FSH - Decentralised Digital Pet Care with Conditional Privacy Disclosure

Jake Dalton
University of Canterbury
Christchurch, New Zealand
jda178@uclive.ac.nz

***Abstract*—Tamagotchi-style games allow users to care for virtual pets, but often lack long-term accountability: players can simply start over without consequence after neglecting their pets. Additionally, existing apps struggle to strike a balance between community engagement and user privacy. Fishies Swimmin Hungry, or FSH, addresses these issues by building a decentralised Tamagotchi-like fish tank game on the Secret Network using CosmWasm smart contracts. Players adopt, feed, and care for fish stored privately on-chain. When a fish dies, it becomes a permanent public gravestone, linking the pet to its owner's address. This design incentivises sustained care and provides a novel blend of community interaction and selective privacy. Our approach highlights the unique capabilities of privacy-preserving blockchains to support playful, yet meaningful, accountability mechanisms.**

I. Problem statement and motivation

Tamagotchi-style virtual pet games have long appealed to users by offering emotional connection and routine-based care. However, these games often lack meaningful accountability. Players can neglect their pets, let them "die," and then simply restart with a new one, without consequence or memory. This undermines long-term engagement and strips the experience of any real weight or social meaning.

Beyond this lack of persistence, many virtual pet systems fail to strike a balance between community and privacy. Some operate in total isolation: your pet is yours alone, and your interaction is entirely private. While this preserves user anonymity, it also creates a lonely, single-player experience. Others lean fully into social exposure: everyone's pets are public, and viewable by all. This encourages community building but sacrifices user privacy, which may be undesirable for those who don't want their in-game behaviour tied to their identity.

What's missing is a hybrid approach: one where users can participate in a shared, living world without constant exposure, and where in-game actions carry lasting consequences. Players should feel both a sense of ownership and accountability, knowing that neglecting their pets will leave a permanent, visible trace.

A decentralised solution is well-suited to address this gap. Blockchains offer trustless infrastructure, ensuring that pet data is tamper-proof and survives even if the game's developers step away. By leveraging privacy-preserving smart contracts, such as those offered by Secret Network, we can store user-pet relationships privately, revealing them only when a pet dies. This creates a selective disclosure model, preserving user privacy while introducing meaningful, irreversible consequences.

In this way, decentralisation is not just a technical choice, it becomes a fundamental part of the game's ethical and emotional design. By tying identity to responsibility in a privacy-aware and community-sensitive way, FSH (Fishies Swimmin Hungry) creates a novel and reflective twist on the classic Tamagotchi formula.

II. Related work

Digital pet games and NFT-based creature ecosystems have seen widespread popularity over the last decade, most notably with projects like CryptoKitties or Axie [1] [2]. These platforms tokenize virtual creatures using NFTs, often enabling players to breed, trade, and battle with them. While they offer ownership and financialization of pets, they typically treat pets as assets rather than emotionally significant companions. The focus is on utility and speculation, not responsibility or the ethical implications of neglect.

CryptoKitties, one of the first blockchain games to gain mainstream attention, introduced the concept of unique, collectable on-chain pets. However, the system had no death or failure mechanic; cats could be bred endlessly without consequence. This led to inflationary issues and a lack of long-term accountability, as every asset existed indefinitely.

Axie Infinity, while more complex, followed a similar approach. Though it featured health and battle stats, pets were primarily resources in a play-to-earn economic system. The emphasis remained on value extraction, not care. Pet death was not part of the design space.

Outside of NFT pets, there are systems that explore identity and permanence, like Soulbound Tokens (SBTs), which are non-transferable tokens that aim to represent reputation or achievements [3]. However, these are typically static once issued and not conditional on gameplay or user behaviour.

Our app draws from these prior systems but introduces a new gameplay-experience loop centred on secrecy and consequence. While your fish is alive, it is fully private, tied only to your wallet with encrypted data. Once it dies (through neglect), it becomes publicly visible as a gravestone on the global tank floor. This approach borrows the permanence of SBTs and the uniqueness of NFTs, but adds the emotional and reputational layer of irreversible disclosure. In doing so, it

creates a novel fusion of digital pet care, social accountability, and privacy-aware design.

III. Architecture and Contract Design

The FSH smart contract is written in Rust using CosmWasm and deployed on the Secret Network, which supports private smart contract state. This architecture allows for selective privacy: users can care for their fish in private, but if a fish dies due to neglect, it becomes publicly visible. The contract is structured around two core message types (ExecuteMsg and QueryMsg) and a set of privacy-preserving states that differentiate between public, private, and conditionally revealed data.

A. Diagrams

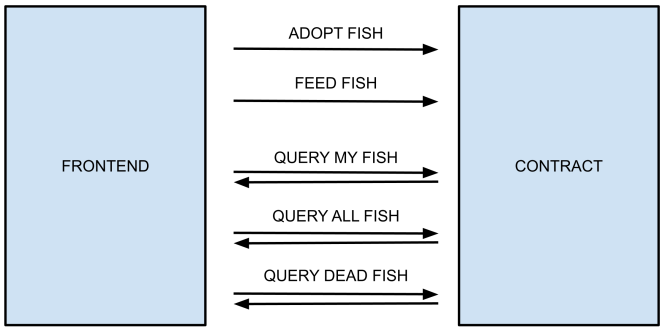

Fig. 1. Diagram showing the different messages and how they are sent between the frontend and contract

Fig. 2. Diagram showing where the different fish status structs are used in the interface. AllFish and DeadFish each have hover effects that show more information

B. Schema and state

The ExecuteMsg enum defines two core user actions: AdoptFish and FeedFish. Adopting a fish assigns it a unique ID and initializes its metadata, including name, owner address, creation timestamp, and a randomly assigned colour. Feeding a fish resets its feeding timestamp, keeping it alive. Each of these actions is gated by permission checks—only the wallet that owns a fish may feed it, and a maximum of five fish per wallet is enforced in the logic.

Querying fish status is handled by the QueryMsg enum, which provides three pathways: FishStatus for retrieving detailed information about a specific wallet's fish; AllFish for accessing a list of all living fish with minimal metadata; and DeadFish for returning all fish that have died, including the public revelation of their owner addresses. The contract responds to these queries with one of three QueryAnswer variants: AllFishStatus, DeadFishStatus, and MyFishStatus, each with increasing levels of detail based on access level and fish life status.

The contract's internal state is managed using Secret Network's privacy-focused storage tools. Fish data is stored in an u64 to Fish keymap where each fish is associated with a unique numeric ID. A separate Keymap maps owner addresses to lists of their fish, enabling efficient lookups. A singleton FISH_COUNTER tracks the next available fish ID to ensure uniqueness. Each Fish struct includes key data such as name, colour, owner, creation and feeding timestamps, and a boolean to track whether the fish is dead. Ownership is stored as a CanonicalAddr, preserving privacy until the fish dies.

C. Special features

One of the contract's defining features is its progressive privacy model. Fish are fully private while alive, and full metadata is only visible to their owner. When queried globally via AllFish, only non-sensitive data (ID, name, and colour) is returned. Once a fish dies, however, its status is moved into a public set, and its owner's address is included in the response to DeadFish. This design creates a delayed transparency mechanism that motivates users to keep their pets alive, not just to maintain a private collection, but to avoid the public association with neglect.

Security and access control are central to the contract's design. Execution functions perform address checks to ensure users cannot feed fish they do not own. Feeding a dead fish is also blocked. Query logic is carefully partitioned to ensure users cannot access sensitive data for fish they do not own, and all metadata not intended for public view is stored using Secret Network's encrypted contract state. Constants like MAX_HUNGER_DURATION (set to 180 seconds for ease of testing) govern how long a fish can go unfed before it is marked dead. Edge cases, such as repeated feeding or attempts to care for more than five fish at a time, are handled gracefully in the contract logic.

In terms of future extensibility, the use of simple, versioned key-based storage (via Item and Keymap) makes the contract straightforward to migrate. The migrate function could be used to update logic or add new features such as breeding, fish trading, or on-chain NFT minting. Because fish IDs are sequential and all ownership links are cleanly tracked, the state can be evolved without complex migrations or data loss.

Overall, the contract architecture emphasises a clean separation of concerns: state integrity, user privacy, and gameplay

accountability. Through careful use of CosmWasm and Secret Network's features, FSH creates a system that is playful, private, and socially meaningful.

IV. Frontend workflow and UX

User interaction with FSH is designed to be smooth, intuitive, and minimally intrusive. Upon visiting the application, wallet connection is handled automatically via Keplr. If a wallet is not detected, the user is prompted with a clear error message in a popup at the bottom right. A wallet icon in the top-right corner of the UI allows users to switch wallets at any time, providing flexibility for multi-account users.

Once connected, the application continuously polls the contract every five seconds to fetch the latest fish data. This keeps the interface updated in real-time without requiring the user to manually refresh or re-query. Each owned fish is represented in a sidebar with clear indicators showing its name, colour, and current hunger level. These visual cues allow users to monitor their fish at a glance and act before any are lost.

Animations enrich the experience: fish swim across the tank at varying speeds, sizes, and opacities to simulate depth and movement, adding to the game's ambience. Transactions, such as adopting or feeding fish, are initiated through intuitive UI buttons. Invalid actions (like feeding a dead fish or adopting more than five fish) are hidden in the interface to avoid confusion and reduce error frequency. When unexpected issues do occur, they are caught and displayed via a non-intrusive error popup in the bottom right corner, ensuring timely but unobtrusive feedback.

V. Implementation choices and trade-offs

A few key design decisions were made to make the game unique and technically sound within the limitations of CosmWasm and the Secret Network. One of the first choices was to assign each fish a random colour using on-chain randomness. This ensures every fish looks unique and creates variety in the fish tank.

Another important decision was how feeding and fish death are handled. In CosmWasm, queries are read-only and can't change the contract's state. That means we can't automatically mark a fish as dead during a query if it hasn't been fed in time. To work around this, the logic for death is included in the FeedFish execute message. If a fish hasn't been fed within the allowed time window, trying to feed it will instead cause it to die. It's a bit unintuitive (fish don't die until someone interacts with them) but this keeps the system consistent and ensures that only execute messages change state.

VI. Future work and roadmap

There are several exciting directions for expanding FSH in future iterations. One planned feature is the ability to breed fish, combining metadata such as colour, speed, and ancestry from parent fish to create a new one. This would introduce a new layer of interaction and long-term planning, while also encouraging users to keep multiple fish alive for strategic breeding opportunities. The breeding logic would be handled entirely on-chain, ensuring fair inheritance and uniqueness.

Another area for growth is adding additional care metrics beyond hunger, like cleanliness, mood, or health. This would turn fish care into a more strategic and engaging system, where players must balance multiple needs to keep their fish alive and healthy. Each stat could decay at different rates or be affected by different actions, introducing new gameplay loops and increasing the emotional attachment to each pet.

References

[1] CryptoKitties, "CryptoKitties — Collect and breed digital cats! — cryptokitties.co," https://www.cryptokitties.co, [Accessed 20-06-2025].

[2] AxieInfinity, "Axie infinity," https://www.axieinfinity.com, [Accessed 20-06-2025].

[3] P. Ohlhaver, E. G. Weyl, and V. Buterin, "Decentralized society: Finding web3's soul," *Available at SSRN 4105763*, 2022.

Appendix

A. Source code

The application's source code can be found at https://eng-git.canterbury.ac.nz/jda178/fsh, which includes integration tests, documentation, and deployment scripts.