LimeLend: Blockchain-Based Game Lending Platform

Smart Contract Report

Nima Najafian Noah Pursell

May 1, 2025

github.com/noahapursell/CS-5833-Eth-Software-Lending-Project

Contents

1 Overview

LimeLend is a decentralized application (dApp) built on Ethereum that allows users to **own**, **lend**, **rent**, and **access** full games using smart contracts. By moving digital game ownership to the blockchain, LimeLend gives players full control over their licenses while enabling new monetization models through trustless rentals and automated profit-sharing with developers.

2 Motivation

Traditional digital game platforms such as Steam and Epic Games limit users' control over the software they purchase. Licenses are bound to centralized platforms, and sharing or reselling is either prohibited or heavily restricted. Moreover, revenue sharing between developers and platforms lacks transparency and often disadvantages small developers.

LimeLend addresses these challenges by enabling users to fully own, rent, and lend games using Ethereum smart contracts—without the need for centralized infrastructure. This opens new opportunities for decentralized game economies, player empowerment, and fair monetization models.

3 Contributions

In this work, we make the following key contributions:

- **Smart Contract Design and Implementation:** We develop `GameRental`, a Solidity smart contract that encapsulates the full game-lending and rental logic on Ethereum—handling game registration, ownership transfers, time-based billing, and automated revenue sharing between developers and owners.
- **Proof-of-Concept Python CLI Client:** To demonstrate real-world usage, we build a simple Python client with a command-line interface. This CLI allows users to connect to the smart contract, publish new games, purchase licenses, start and stop rentals, query access rights, and withdraw balances. Its minimal design shows how developers can easily build richer frontends or integrate with existing game launchers.
- **Open Protocol for Extensibility:** While our Python client serves as a proof-of-concept, the underlying protocol is fully documented and open. Any developer may create their own client—be it a web app, desktop GUI, or game-embedded launcher—as long as it adheres to the on-chain method signatures and event flows defined by `GameRental`.

This modular separation between a trustless on-chain core and an off-chain client enables rapid experimentation and community-driven enhancements, laying the groundwork for a decentralized ecosystem of game lending and rentals.

4 Smart Contract Architecture

Our platform is built around a Solidity smart contract called `GameRental`, which implements a full-stack, tokenless digital rights management (DRM) system for games.

4.1 Major Functionalities

- **Game Registration:** Developers publish games, setting purchase price and rental rates.
- **Ownership:** Users can buy games and become owners who can optionally make their licenses rentable.
- **Rentals:** Renters pay upfront deposits, which are drained over time as they play the game.
- **Payouts:** Developers and owners can withdraw their accumulated earnings directly from the contract.
- **Access Control:** The `canPlay` function lets the game client check whether a rental is active and valid.

4.2 Sample Code Snippet: Time-Based Rent Billing

```
uint256 requiredOwnerFee = (ownerData.ownerRate * elapsed + 59) / 60;
uint256 requiredDevFee = (game.devRate * elapsed + 59) / 60;
```

Listing 1: Excerpt from internal rent collection logic

This billing mechanism ensures precision while avoiding undercharging by rounding up partial minutes.

5 Benefits of Using Ethereum

5.1 Decentralization and Control

By using Ethereum, LimeLend removes the need for third-party platforms like Steam or Epic Games. Game licenses live entirely on-chain, allowing users to trade, lend, or rent without permission. This fosters a truly user-owned digital economy.

5.2 Trustless Revenue Sharing

Smart contracts automatically split rental income between owners and developers based on predetermined rates. No intermediaries are needed, and no manual payouts or accounting are required.

5.3 Transparency and Security

All interactions—purchases, rentals, and withdrawals—are fully auditable via the Ethereum blockchain. No one can alter the rules or withhold funds arbitrarily, unlike centralized systems.

6 Lifecycle & User Flows

This section outlines the core processes of the LimeLend platform—how developers publish games, users buy or rent them, how access is granted, and how earnings are collected. Each flow is accompanied by a diagram that visualizes the smart contract interactions and system logic.

6.1 1. Developer Publishes Game

Developers register their game on the LimeLend smart contract, specifying the purchase price and rental rate per minute. This creates a new game entry on-chain, making it available for purchase or rental.

In addition to registering the game on-chain, the developer must also publish the actual game executable file. This binary is **not stored on the blockchain** due to the inefficiency and cost of storing large files on-chain. Instead, the executable must be hosted elsewhere—either using traditional centralized cloud services (e.g., AWS, IPFS gateways) or decentralized file storage networks like **IPFS** or **Arweave**.

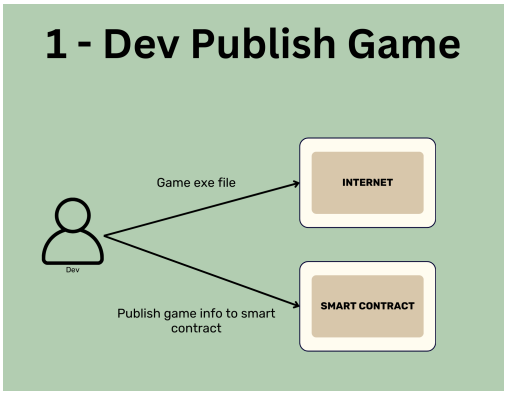

Figure 1: Game Publishing Flow

6.2 2. User Buys Game

Users can purchase a game by sending the required ETH to the smart contract. Once purchased, they become the on-chain owner and gain unlimited access to the game.

After the purchase, users must also **download the corresponding game executable** from the provided online location. Since the executable is not stored on-chain, access to the game relies on this external download step.

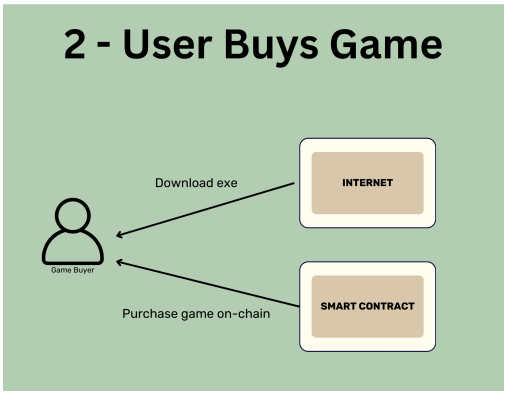

Figure 2: Buying Flow

6.3 3. User Rents Game

Alternatively, users can rent games for a lower upfront cost. Renters deposit ETH into the contract, and the funds are gradually consumed over time.

Importantly, **users are billed based on how long they are registered as renters**, not strictly based on how much time they are actively playing. If a user rents a game but leaves it running in the background or forgets to end the session, charges will still accrue.

To maintain fairness and consistency, the contract enforces an invariant: **only one party can be actively playing a game at any given time**—either the owner or a renter. This prevents overlapping sessions and double-access violations.

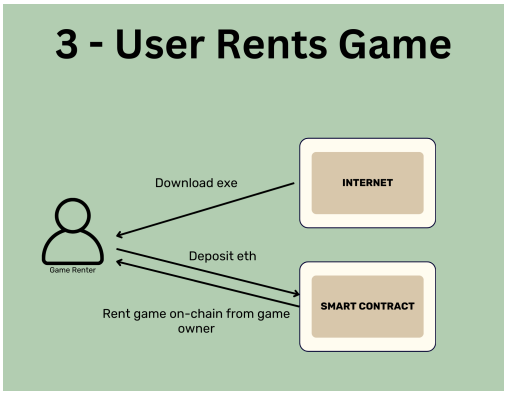

Figure 3: Renting Flow

6.4 4. User Plays Game (Access Check)

To verify access rights, the game executable periodically calls the `canPlay` function on the blockchain. This function checks whether the current user has a valid ownership or rental status and that the rental session has not been terminated.

If a rental has ended—either because the owner called `collectRent` and the user's balance is insufficient, or because the owner manually stopped the session—the `canPlay` check will fail. When that happens, **the game must immediately shut down or lock access**.

To enforce this security measure, the `canPlay` check must be **integrated directly into the game executable itself**. This ensures that users cannot bypass access controls by tampering with an external launcher or CLI tool. Without this embedded logic, unauthorized users could potentially pirate the game by acquiring the raw executable.

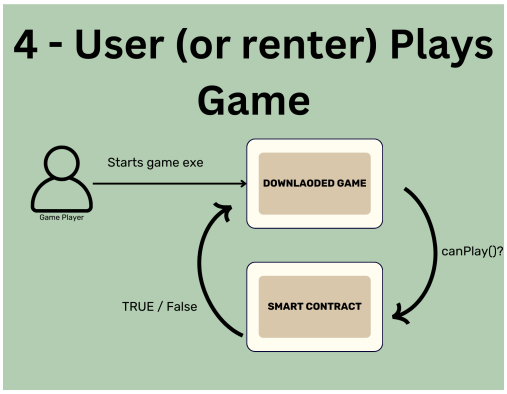

Figure 4: Access Verification During Gameplay

6.5 5. Rent Collection by Owner

Owners of rented licenses must periodically collect rent to receive their share of rental revenue. This is done by calling the `collectRent` function manually or through automated scripts. The smart contract calculates the owed amount based on elapsed time since the start of the rental.

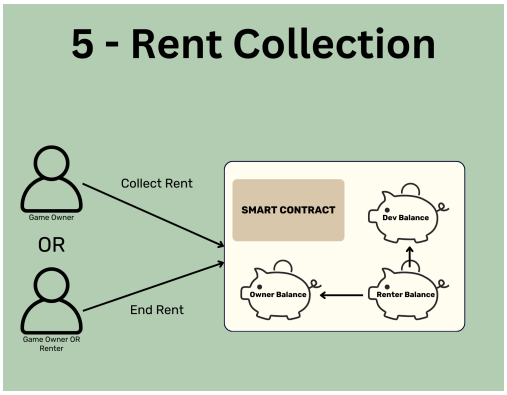

Figure 5: Rent Collection Flow

6.6 6. Withdrawal of Funds

Both game developers and owners can withdraw their earnings from the contract at any time. All balances are transparently tracked and available for withdrawal via standard function calls. Only the rightful address can trigger a withdrawal, ensuring full custody of funds.

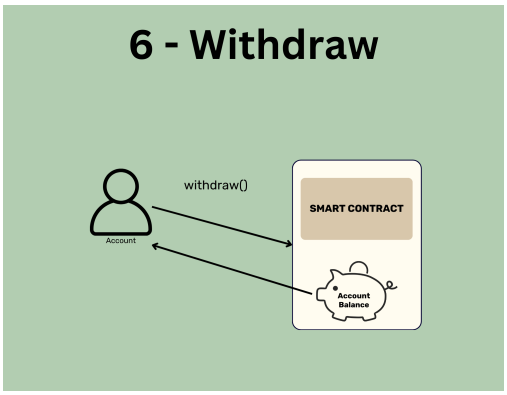

Figure 6: Developer and Owner Withdrawal Flow

7 Design Considerations and Tradeoffs

While decentralizing game ownership and rentals provides many powerful benefits, several key tradeoffs and implementation challenges arise. These influenced our system design and should be carefully considered by future developers and researchers in blockchain-based DRM.

7.1 1. Offline Game Limitation

LimeLend is designed primarily for offline or single-player games. Online multiplayer games, MMOs, or games that require frequent interaction with a centralized server (for matchmaking, state synchronization, or anti-cheat enforcement) fall outside the scope of our decentralized rental model.

While rental access can still be enforced via blockchain, the actual gameplay experience in online titles often hinges on proprietary infrastructure that cannot be decentralized. As a result, LimeLend complements rather than replaces such services and is best suited for downloadable, self-contained game binaries.

7.2 2. Requirement for Internet Connectivity During Gameplay

Even though the games themselves can run offline, users must be connected to the internet at runtime to query the blockchain. This is necessary for calling the `canPlay` function, which determines whether the current user has valid rental or ownership access.

This requirement introduces a tradeoff between decentralization and usability. Offline gameplay is technically feasible, but enforcing access control without an internet connection would require local caching or off-chain verification, which is vulnerable to circumvention.

7.3 3. Embedded DRM Check in Game Executable

To prevent users from bypassing the LimeLend system, we require that the game executable includes a call to the `canPlay` function at regular intervals (e.g., every few seconds or minutes).

This was a deliberate design decision. If access control were enforced purely via the launcher or external scripts, a knowledgeable user could simply bypass the launcher and run the game directly. By embedding the blockchain check inside the executable, the game ensures that unauthorized users cannot play, even if they tamper with the client software.

However, this introduces extra responsibilities for game developers:

- They must modify their game code to include blockchain interaction logic.
- The executable must have network access and Ethereum integration libraries.

These requirements may pose a barrier for indie developers or legacy games, and thus represent a practical limitation.

7.4 4. Deferred Rent Collection and Gas Efficiency

A crucial design tradeoff in our system is how and when rent is collected. Rather than charge users continuously (e.g., in real-time or on every `canPlay` call), we opted for a manual rent collection model. Rent is only collected when:

- The game owner manually calls the `collectRent` function, or
- The renter ends the session using `stopRenting`.

This design provides a significant benefit: it reduces gas fees by avoiding frequent state-changing contract calls. Ethereum transactions are costly, and charging users every minute would be infeasible. By deferring rent collection until user actions or owner intervention, we reduce costs dramatically.

However, this tradeoff introduces two challenges:

1. **Owners must take action to get paid.** They must periodically call `collectRent` or run automated scripts to ensure that their rental earnings are collected.
2. **Renter overuse risk.** If owners do not call `collectRent` in a timely manner, renters could consume more rental time than they have budgeted for. Our system attempts to mitigate this by auto-terminating rental sessions when deposits are insufficient, but real-time enforcement is limited by when rent is collected.

This design places some responsibility on owners, who may choose to use automated tooling (such as our CLI dashboard) to intelligently monitor sessions and trigger rent collection.

7.5 Summary of Tradeoffs

- **Pros:** Lower gas costs, decentralized control, improved user ownership, trustless revenue splits.
- **Cons:** Requires developer integration, online access, owner-side monitoring, and limited to games with no central server dependency.

We believe these tradeoffs are acceptable for many offline titles, especially those developed with modding and experimentation in mind. However, further enhancements—such as integration with Layer 2 solutions or off-chain verifications with zk-proofs—may address these limitations in future versions of the platform.

8 Security Analysis

While LimeLend is designed to enable open access to game executables and minimize blockchain overhead, its architecture also accounts for various attack vectors that might compromise fairness, developer compensation, or access control. This section analyzes several relevant scenarios and demonstrates how LimeLend defends against each.

8.1 1. Sybil Attacks

A Sybil attack involves creating many fake identities (wallets or user accounts) to gain an unfair advantage. In traditional peer-to-peer systems, this can allow attackers to influence consensus, access quotas, or exploit usage-based pricing.

In LimeLend, however, a Sybil attack provides no practical benefit. This is because **only one account can play a given game copy at any given time**. Whether a user controls one account or a thousand, only a single identity can hold the active rental or ownership slot for a game. As a result, creating multiple addresses does not increase a user's ability to play, nor does it allow them to bypass access restrictions.

8.2 2. Playing Without Sufficient Funds

A renter could technically continue playing even after their deposit has run out, but only if the owner has not yet called `collectRent` to enforce payment. This raises the question: can a renter exploit this delay to get more playtime than they paid for?

In practice, this scenario **harms the owner, not the developer**. The developer has already been paid in proportion to their defined rate, and any underpayment affects only the owner who made the game rentable. More importantly, owners are strongly incentivized to collect rent frequently to prevent such overuse.

This behavior is not an exploit or breach of system integrity—it is a byproduct of deferred billing. Since only one party can use the game at a time, and usage is still logged and enforceable, this does not allow users to "cheat" the system, merely to shift risk between owner and renter. Automated tooling can minimize this risk by triggering rent collection at regular intervals.

8.3 3. Executable Piracy Concerns

Since LimeLend does not store game binaries on-chain, but rather they are hosted on the open web, anyone can theoretically download the game executable—even if they haven't purchased or rented the game. This raises an important question: is it safe to expose the game file to the public?

The answer lies in the fact that the **blockchain access check is embedded within the executable itself**. The game calls the `canPlay` function during runtime, and if the user does not have valid on-chain ownership or an active rental, the game immediately terminates or locks access. Tampering with this behavior would require reverse-engineering and altering the binary, which is nontrivial for most users.

In this way, LimeLend mimics DRM approaches where the enforcement logic is built into the game itself, but without relying on centralized servers. As long as developers correctly embed the `canPlay` logic, simply downloading the executable without paying does not enable piracy.

9 Team Collaboration & Contribution

9.1 Role Definition

- **Nima Najafian:**
  - Developed the proof-of-concept Python CLI client using the Typer library, enabling seamless command-line interactions (publish, buy, rent, query, withdraw).
  - Hosted and maintained a local Ethereum test network on his "schmike-server" to facilitate end-to-end integration and testing.

Figure 7: Schmike Server

- **Noah Pursell:**
  - Designed and implemented the `GameRental` smart contract in Solidity using the Hardhat development environment.
  - Authored comprehensive test suites in Solidity/JavaScript to validate contract behavior.
  - Built a Python API wrapper around the deployed contract, allowing the CLI client to invoke on-chain methods programmatically.

9.2 Coordination & Synchronicity

- All work was tracked on GitHub: we opened and assigned Git issues to define tasks, record progress, and flag bugs.
- Weekly check-ins and ad-hoc discussions in issue comments and pull request reviews ensured alignment on design decisions, rapid feedback loops, and shared problem-solving.

9.3 Source Code

The full implementation (smart contracts, tests, and Python CLI client) is available on GitHub:

github.com/noahapursell/CS-5833-Eth-Software-Lending-Project

10 Conclusion

LimeLend demonstrates the potential of Ethereum smart contracts to revolutionize digital ownership and rental models for games. With a smart contract-based backend and CLI-based frontend, it enables players and developers to interact in a trustless and programmable environment. Our architecture balances security, decentralization, and cost efficiency—though with clear tradeoffs that require thoughtful coordination between game developers and platform users.