Game-Channels for scalable ERC-721 based games

Alessandro Buser

March 17, 2018

1 Introduction

This text should be regarded as a blog post written in LaTeX and not a whitepaper, as it proposes a possible solution to the problem of high gas cost for ERC-721 based games, without proving its feasibility.
The solution I propose is based on the idea of state channels, such as Lightning Network and Raiden, and adapting it to turn based games with collectible items. These "game-channels" are created by using IPFS to store the state of the game after each turn, and adding only a hash of this state to a Smart Contract for, if necessary, automatic auditing.
By aligning the incentives right, the players will likely follow the rules of the game.

2 The problem of scalability for gameplay complexity

As up to today the number of video games leveraging the Ethereum Bockchiain is still very small. The first (and best known) is CryptoKitties, in which players can buy, breed and sell digital kitties. These kitties are the first implementation of the ERC-721 non-fungible-token standard, this new token format comes with the characteristic of non-fungiability, meaning that each individual kitty is unique and distinguishable from the others, since two kitties can be bred to make new kitties which inherit some of the attributes from the parents, players of the game have a big impact on what kind of kitties are in circulation. As ERC-721 tokens are transferable the kitties can also be traded between players creating a marketplace for kitties with certain features.

The problem is that other than collecting the kitties and breeding new ones they don't have any use in the game. So, once you bought a kitty, you have two options: either you hold it and are happy to have a cute digital pet to which only your private key has access, or you breed new kitties to add to your collection. Clearly this are not really engaging game mechanics.
Soon some people (myself including) had the idea to go a step further and build a multiplayer game in which players can actively evolve the attributes of these collectible digital items. The first game that came out having such mechanics, in combination with ERC-721, is Etherbots: here you collect robot parts, of which there are around 32 different ones, each part exists in three rarity classes and is part of one of the four elements(fire, water, steel electricity). A robot can be build out of four different kinds of parts: body, melee weapon, range weapon, and shield. The battles works with a 2-part-commit-reveal scheme(2PCR), where the player can decide to use his robot as a defender or an attacker. If set up as a defender the player commits his robots information together with a sequence of five moves hashed together with his address. As an attacker you first choose one of the robots set out as defenders (without knowing the moves that where committed) and than attack it by also committing five moves. When both sides have moves committed the defender has to recommit his moves (to prove that they are the same he committed before) than the two sets of moves get compared in a rock-paper-scissors fashion to decide how much damage each robot takes, this process also takes into consideration bonuses given by combining same element parts, experience of the part and rarity class of the parts. There is also a perk system which allows the player to specialize on e certain move and element type.
The complexity of this gameplay comes mostly from combining different parts into robots and than guessing enemy moves(if you are the attacker) based on how he combined his parts and the perks he choose. The reason why the gameplay is still mostly based around collecting is that there is no way of having complex and cheap to play game mechanics build on Ethereum Smart Contracts. Since the gas cost scales together with the amount of computations and storage required to run a game it is unfeasible to do something more complex than simple move comparisons without having the players to pay immense amounts of gas.

3 IPFS, Game-Channels and turn based games

One of the scalability solutions gaining traction for payments is off-chain State Channels (Lightning Network for Bitcoin, Raiden for Ethereum), but these are solutions for payments and token transfers, not for computations and data storage. The principle of using the Blockchain only to anchor an off-chain solution is still relevant and can be extended to non-payment problems.
For turn based games a "Game-Channel" can be created by using IPFS for off-chain storage: after each turn the state of the game, containing all the relevant information about that turn, is stored on IPFS. The hash pointing to the state data is then added to the an Ethereum Contract containing the information about the two players and the tokens used in this battle.
The log of game-state hashes saved on the Blockchain allows the players to ask the contract to perform an audit of the last two states. To do this the smart contract retrieves the states from IPFS through an oracle, than he can compare the two states and see if there where moves that don't follow the rules of the game, if such illegal moves are found the contract can execute a punishment on the player who submitted the illegal state, for example by transferring a collateral, submitted when opening the game-channel, from the cheating player to the one reporting him.
To avoid punishing an innocent player, one should only allow the two players who entered the battle to submit moves, and an oracle with verifiable authenticity proves needs to be used by the smart contract when retrieving the states from IPFS.
To further decrease the chance of illegal moves going unnoticed one could allow external users(bounty hunters) to follow other players games and check their moves, if they find players misbehaving they get compensated with their collateral. In "high-end" games such as tournament finals, the organizers can choose to check the state after every move for 100% certainty. Most of the Dapps today are build in the browser and are interfaced with smart contracts through web3. For a game one could have the website not allowing players to submit illegal moves, this would not hinder a malicious player from submitting a faulty state directly to the contract, but one would also have the website check all the states the opponent submits, and notify the player when some illegal moves have been submitted, when this happens the player can then make use of the automatic auditing function in the contract.
In such an environment, where cheating is disincentivised and easily discovered and reporting cheating players is incentivised, one can assume that most of the users follow the rules of the game.

4 Combining Game-channels with ERC-721 non-fungible-tokens

Combining the solution described above with the collectible nature of ERC-721 tokens opens many new possibilities to game developers, as one can now have items bound to players addresses which can be evolved by playing with them in complex turn based matches, one can even have the same tokens being used in multiple games with different mechanics.
One example of such a game would be a turn based RPG or Strategy game (like the older Final Fantasy, Pokemon, and many others) in such a game one would have the characters used in the battles as ERC-721 tokens each with a unique ID and some game relevant attributes. Through battling the attributes of the tokens used are modified (enough to make up for the cost of the battle) based on how good they performed, after some number of battles a player can then choose to unlock some new ability for his token making it more specialized. As these tokens are also tradable, willing players can buy tokens from other players who put the effort, time and gas into evolving and leveling them up. Players can therefore personalize the set of tokens they own, to meet their preferred gamestyle, by playing battles or by buying specific tokens from some marketplace. To keep up the competitiveness of the game scoreboards and tournaments can be organized.
There are two main reasons why a user would prefer playing an ERC-721 based game over more traditional online games. The first is that in most games where players can buy in-game items for real money, they don't actually gain full ownership over these items as there is no in-build proof of ownership and there is often no way to sell them again once the player gets bored with the game. The second thing is that some games have a "pay-to-win" model where non paying players have a disadvantage against paying ones, in a p2p game the players selling the tokens can decide the price needed to gain this advantage, and the money raised by giving players a boost flows to other players and not to external entities.

5 Tradeoffs

As in any design task there are tradeoffs to be made. The biggest upside of using game-channels is that the cost of making a battle only depends on the number of turns, and not on how complex the game is, this is due to the fact that the execution cost of adding a state-hash is always the same, independently from how big the state itself is, the cost of this operation is:

66820 gas if the hash is added as a string, and 20485 gas if its added as a bytes32 (by trimming "Qm" away from the beginning of the hash), at an average gas price of 2Gwei this is around 0.025$. This is without taking into consideration the transaction cost of sending, which is influenced by many factors, one of which is the size of the contract. There is absolutely no way of making a complex on-chain game that cheap. The cost of checking the states for illegal moves depends on the specifics of the game and the way the state is encoded. Since the action of checking for illegal moves is used only if one player suspects that the opponent is cheating this wont happen very often, as there is a very low chance for a cheating player to get away with it, and he probably knows that.
The downside is that users need to trust a third party oracle, which if colluding with the opponent, might submit a wrong state to the contract, tricking it into believing an illegal move was made, resulting in the collateral of an innocent player being lost.
To limit the power of the oracle to submit a different state and get away with it, the player about to be punished can check the hash of this state against the hash of the state he submitted initially. Another way is to give the accused player the chance to submit the authenticity proof, given by the oracle, to a group of independent judges(other players) so that they can verify the validity of this proof. Both of this actions are only required if a knowingly innocent player gets accused by another player colluding with the oracle, a highly unlikely scenario, as the gains of the oracle are minuscule compared to the revenue it makes by offering its service honestly. Another problem with third party oracles is that the functioning of the game depends on the availability of the oracle when it is needed, any interruption off service will result in an inability for players to enter battles.

6 Incentives and implementation details

For any ecosystem build around tradable tokens incentives are a crucial part. In the case of a games build on game-channels a collateral is needed upon entering a battle, one could use Ether or the ERC721 token itself. By using Ether the cost of entry is increased making the game less accessible to new players. By using the tokens used for playing as collateral two players might have a different stake in the game, and games in which more "high-end" tokens are used have a higher collateral, making it less likely players want to face the risk of loosing them by cheating. This would make the staked collateral grow together with the time spent on playing the game, or the

Ether spent to buy the tokens, therefore I regard this option as the one incentivising honest behavior more.
All of the currently existing ERC721 games are run by interfacing Ethereum Smart Contracts with Web3 through the browser, this allows player to only need MetaMask to play the game, all tokens are bound to the address of the MetaMask wallet used by the player. To use game-channels a player also needs to connect to IPFS, this requires a node, there are two IPFS implementations that can be used: the Go version of IPFS can be run locally and the browser can connect to it through the js-ipfs-api. Or one can use the JavaScript version of IPFS with which a node is created directly in the browser.
At the moment the most convenient oracle service to be used by the Smart Contract to retrieve data from IPFS is Oraclize, as they provide a library containing a one-line query for IPFS data, this service also provides verifiable Authenticity Proves, the drawback is that each query costs 0.01$, but since this service is only needed in case of a disagreement the fee wont make a notable difference.
Once the Mist browser is fully functional neither MetaMask nor Oraclize are needed and games can be run without the need for third-party services.
A Proof-of-Concept implementation of Game-Channels was build by myself and can be found at: https://github.com/AleBuser/TicTacToe-ipfs.