

Decentralized Whot Game - User Stories & On-Chain Requirements

By:

Godwin Idemudia

rustoshidev@gmail.com

12th October, 2025.

Part A: User Stories & On-Chain Requirement Document

1. Core User Personas

- African Casual Gamers
- Web3 Enthusiasts
- Developer (Administrator)

2. Core Function Map

User Type 1: African Casual Gamers

Description: Players who enjoy quick, fun, culturally familiar games like *Whot* and are interested in earning small rewards while playing.

Core Functions / Interactions:

- Create or connect a Solana wallet to access the game.
- Join quick casual matches (public or private).
- Play *Whot* in real-time against other players.
- Earn small token rewards for wins or streaks.
- View basic stats (win/loss, game history).
- Refer friends or share game links.
- Withdraw or use earned rewards within the app.

User Type 2: Web3 Enthusiasts

Description: Users already familiar with crypto, NFTs, or DeFi who want to experience decentralized gaming.

Core Functions / Interactions:

- Connect existing Solana wallets (Phantom, Backpack, etc.).
- Verify game fairness through on-chain VRF results (Switchboard/Magicblock).
- Explore smart contract transparency (match logs, rewards, VRF proofs).
- Trade or showcase earned NFTs or badges.
- Participate in governance or feedback polls.
- Bridge tokens or use other Solana dApps connected to the platform.

User Type 3: Developer (Administrator)

Description: Developers or maintainers responsible for system management, fair play enforcement, and new feature deployment.

Core Functions / Interactions:

- Deploy and upgrade smart contracts using Anchor.
- Configure VRF and ephemeral rollup settings.
- Monitor gameplay data and transaction logs.

- Manage game logic updates, and reward.
- Debug or review failed transactions.
- Moderate reports of cheating or bugs.
- Add or remove in-game items and balance reward distributions.

3. User Stories

Story 1: Player joins a game

A player opens the Whot app and joins a game lobby to play against another online player.

Story 2: System starts a fair game

The system starts the match and automatically shuffles and shares cards using a fair random process that both players can verify.

Story 3: Players take turns playing

Each player takes turns placing cards until one player wins the match.

Story 4: System shows the winner

The app shows who won the match and displays proof that the game was fair.

Story 5: Winner receives reward

After the match ends, the system automatically sends a token reward to the winner's connected wallet and displays the transaction result.

4. Potential On-Chain Requirements

Story 1: Player joins a game

- Create or reference an on-chain match account that represents a new or existing Whot game.
- Store player public keys (up to 4 players) inside the match account.
- Initialize match state: waiting, active, completed.
- Lock or escrow each player's stake amount (if applicable) in the match account.
- Verify both players signed and funded the transaction before starting the match.

Story 2: System starts a fair game

- Request a verifiable random number from Switchboard/Magicblock VRF to shuffle cards.
- Store the VRF proof and the resulting seed in the match account for auditability.
- Use the random seed to generate a deterministic deck order (e.g., via on-chain or verified off-chain computation).
- Mark the match as "active" once the fair shuffle proof is confirmed.
- Emit an on-chain event or log containing the VRF result for client verification.

Story 3: Players take turns playing

- Maintain match state transitions (whose turn, last card played, cards remaining).
- Validate each player's move against Whot rules stored on-chain (e.g., card number/shape match).
- Prevent illegal moves using on-chain checks and signature verification.
- Update match state after every valid move and emit a state-update event.
- End the match automatically when a winning condition is detected.

Story 4: System shows the winner

- When the game-ending condition is met, store the winner's public key in the match account.
- Verify the result deterministically using stored move history or counters.
- Write a finalized match result (winner, duration, proof ID) to chain.
- Emit an event "MatchCompleted"

Story 5: Winner receives reward

- Smart contract releases escrowed tokens to the winner's wallet.
- Update match account to mark rewards_distributed = true.
- Record reward amount, token mint address, and transaction ID.
- Emit an on-chain log confirming the payout.
- Optionally close the match account to reclaim rent or clean up state.

Part B: Process Appendix

Part A: Initial User & Function Mapping

1. Manual User Brainstorming

Direct Users

- African Casual Gamers (18-35): Play Whot/Ludo for fun and small earnings, using Naira on-ramps.
- Web3 Enthusiasts (20-40): Value transparent, skill-based PvP gaming with token rewards.
- Diaspora Africans (18-45): Seek nostalgic Whot connections with Web3 rewards.

Indirect Users/Beneficiaries

- Watch public Whot matches or tournaments, boosting engagement.

Administrators/Moderators

- Developer(Me), build and manage the platform.

Stakeholders

- Token holders that invested in the native token's value and platform adoption.

2. AI-Assisted User Prioritization

Prompt Used: My project's value proposition is The core value proposition of the decentralized Whot platform is to deliver a transparent, accessible, and culturally resonant PvP gaming experience that empowers players to compete and earn rewards with confidence. Built on the Solana blockchain, it uses Switchboard VRF for verifiably fair gameplay and MagicBlock ephemeral rollups for fast, low-cost transactions, addressing trust and cost barriers in traditional Web2 gaming. Easy on-ramp and off-ramp options for local currencies ensure casual gamers can participate without crypto expertise, broadening accessibility. Here is a brainstormed list of all potential user types:

Direct Users

African Casual Gamers (18-35): Play Whot/Ludo for fun and small earnings, using Naira on-ramps.

Web3 Enthusiasts (20-40): Value transparent, skill-based PvP gaming with token rewards.

Indirect Users/Beneficiaries

Watch public Whot matches or tournaments, boosting engagement.

Administrators/Moderators

Developer(Me), build and manage the platform.

Stakeholders

Token holders that invested in the native token's value and platform adoption.

Based on the value proposition, which 2-5 of these user types are the most critical to focus on for an initial Proof-of-Concept? For each user you recommend, provide a brief rationale explaining why they are essential for proving the project's core value.

AI Recommendations:

- *African Casual Gamers – Primary proof of accessibility and ease of onboarding.*
- *Web3 Enthusiasts – Secondary testers validating blockchain transparency and evangelizing the project.*
- *Developer (Administrator) – Required for platform operation and on-chain management during the POC.*

Final Prioritized Users

- African Casual Gamers
- Web3 Enthusiasts
- Developer (Administrator)

Rationale:

I accepted the AI's recommendations almost entirely because they align with my POC goals of proving fairness, low cost, and accessibility.

I chose **not to include Diaspora Africans or Token Holders** at this stage since their engagement depends on a proven base game and active player ecosystem. They will be targeted post-POC once core fairness and on-ramp functionality are validated.

3. Core Function Mapping

Prompt Used:

For a project with this value proposition "The core value proposition of the decentralized Whot platform is to deliver a transparent, accessible, and culturally resonant PvP gaming experience that empowers players to compete and earn rewards with confidence. Built on the Solana blockchain, it uses Switchboard VRF for verifiably fair gameplay and MagicBlock ephemeral rollups for fast, low-cost transactions, addressing trust and cost barriers in traditional Web2 gaming" and focusing on these prioritized user types

African Casual Gamers

Web3 Enthusiasts

Developer (Administrator)

help map out the key functions or interactions each user would need to perform.

Output:

User Type 1: African Casual Gamers

Description: Players who enjoy quick, fun, culturally familiar games like Whot and are interested in earning small rewards while playing.

Core Functions / Interactions:

- Create or connect a Solana wallet to access the game.
- Join quick casual matches (public or private).
- Play Whot in real-time against other players.
- Earn small token rewards for wins or streaks.
- View basic stats (win/loss, game history).
- Refer friends or share game links.
- Withdraw or use earned rewards within the app.

Rationale:

This group represents the largest potential user base — people familiar with Whot but not necessarily blockchain. They need simple, frictionless access and clear incentives.

User Type 2: Web3 Enthusiasts

Description: Users already familiar with crypto, NFTs, or DeFi who want to experience decentralized gaming.

Core Functions / Interactions:

- Connect existing Solana wallets (Phantom, Backpack, etc.).
- Verify game fairness through on-chain VRF results (Switchboard).
- Explore smart contract transparency (match logs, rewards, VRF proofs).
- Trade or showcase earned NFTs or badges.
- Participate in governance or feedback polls.
- Bridge tokens or use other Solana dApps connected to the platform.

Rationale: They serve as early adopters who validate the game's fairness and decentralization, providing credibility within the Web3 ecosystem.

User Type 3: Developer (Administrator)

Description:

Developers or maintainers responsible for system management, fair play enforcement, and new feature deployment.

Core Functions / Interactions:

- Deploy and upgrade smart contracts using Anchor.
- Configure VRF and ephemeral rollup settings.
- Monitor gameplay data and transaction logs.
- Manage game logic updates, and reward.
- Debug or review failed transactions.
- Moderate reports of cheating or bugs.
- Add or remove in-game items and balance reward distributions.

Rationale:

This group maintains system integrity, ensuring fair gameplay, smooth operations, and continual innovation within the decentralized infrastructure.

4. Deriving Core PoC Requirements

Manual Task: Identify Top 1–2 Critical User Interactions

The most critical interactions for PoC are:

1. Casual Gamer joins and plays a Whot match with another player, with verifiably fair results.

This demonstrates real-time multiplayer functionality, fairness verification (Switchboard/MB VRF), and low-cost execution (MagicBlock ephemeral rollups).

2. Player receives rewards automatically after a match ends.

This validates the token reward logic, wallet connection, and transaction flow on Solana.

AI Prompt:

Based on these top 2 critical user interactions:

"

1. A player joins and plays a Whot match with another player with verifiably fair results, and

2. The player receives token rewards after the match ends "

What are the key technical requirements needed to build a proof-of-concept for these interactions?

AI Output:

Key Technical Requirements for PoC:

1. **Wallet Integration (Solana)**
 - Support wallet connection (e.g., Phantom or Backpack).
 - Enable signing of transactions for joining and completing matches.
2. **Smart Contract (Anchor) for Match Logic**
 - Handle player registration, game state storage, and result resolution.
 - Distribute rewards based on verified outcomes.
 - Manage escrowed tokens during matches or tournaments.
3. **Switchboard VRF Integration**
 - Generate verifiable random values for fair card shuffling and draw order.
 - Include VRF proofs on-chain to prevent manipulation.
4. **MagicBlock Ephemeral Rollups**
 - Enable fast, low-cost match interactions and state updates.
 - Reduce on-chain transaction costs while maintaining verifiable results.
5. **Game Client (Frontend)**
 - Allow users to join lobbies, view opponents, and play the Whot game.
 - Display real-time actions, scores, and match completion status.
 - Integrate wallet connection and reward claim UI.
6. **Reward Distribution System**
 - Automatically send tokens or points to the winning player.
 - Log match history and verify distribution events on-chain.
7. **Developer/Admin Tools**
 - Ability to reset matches or monitor VRF outputs for debugging.
 - Dashboard or CLI for managing PoC configurations and matches.

These requirements form the foundation for demonstrating a minimum viable decentralized gameplay loop:

- Two players join → VRF ensures fair deck → Match resolves → Smart contract distributes rewards.

That end-to-end flow proves transparency, fairness, and low-cost accessibility – the essence of the Decentralized Whot value proposition.

Part B: Initial User & Function Mapping

1. Critique & Refine User Stories / Requirements:

AI Prompt: "Review my core user functions/stories

[User Type 1: African Casual Gamers

Description: Players who enjoy quick, fun, culturally familiar games like Whot and are interested in earning small rewards while playing.

Core Functions / Interactions:

Create or connect a Solana wallet to access the game.

Join quick casual matches (public or private).

Play Whot in real-time against other players.

Earn small token rewards for wins or streaks.

View basic stats (win/loss, game history).

Refer friends or share game links.

Withdraw or use earned rewards within the app.

Rationale:

This group represents the largest potential user base – people familiar with Whot but not necessarily blockchain. They need simple, frictionless access and clear incentives.

User Type 2: Web3 Enthusiasts

Description: Users already familiar with crypto, NFTs, or DeFi who want to experience decentralized gaming.

Core Functions / Interactions:

Connect existing Solana wallets (Phantom, Backpack, etc.).

Verify game fairness through on-chain VRF results (Switchboard).

Explore smart contract transparency (match logs, rewards, VRF proofs).

Trade or showcase earned NFTs or badges.

Participate in governance or feedback polls.

Bridge tokens or use other Solana dApps connected to the platform.

Rationale: They serve as early adopters who validate the game's fairness and decentralization, providing credibility within the Web3 ecosystem.

User Type 3: Developer (Administrator)

Description:

Developers or maintainers responsible for system management, fair play enforcement, and new feature deployment.

Core Functions / Interactions:

Deploy and upgrade smart contracts using Anchor.

Configure VRF and ephemeral rollup settings.

Monitor gameplay data and transaction logs.

Manage tournaments, game logic updates, and reward pools.

Debug or review failed transactions.

Moderate reports of cheating or bugs.

Add or remove in-game items and balance reward distributions.

Rationale:

This group maintains system integrity, ensuring fair gameplay, smooth operations, and continual innovation within the decentralized infrastructure.]

and requirements

[Key Technical Requirements for PoC:

Wallet Integration (Solana)

Support wallet connection (e.g., Phantom or Backpack).

Enable signing of transactions for joining and completing matches.

Smart Contract (Anchor) for Match Logic

Handle player registration, game state storage, and result resolution.

Distribute rewards based on verified outcomes.

Manage escrowed tokens during matches or tournaments.

Switchboard VRF Integration

Generate verifiable random values for fair card shuffling and draw order.

Include VRF proofs on-chain to prevent manipulation.

MagicBlock Ephemeral Rollups

Enable fast, low-cost match interactions and state updates.

Reduce on-chain transaction costs while maintaining verifiable results.

Game Client (Frontend)

Allow users to join lobbies, view opponents, and play the Whot game.

Display real-time actions, scores, and match completion status.

Integrate wallet connection and reward claim UI.

Reward Distribution System

Automatically send tokens or points to the winning player.

Log match history and verify distribution events on-chain.

Developer/Admin Tools

Ability to reset matches or monitor VRF outputs for debugging.

Dashboard or CLI for managing PoC configurations and matches. These requirements form the foundation for demonstrating a minimum viable decentralized gameplay loop:

Two players join → VRF ensures fair deck → Match resolves → Smart contract distributes rewards.

That end-to-end flow proves transparency, fairness, and low-cost accessibility – the essence of the Decentralized Whot value proposition. Considering my project's refined value proposition [The core value proposition of the decentralized Whot platform is to deliver a transparent, accessible, and culturally resonant PvP gaming experience that empowers players to compete and earn rewards with confidence. Built on the Solana blockchain, it uses Switchboard VRF for verifiably fair gameplay and MagicBlock ephemeral rollups for fast, low-cost transactions, addressing trust and cost barriers in traditional Web2 gaming.

Easy on-ramp and off-ramp options for local currencies ensure casual gamers can participate without crypto expertise, broadening accessibility.] Do these stories truly hit the mark? Are the requirements granular enough to map to specific technical components (e.g., database schemas, API endpoints, specific blockchain programs)? What's missing or unclear?"

AI Output:

Overall alignment:

The user stories map well to the value proposition – they clearly express accessibility (for casual gamers), competitiveness (for pro players), decentralization (for Web3 users), and system integrity (for developers). They demonstrate the full lifecycle of interaction on a decentralized PvP platform.

However, several improvements are needed for granularity and technical traceability:

1. Action granularity:

- Many stories combine multiple steps (e.g., "join and play a match"). Each discrete action should be its own story to map directly to a smart-contract instruction (`join_match`, `start_match`, `submit_move`, `resolve_match`, `claim_reward`).

2. Outcome specificity:

- Stories should explicitly describe **expected outcomes** (e.g., "after the player wins, rewards are transferred and recorded on-chain").
- This clarifies what state change or transaction confirmation should follow each interaction.

3. Technical linkage gaps:

- Requirements should reference specific **on-chain data accounts** (e.g., `PlayerAccount` and `MatchAccount`).
- Frontend interactions should map to **API calls or RPC endpoints** that trigger program instructions.

4. **Off-chain / Rollup layer missing:**
 - While MagicBlock is listed, no description exists of how ephemeral rollups sync final state back to Solana. A PoC should include at least a summary of this bridging or commit mechanism.
5. **Security and verification coverage:**
 - Add explicit mention of VRF proof verification and match-integrity checks on-chain before rewards are released.
 - Include anti-cheat or timeout mechanisms (e.g., auto-resolve after inactivity).
6. **Developer observability:**
 - Admin tools should log VRF outputs, match IDs, and results to IPFS or a dashboard for transparency and debugging.

In summary: the stories “hit the mark” conceptually but need finer atomic breakdowns and clearer mapping to on-chain programs, accounts, and RPC endpoints to be implementation-ready.

My Analysis:

The AI feedback is correct and the user stories focus on the right things, but I didn't go deep enough into the technical details.

For a PoC, I need to ensure each user interaction translates into clear system components I can actually build or mock.

That means identifying:

- **Frontend interaction flows** (joining matches, UI for fairness proofs, result display)
- **Backend coordination** (match creation, state sync, player matchmaking)
- **On-chain validation** (VRF proof request, fair result commit, reward distribution)

Refined User Stories:

1. **Casual Gamer joins and plays a Whot match with another player, with verifiably fair results.**
 - The user opens the Whot game and joins a lobby or invites another player.
 - The game backend requests a VRF proof from MagicBlock or Switchboard to randomize cards fairly.
 - Both players receive their cards via the game UI, synced through a real-time server.
 - The match concludes when one player wins; the result and proof of fairness are logged.
2. **The player receives token rewards automatically after the match ends.**
 - Once a winner is confirmed, the backend signs and submits a Solana transaction distributing rewards using a pre-funded on-chain program.
 - The winner's wallet receives the reward automatically.
 - Transaction details and result summary are displayed in the UI.

Refined Technical Requirements (Granular):

Layer	Requirement	Purpose
-------	-------------	---------

Frontend (Client)	<ul style="list-style-type: none"> UI for joining/creating a match Display VRF fairness proof Wallet connection (Phantom/SolanaConnect) 	<ul style="list-style-type: none"> Enables matchmaking between two players Keeps both players' moves in sync Shows randomization transparency Allows reward distribution
Backend (Server)	<ul style="list-style-type: none"> Match creation + lobby logic VRF proof request and response verification (MB/Switchboard SDK) Game state management Result submission endpoint 	<ul style="list-style-type: none"> Coordinates game sessions Ensures randomness integrity Tracks turns, decks, and results Finalizes and sends result to blockchain
On-chain (Solana)	<ul style="list-style-type: none"> Smart contract to distribute rewards Event logging of match outcomes Ephemeral Rollup simulation(MB) 	<ul style="list-style-type: none"> Handles automatic winner payout Provides immutable record for transparency Tests low-cost, fast transaction batching

Rationale for Refinement:

The new breakdown ensures every story step can be implemented in the PoC:

- It connects user actions to specific components (frontend, backend, and on-chain).
- It clarifies how fairness and rewards are technically verified and distributed.
- It prepares for testing PoC claims: provable fairness (VRF), low-cost execution (MagicBlock), and reward automation (Solana smart contract).

Part C: Granularity & Clear Refinement:

1. Final User Stories (Simplified & Granular)

Story 1: Player joins a game

A player opens the Whot app and joins a game lobby to play against another online player.

Story 2: System starts a fair game

The system starts the match and automatically shuffles and shares cards using a fair random process that both players can verify.

Story 3: Players take turns playing

Each player takes turns placing cards until one player wins the match.

Story 4: System shows the winner

The app shows who won the match and displays proof that the game was fair.

Story 5: Winner receives reward

After the match ends, the system automatically sends a token reward to the winner's connected wallet and displays the transaction result.

2. Refinement Log

Before	After	Rationale for Change
--------	-------	----------------------

Casual Gamer joins and plays a Whot match with another player, with verifiably fair results.	Split into Story 1 (“Player joins a game”), Story 2 (“System starts a fair game”), Story 3 (“Players take turns playing”), and Story 4 (“System shows the winner”).	The original story contained multiple actions. Split for atomicity and clearer implementation mapping.
The player receives rewards automatically after a match ends.	Story 5 (“Winner receives reward”).	Kept single-action focus; simplified language for non-technical readers.
Backend requests VRF proof from Switchboard/MagicBlock for fairness.	Integrated concept into Story 2 as “system starts the match... using a fair random process.”	Removed technical jargon (“VRF”) to make story stakeholder-friendly; still reflects fairness goal.
On-chain reward distribution via Solana program.	Rewritten in Story 5 as “system automatically sends a token reward to the winner’s connected wallet.	De-jargonized “on-chain” and “program” language to plain English for clarity.

Part D: Defining Potential On-Chain Requirements

1. Brainstorming On-Chain Requirements for Each User Story

Story 1: Player joins a game

- Create or reference an on-chain match account that represents a new or existing Whot game.
- Store player public keys (up to 4 players) inside the match account.
- Initialize match state: waiting, active, completed.
- Lock or escrow each player’s stake amount (if applicable) in the match account.
- Verify both players signed and funded the transaction before starting the match.

Story 2: System starts a fair game

- Request a verifiable random number from Switchboard/Magicblock VRF to shuffle cards.
- Store the VRF proof and the resulting seed in the match account for auditability.
- Use the random seed to generate a deterministic deck order (e.g., via on-chain or verified off-chain computation).
- Mark the match as “active” once the fair shuffle proof is confirmed.
- Emit an on-chain event or log containing the VRF result for client verification.

Story 3: Players take turns playing

- Maintain match state transitions (whose turn, last card played, cards remaining).
- Validate each player’s move against Whot rules stored on-chain (e.g., card number/shape match).
- Prevent illegal moves using on-chain checks and signature verification.
- Update match state after every valid move and emit a state-update event.
- End the match automatically when a winning condition is detected.

Story 4: System shows the winner

-
- When the game-ending condition is met, store the winner's public key in the match account.
 - Verify the result deterministically using stored move history or counters.
 - Write a finalized match result (winner, duration, proof ID) to chain.
 - Emit an event "MatchCompleted"

Story 5: Winner receives reward

- Smart contract releases escrowed tokens to the winner's wallet.
- Update match account to mark rewards_distributed = true.
- Record reward amount, token mint address, and transaction ID.
- Emit an on-chain log confirming the payout.
- Optionally close the match account to reclaim rent or clean up state.