
 - 2. Matchmake between two players.
 - 3. Stake GT before starting the game.
 - 4. Play the game in real-time.
 - 5. **Determine the winner** and automatically transfer 2× stake GT to them.
 - 6. Record the transaction on blockchain and provide an explorer link.

2. Matchmaking Requirement

- When one player joins, your system must automatically search for another online player with the same stake.
- Both players are then placed into the same game room.
- Only after both have staked GT should the game start.

3. Game Source Code

- Reference Repo (for practice only): <u>Example Multiplayer Games GitHub</u> (Do not copy directly; it's for understanding structure only.)
- You may also:
 - Search GitHub, Itch.io, or OpenGameArt for open-source two-player games.
 - Find mini-games from public game portals and scrape or download their source code.
- The more **unique and challenging** the game you choose, the better.

4. AI Usage Policy

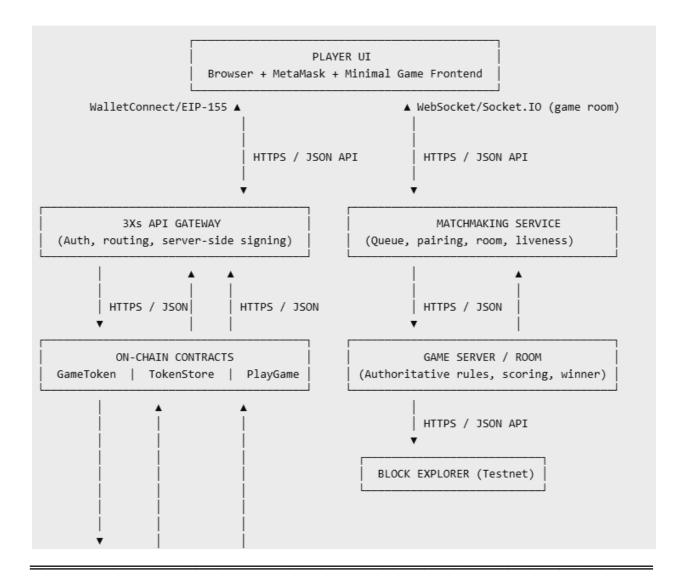
- You may research using Al, but you cannot copy-paste Al-generated code directly.
- If your submission contains directly copy-pasted AI code without modification, **you will** be disqualified.
- The focus is on your own logic, integration ability, and understanding.

5. Deliverables

Your submission must include:

- 1. Working Game with complete TriX integration
- 2. Blockchain transaction proof (buy, stake, payout).
- 3. Matchmaking demo.
- 4. **GitHub repo link** for your final code.
- 5. Vercel Link of the Game + Entire Flow has been integrated
- 6. Short README explaining:
 - Game source reference
 - API/contract integration points
 - Matchmaking logic
 - How to run the game locally

System Flow - End-to-End



(0) CONNECT

Player opens app → Connect Wallet (MetaMask) → Show address & GT/USDT balances.

(1) BUY GT (USDT → GT)

Player UI —(USDT.approve + buy)→ 3Xs API Gateway → TokenStore.buy(usdtAmount) [ON-CHAIN TX On success: GameToken.mint(player, gtOut) → UI refreshes GT balance.

(2) JOIN QUEUE & MATCH

Player clicks "Find Match" with chosen stake.

Player UI → Matchmaking Service: enqueue {address, stake}.

If peer with same stake exists → pair instantly:

- Assign matchId
- Create game room (Socket.IO)
- Notify both players
- 3Xs API Gateway → PlayGame.createMatch(matchId, p1, p2, stake) [ON-CHAIN TX]
- (3) STAKE (BOTH MUST STAKE)

Each Player UI:

- GameToken.approve(PlayGame, stake) [ON-CHAIN TX]
- PlayGame.stake(matchId) [ON-CHAIN TX]

When both confirmed → Start Game.

(4) PLAY GAME (LIVE ROOM)

Both clients join the room.

Game Server runs state; determines winner.

(5) COMMIT RESULT → PAYOUT (WINNER GETS 2×STAKE)

Game Server → 3Xs API Gateway → PlayGame.commitResult(matchId, winner) [ON-CHAIN TX] PlayGame transfers 2×stake GT to winner.

(6) PROOF & HISTORY

UI shows tx hash + Explorer Link.