Foundations of Blockchain, Cryptocurrencies, and Smart Contracts

Exam Paper:

Building efficient token auctioning systems for video game items using smart contracts on Ethereum

04.01.2021 | Character count: 34111 | Page count: 15

M.Sc. in Business Administration and E-Business

Maxwell Burda - 133903

Course coordinator:

Raghava Rao Mukkamala
ASSOCIATE PROFESSOR,

Department of Digitalization

Room: HOW/60-2.01d

Tel:+4541852299

E-mail: rrm.digi@cbs.dk

TABLE OF CONTENTS

Introduction	**2**
Case Study Description	**3**
Methodology and Concepts	**6**
Conceptual Design	**9**
Discussion	**12**
Conclusion	**15**
References	**16**
Appendix A - Solidity code for ERC1155 auction smart contract	**19**
Appendix B - Linked list data structure used to track bids	**20**
Appendix C - Conceptual Model: ERC1155 smart contract auction	**21**
Appendix D - Conceptual Model: Step 1 - Contract creation	**22**
Appendix E - Conceptual Model: Step 2 - Landing page publication	**23**
Appendix F - Conceptual Model: Step 3 - Auction starts	**24**
Appendix G - Conceptual Model: Step 4 - Auction ends	**25**

Introduction

In recent years, a frenzy for virtual game items has allowed game developers such as Epic Games and Tencent to offer video games for free to the masses, while making money off of in-game purchases (Donnan, 2020). Games such as League of Legends or Fortnite have sold several hundred million USD in virtual merchandise in 2019 (Donnan, 2020). An emerging trend in virtual item ownership is the tokenization of such items on the blockchain. The popular racing game Formula 1, for instance, issued blockchain based tokens on Ethereum that each represent a unique race car in its game (Peng, 2020). One possible way of distributing such novel tokens is through an auctioning process. Auctions have been a trusted way of selling high value, unique goods for their fair market value (Fine, 2020). However, traditional online auctions suffer from several issues, which Pop et al. (2020) argue can be solved by using smart contracts to enforce the bidding process. Pop et al. (2020) developed a framework for the auctioning of ERC721 tokens on Ethereum. ERC721 tokens are non-fungible tokens on the Ethereum blockchain, which identify one unique physical or virtual good (OpenZeppelin, n. d.). An alternative to the ERC721 token standard is the ERC1155 standard, which promises the inexpensive declaration of several items within a contract, which can be non-fungible or fungible (OpenZeppelin, n. d.). This quality of the ERC1155 tokens makes auctioning them more complicated than ERC721 tokens. Therefore, this paper proposes a design for an ERC1155 auction contract, in order to explore the research question:

How can smart contracts be used to simultaneously auction fungible and non-fungible tokens for blockchain based games?

The answer to this research question can help video game projects building on Ethereum reduce the costs of auctioning their in-game items. To investigate this question, the state of blockchain gaming, the benefits of smart contract based auctions, and the ideal type of blockchain for an auction are investigated. Thereafter, an introduction to Ethereum is provided and the paper's methodological choices are presented. Following this, the conceptual contract design is presented, its requirements described, and its interactions identified. The authors relate the findings of the analysis to the academic literature and discuss the limitations of the design. Finally, the paper concludes with key learnings for researchers and practitioners.

Case Study Description

The rising trend of virtual item tokenization

Video games have featured wearable in-game items for characters for many years, but only in the last decade did large game publishers decide to integrate micro-transactions into their games, in order to sell items for income (Tomic, 2018). Micro-transactions allow game publishers to finance themselves by giving players the option to buy special in-game items for real currency, in order to show them off to competitors or friends (Park, 2020). Entire marketplaces for trading in-game items have been created, which have accrued several thousands of Euros in commissions to their owners (Kharif, 2020). The latest emerging trend in gaming has been the use of blockchains to host tokenized forms of in-game items (Scholten et al., 2019). Instead of maintaining a centralized database for in-game items and the quantities owned by each player, these new game developers public smart contracts to a public blockchain that store the attributes and ownership information of virtual goods for them (Scholten et al., 2019). The sale of newly minted tokenized items can then be used as a method to finance the video game's production before its release. Subsequently, the tokens can be traded among players on the open market and used in the games upon release (Scholten et al., 2019). Examples of games that have used this approach to raise funds are Decentraland, which sold virtual real estate to players, in order to raise funding (Jha, 2018).

The equitable distribution of a new game's tokenized items to its community members is a problem for game creators (Parker, 2017). An unfair distribution of game assets will demotivate the userbase to continue playing the game. One popular way of determining the fair price for unique items being sold to the public is to hold an auction (Fine, 2020). Several different types of auctions exist, however the classic English auction has prevailed as the most popular for selling antiques, art, and other collector items (Fine, 2020). In an English auction, a low initial price is set by the auction house and bidders compete to offer incrementally higher prices for the item, until no one bids anymore, and the highest bidder receives the item (Pop et al., 2020). The English auction promises a high price for the seller of the item, in comparison to other auction models, since only the highest bidder receives the item. However, online auctions suffer from several issues, including invalid bids by users, non-delivery by the host, or cancellation by the platform (Braghin et al., 2020).

Challenges solved by blockchain

The blockchain can be used to solve several of the issues faced by regular online auctions. By streamlining the interactions among the parties of the auction with the use of a smart contract, the bidding process can be made more transparent (Pop et al., 2020). In general terms, a smart contract is a piece of publicly hosted code on the blockchain that self-enforces the terms of a contract by executing computerized transactions without intermediaries (Szabo, 1994). By using a blockchain, all bids are public, the participants are identifiable, and transactions are non-repudiable, which prevents malicious actors from posting bids they don't intend to honor or anonymously trying to bid up the price (Braghin et al., 2020). The code within the smart contract that directs the auctioning process is publicly auditable, allowing every prospective participant in the auction to verify its rules. These rules also cannot be changed after initial publication of the contract and the winning bid can be easily verified by searching the contract in a blockchain explorer (OpenZeppelin, n. d.). Most importantly, if a decentralized public blockchain is used to host the smart contract, the participants in the auction process don't have to rely on a central platform (Braghin et al., 2020). A central platform can prematurely cancel the auction, in the case of an unfavorable outcome, or subversively manipulate the bidding process (Scholten et al., 2019). A smart contract will blindly execute the instructions it is provided and will not take a commission for the auction (unless maliciously specified by the host) (Braghin et al., 2020). However, a processing fee will be charged by miners for running a transaction on the blockchain (OpenZeppelin, n. d.).

The scope of decentralization

In order to accrue the previously mentioned benefits of hosting virtual auctions on a blockchain using smart contracts, the underlying blockchain network must be a public, peer-to-peer network, have a robust, decentralized consensus mechanism, must be immutable, and immune to manipulation attempts (Narayanan et al., 2016). A public blockchain is one where all participant accounts, their account values, and the transactions that they broadcast to the network can be viewed by anyone in the blockchain's transaction history (Narayanan et al., 2016). The participants are identified by a public address, which allows for pseudonymous interactions. This allows for the aforementioned identification of all auction participants and the tracking of bids.

The blockchain network must be distributed and peer-to-peer in nature, in order to prevent a single point of auction failure (Braghin et al., 2020). The peer-to-peer nature allows for a blockchain's contents to be hosted redundantly by its participants. Moreover, its integrity can be confirmed by propagating new transactions to other network participants, who confirm the validity of the transaction based on a joint consensus protocol (Narayanan et al., 2016). In the Bitcoin network, this protocol stops double spends and invalid transactions from entering the blockchain. Transactions are confirmed by competing to solve cryptographic hash puzzles, which adjusts in difficulty, in order to maintain a fixed amount of time between transaction publications (Narayanan et al., 2016). The difficult computation process randomizes the selection process that determines who gets to publish new transactions and deliver a proof of work to the network. This makes malicious publications of transactions more difficult (Narayanan et al., 2016). A malicious behavior in an auction setting could be the inclusion of certain auction bids and not others.

Tampering with historical transactions is avoided in a proof-of-work blockchain by publishing sets of transactions to the blockchain network as a Merkle tree hash alongside with the cryptographic hash of the previous block of transactions. This forms a chain of transaction blocks that back reference each other and cannot be changed without the other network participant's noticing (Narayanan et al., 2016). A modified block produces a different hash value, which invalidates all blocks thereafter, since these reference the original hash and not the modified one (Narayanan et al., 2016). This ensures that modification of a historic auction transaction entails unrealistic computational expenditures, since valid hashes for all following blocks would need to be found faster than the rest of the network. A blockchain network that fulfills all these criteria is Ethereum, which is simultaneously the most popular blockchain for hosting tokenized game assets (O'Neal, 2020; Scholten et al., 2019; Ethereum Foundation, 2020).

Methodology and Concepts

Before delving into the methods used to build a conceptual design for an auction contract for video game tokens, this paper reviews the Ethereum blockchain and how smart contracts are executed upon it.

The Ethereum blockchain and smart contracts

Ethereum is a distributed peer-to-peer network, which consists of several independent nodes that adhere to an open source protocol for inter-node communication and validation of transactions broadcasted on the blockchain (Ethereum Foundation, 2020). The blockchain is best described as a digital ledger that keeps track of the transaction history between accounts in the Ethereum network, which transact in the native currency Ether. Accounts in Ethereum are identified by a unique 20-byte address and interact with each other by sending transactions to one another that contain an Ether balance and information (Ethereum Foundation, 2020). Each set of transactions within a roughly 15 minutes interval is saved as a block, which contains a timestamp, a Merkle tree hash of all contained transactions (which allows the confirmation of a transaction's inclusion and prevents alteration) and the state changes they produce, the block number, the block difficulty, and the cryptographic hash of the previous block (Braghin, 2020; Ethereum Foundation, 2020).

Ethereum extends the traditional notion of the blockchain by adding a Turing-complete programming language to the system, which is called Solidity and is compiled to bytecode (Ethereum Foundation, 2020). Each account in Ethereum contains a nonce, an Ether balance, the account's contract code, and the account's internal storage. However, there exist two different types of accounts: externally owned accounts and contract accounts (Ethereum Foundation, 2020). Externally owned accounts are used by network participants to transact in Ether, while contract accounts (a.k.a. Smart contracts) host code that is executed upon the account receiving an Ether transaction. Participants within the Ethereum network identify smart contracts by their unique public account address (Ethereum Foundation, 2020). The contract account's code is publicly visible upon publication and is executed by all Ethereum miners, when one of its functions is called by an external account, which sends a transaction to the contract with instructions to execute. Each contract can hold an internal state, which it stores in its storage field (Ethereum Foundation, 2020).

The possibility to perform permanent state changes to the storage field of smart contracts in Ethereum paved the way for the creation of tokens (Ethereum Foundation, 2020). The storing of key/value pairs in the storage field of smart contracts created the possibility of hosting another ledger (in the form of a hashmap) for a subclass of assets within a smart contract (OpenZeppelin, n. d.). A sender's public Etheruem account address could be used as a key and associated with a value of assets held in the contract ledger. The units of this new subclass of asset on Ethereum are called tokens. Tokens can be transferred from one Ethereum account to the other by sending an instruction to the host smart contract, instructing it to change the ownership of the token in the hashmap (OpenZeppelin, n. d.). Over time, standard implementations of smart contracts for tokens were created. Today, the most popular are the ERC20, ERC721, and ERC1155 token standards. When talking about these standards, one must differentiate between fungible and non-fungible tokens (OpenZeppelin, n. d.). Fungible tokens are ownership rights that are equal to one another. For instance, a new currency would want each share to be worth the same amount and have the same rights. On the other hand, non-fungible tokens are meant to be unique and have certain properties associated with them. They can be used to represent unique items, such as a piece of real estate or a rare digital artifact (OpenZeppelin, n. d.).

Different token standards can be used for different digital representation of in-game items (Scholten et al., 2019). The ERC20 standard is used to distribute tokens that are fungible in nature, divisible, and have a fixed amount of total supply, which is determined by the contract publisher. ERC20 tokens are often used to mint new sub-currencies for games on Ethereum, since the tokens can display decimal values (up to 10^{-18}) and therefore be traded in increments. The ERC721 standard is used in instances, where the creator of the contract wants to create a single unique token that is non-fungible and can represent the ownership of a virtual or physical asset. Lastly, the ERC1155 is the newest of the three standards and is a combination of the ERC20 and ERC721 standards. ERC1155 allows for the definition of several classes of assets that each have an individual fixed amount of supply that is minted with the contract's creation (OpenZeppelin, n. d.). This way the host can create several different tokens with their individual supply profiles. However, each token is not divisible, such as those created in the ERC20 standard. In video games, the ERC1155 standard tokens can be displayed as one of many common clothing pieces or, if the total supply is low, as a rare wearable armour.

Methodology for the conceptual design

This paper extends the prior research by Pop et al. (2020), who discussed a framework for deploying smart contracts on Ethereum for the auctioning of ERC721 tokens. This paper extends their research by building a conceptual design and programming the basic functions that would be needed in a smart contract used to auction ERC1155 tokens. The ERC1155 contract allows for more item classes with varying supply profiles, in contrast to the ERC721 tokens, which makes token auctions more complicated (Ethereum Foundation, n.d.). However, ERC1155 tokens allow developers to save gas fees, since they can bundle the capabilities of ERC20 and ERC721 contracts together, which makes the subject of auctioning these kinds of tokens an interesting problem. Instead of creating a new contract, adjustments can be made to the ERC1155 smart contract, which allows for an auction to take place to distribute the initial tokens.

This paper uses Draw.io to construct an interaction model of all relevant parties in a smart contract based auction. The author uses the popular online IDE Remix to write and test the smart contract's Solidity code on the Ethereum testnet. The code was compiled with Solidity version 0.7.0 and can be found in Appendix A. Audited smart contract templates for the ERC1155 smart contract were used, which are provided by the smart contract auditing service OpenZeppelin (n. d.) and were recommended by Chirtoaca, Ellul, & Azzopardi (2020) in their paper about frameworks for smart contract creation. The remaining code was written by the authors themselves, using the Solidity documentation.

Conceptual Design

The ERC1155 smart contract cannot be used in isolation to host a proper game token auction. Several different components interact together, in order to ensure that a fair auction can take place. The stakeholders in the auctioning process are the game developer, the game players, and the Ethereum miners. The developer creates the ERC1155 contract to sell video game items that he has designed. The video game players want to bid for the tokens, in order to receive special in-game items. Lastly, the Ethereum miners earn fees by validating transactions that are sent to the contract and executing its code.

The game developer must fulfil a set of requirements, in order to be able to distribute his tokenized video game assets in an efficient manner. First, the game developer should know how the Ethereum network works, how to send transactions, how to host a node, and how to program in Solidity (Ethereum Foundation, 2020). If he doesn't, he should hire a blockchain engineer to develop and deploy the necessary contracts and systems for him. In order to simplify interacting with the Ethereum blockchain, the game developer can use the free web IDE Remix to write the smart contract's code and deploy it to the Ethereum mainnet or testnet. The game developer must also run an Ethereum node or pay for an API that allows him to interact with one (e.g. hosted by Infura.io), in order to read data from the blockchain and host his smart contract (Chirtoaca et al., 2020). Another requirement is that the game developer allocates a budget towards online marketing that is meant to create interest in the auctioning process.

The conceptual design of the ERC1155 token auction in Appendix B shows the interactions of all parties involved in the auction. The first step in the auction process is the game developer deciding what in-game items he wants to mint via the ERC1155 contract and in what quantities. He then proceeds to declare those items in lines 8 through 34 of the ERC1155 auction contract in Appendix A. The OpenZeppelin ERC1155 contract allows the game developer to specify the name, total supply, and initial bid value for each in-game item in the contract constructor. Moreover, the developer must give a value to the newly added "AUCTION_END_TIME" field (in unix time) and set the constructor parameter called "IERC1155MetadataURI". This constructor parameter allows the game developer to add a hyperlink of each in-game item token specified in the ERC1155 contract. This link can refer to a picture of the item on the game developer's website (OpenZeppelin, n. d.). In this conceptual design for the auction contract, we want the ERC1155 auction contract to be independent from the owner, once it has been published. Therefore, no owner is specified, but the game developer's address is declared in the constructor as the "msg.sender", in order to send the raised funds back to him, once the auction has ended.

This paper's ERC1155 auction contract relies on the expert auditing of OpenZeppelin and extends their work with custom functions to enable an auction of the tokens within the ERC1155 contract to take place. The first extension is the bid() function, which allows players to place one bid for one item by sending an Ether transaction to the smart contract with an id for the bid function to identify the item they are bidding on. The bid() function can only be executed by outside parties before the end time of the auction. The bid() function enforces this restriction using the "require()" function in Solidity. If the auction end time (AUCTION_END_TIME) is smaller than the current block timestamp, then their bid is sent back to the sending address. The bidder incurs the gas fee. Moreover, the contract will throw an error and revert its last actions if the item "id" passed in the transaction is not within the range of valid ids, the message value is below the initial item price set by the auction owner, or if a bid exists for all available items and the new bid is lower than the lowest bid. If none of the requirements are breached, the bid is accepted.

Since an ERC1155 contract can hold several non-fungible tokens of the same class, the bidding for these tokens must be done in a way that allows the auction smart contract to sell all available tokens to bidders. In this conceptual design, a sorted list of all bids is created, which has a length equal to the amount of the specific item being auctioned. If there are less bids than items available, the new bid is appended to a sorted linked list of bids. If there are more bids than items, a for loop starts at the least expensive bid in the sorted linked list of bids and places the new bid into the sorted list, while popping off the lowest. This allows for a quasi English auction to take place within the contract, where the highest bidders win the items (see Appendix C). Bidders, whose bids get overbid, get their money back, but pay the gas fee. This incentivises high bids from the start. The drawback of this method is the fact that if many items are being auctioned, the reordering of the list will be very expensive in terms of gas and might deter bidders.

The second extension to the ERC1155 contract is the distribute() function, which is used to distribute the tokens to the auction winners and the funds raised to the game developer after the auction has ended. The function can be only called after the auction. This requirement is enforced by using the “require()” function in Solidity to throw an error, if the bid() function is called while the auction’s end time (AUCTION_END_TIME) is smaller than the current block timestamp (block.timestamp). To make sure that the function is not called twice, another require() call checks if a bool value has been flipped yet. The distribute() function can be called by the owner of the auction or any other party that is willing to pay the high gas fee to distribute all tokens to the auction winners and send the owner his raised funds (including gas fee). This allows the players to pool their funds and activate the contract execution if the owner refuses to do so himself or is unable to (e.g. after losing private key or his life).

Once all parameters in the contract are set, the auction contract needs to be published to the blockchain by sending its code in a transaction to the memory pool of transactions without a recipient address (Ethereum Foundation, n. d.). The memory pool contains all transactions that haven’t been included in a block yet. Ethereum miners will choose to include the transaction into a block, depending on its fees, after which it will be published on the blockchain. Once the contract is published, the auction starts (See Appendix D). The second step in the auction process involves building a landing page on the game developers website, in order to display the auction contract address on it, as well as the current highest bids for each of the items within the contract (see Appendix E). In order to do so, the developer must run an Ethereum node or connect to one via an API. The node stores and updates the current state of the blockchain, allowing the developer to show the latest validated bids by the video game players, who send Ether to the auction contract with an id for the item they want to bid on. The game developer could make the auction more user friendly and abstract away the complexity of the ERC1155 smart contract by having different landing pages for each type of items being auctioned. Each landing page could show the public smart contract address, item id, a sorted list of validated Ether bids that have been sent to the contract, and a marking that indicates the last bid that will receive an item at auction expiration. Moreover, the prices of the auction bids could be translated to Euros or USD.

The third step involves the game developer marketing the auction to potential or existing players of his game, in order to spur their participation in the auction (see Appendix F). Players see the online advertising by the game developer for the auction and participate in it by sending Ether transactions to the smart contract displayed on the developer's landing page. These transactions cannot be revoked, once they are mined by the Ethereum miners. At the end of the bidding period, the smart contract calls a function to allocate each token defined in the ERC1155 contract to the public Ethereum addresses named in the list of sorted bidders that the auction contract maintained. In this final fourth step, the distribute() function of the auction contract must be called (see Appendix G). The auction is now complete and the ERC1155 contract can live on and manage the token transfers within the contract.

The proposed ERC1155 auction smart contract was successfully deployed on the Ethereum testnet. The deployment showed that 4,284,622 gas was used for the transaction cost and 3,160,874 gas was needed to execute the contract constructor. A bid() function call cost 173,832 gas to execute.

Discussion

The proposed conceptual design satisfied the requirements of an English auction, while allowing the auctioning of ERC1155 tokens with different supply profiles. The general rules for an English auction are that each participant bids on the item in question with increasing bids, until no one else places a bid, and the highest bidder wins the auction (Fine, 2020). This paper's auction smart contract maintains many of the principles of the classic English auction, but changes one: Several items of the same class are bid upon at the same time and the bids are stored in sorted list, where new bidders must outbid the existing bids in the list, in order to be allocated an item at the end of the auction. This change in the functioning of the auction allows for the transparent auctioning of ERC1155 contract tokens to players, while remaining completely decentralized from any intermediary platform.

The advantage of the blockchain in this auction is that bids cannot be revoked, once they are broadcasted to the blockchain. Moreover, all procedures that are taken during the auction cannot be changed once set in the contract and can be verified by inspecting the transaction history in the blockchain during and after the auction (Pop et al., 2020). The auction itself is arguably more fair than an auction on an online platform, since anyone can participate in a pseudonymous fashion, without the potential for discrimination during an identification or account creation process (Pop et al., 2020). Finally, the proposed structure of the smart contract aligns the incentives of the smart contract creator and the bidders taking part in the auction. The smart contract creator must call the contract's withdrawal function, in order to receive his raised Ether, which is simultaneously the function that distributes ERC1155 token rights to the winners of the auction. Game developers would use such a system to prove the transparent and honest auctioning of in-game items, without favoritism towards certain parties. Game players would be in favor of these auctions, since the items have the potential to be very valuable. A fair playing field at the time of auction is important to keep the players motivated to invest their money and time into the game. Moreover, issuing the in-game items to players via an ERC1155 contract is a lot cheaper than having to create several ERC721 for non-fungible and ERC20 contracts for fungible tokens (OpenZeppelin, n.d.).

Taken together, this paper illustrates that the issuing of both fungible and non-fungible tokens for in-game items is possible for video game developers by using a modified ERC1155 contract to auction tokenize items on the Ethereum blockchain. Only two functions need to be added, in order to allow bidding and the resolution of the auction. The auction contract must be displayed on the video game developer's website and advertised heavily, in order to draw the players attention. Game developers must decide for themselves, if the required investments into blockchain engineering talent, advertising, Ethereum fees, and the heightened security risk are worth the benefits of a more equitable token distribution. Future research should investigate whether Ethereum game players find token auctions held via smart contract to be a more attractive model than the game developer just selling the items directly to the players for a fixed price. The results will determine the value of the auction smart contract model.

However, there exist certain limitations to the decentralization of this smart contract auction model. One of the largest risks to the model is that a hacker compromises the game developer's website during the auction process in step 2 (Appendix E). A hacker could then change the auction smart contract's Ethereum address to a fraudulent smart contract's address, which might send any Ether it receives directly to the hacker's wallet. A similar Ether heist happened to an Ethereum project called CoinDach, whose initial coin offering (ICO) was compromised by hackers (Zhao, 2017). Thus, the website of the game developer remains the single point of failure, since it is the only centralized, attackable component. This threat could be greatly reduced by hosting the landing page for the auction via a distributed content storage network (e.g. the Interplanetary File System (IPFS)) (IPFS, 2020). Future research should investigate the benefits of using IPFS to host websites that display smart contract addresses.

Other limitations of the auction contract stem from the Ethereum network itself. In its current state, the Ethereum network can only handle 14 transactions a second on average (Ethereum Foundation, n. d.). This transaction bottleneck stems from the block gas limit, which limits the gas per block to 10,000,000 gas (ETH Gas Station, 2020). For reference, an average transaction costs 21,000 gas and deploying a smart contract can cost more than 200,000 (ETH Gas Station, 2020). Transaction senders complete to be included in the next block by increasing their transaction fee, incentivizing miners to include their transaction (Ethereum Foundation, n. d.). At current gas prices (200 gwei on January 4th 2021) and Ether prices (942.34 USD), deploying this paper's ERC1155 auction smart contract would cost 1.489 ETH or $1,388.35 USD. Moreover, the auditing and testing costs of the smart contract could incur another 7500 USD for the host, if he wants professional firms to ensure its safety (Mulders, 2018). Therefore, the auction process may only be worthwhile if the items being sold are very rare and valuable or if the developer plans to hold several auctions in the future.

Lastly, consensus disagreements might be a low probability edge case, which should not be ignored when hosting auction smart contracts. It can occur that two Ethereum miners propagate different blocks to the Ethereum network members at the same time (Shurov et al., 2019). Both of these miners found the correct answer to the cryptographic hash puzzle and propagated their transaction blocks to different network participants. In this scenario, the blockchain might split into two parallel chains temporarily, where the network participants decide on the "real" chain that resembles reality by attaching their next blocks to it (Shurov et al., 2019). The transactions in the block that is part of the chain that isn't chosen by the miners will return to the memory pool (Shurov et al., 2019). For this paper's contract, this means that if a block is orphaned with previously valid bids in it, the bid that is shown on the developer's website might not be valid, once the next few blocks have been mined. A solution would be to flag those bids, which have less than 2-3 confirmations with a "unconfirmed" sign. This would indicate that these bids have not been confirmed to be on the longest chain yet and would allow auction participants to adjust their bids.

Conclusion

The conceptual model and empirical research performed in the context of this paper show that auction of both fungible and non-fungible tokens can take place via smart contracts on Ethereum. Game developers can add onto existing ERC1155 smart contract token standards, supplied by the smart contract auditing service OpenZeppelin, in order to create an ERC1155 auction contract for their players. The benefits of such a model include the evasion of platform fees, auction transparency, non-repudiation of bids, anonymity, and non-favoritism during execution (Braghin et al., 2020). However, there exist several non-trivial limitations that make the usage of a smart contract only viable under circumstances where the items for auction are of very high value. These limitations include, but are not limited to, the fees that are incurred on the Ethereum network, the auditing costs, the increased security risks, and development complexity (Mulders, 2018; Zhao, 2017; Ethereum Foundation, n.d.). In the future, the proposed design should be further validated by running trial auctions with large item sets. These trials should explore at what item quantities the Ethereum block gas limit is breached when trying to execute ERC1155 auction contracts, since the for loops within the contract's distribute() and bid() functions are very compute intensive.

References

Braghin, C., Cimato, S., Damiani, E., & Baronchelli, M. (2020). Security with Intelligent Computing and Big-data Services (1st ed., pp. 54-64). Axel Springer.

Chirtoaca, D., Ellul, J., & Azzopardi, G. (2020). A Framework for Creating Deployable Smart Contracts for Non-fungible Tokens on the Ethereum Blockchain. In *2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)* (pp. 100-107). Oxford: IEEE. doi:10.1109/DAPPS49028.2020.00012

Ethereum Foundation. (2020). Ethereum Whitepaper. Retrieved 25 December 2020, from https://ethereum.org/en/whitepaper/

Ethereum Foundation. (2020). Deploying smart contracts. Retrieved December 28, 2020, from https://ethereum.org/en/developers/docs/smart-contracts/deploying/

ETH Gas Station. (2020). What's the Maximum Ethereum Block Size? - ETH Gas Station. Retrieved 31 December 2020, from https://ethgasstation.info/blog/ethereum-block-size/

Fine, L. (2020). Auctions. Retrieved 23 December 2020, from https://www.econlib.org/library/Enc/Auctions.html#:~:text=He%20established%20four%20major%20(one,-bid%20(Vickrey)%20auction.

IPFS. (2020). Host a single-page website on IPFS. Retrieved December 30, 2020, from https://docs.ipfs.io/how-to/websites-on-ipfs/single-page-website/

Jha, A. (2018). Decentraland Prepares for Virtual Real Estate Auction. Retrieved 26 December 2020, from https://www.techworm.net/2017/11/decentraland-prepares-virtual-real-estate-auction.html

Kharif, O. (2020). The World's Biggest Video Game Skins Site Raised $41 Million With Crypto Tokens. Retrieved 25 December 2020, from https://www.bloomberg.com/news/articles/2017-11-13/the-world-s-biggest-video-game-skins-site-raised-41-million-with-crypto-tokens

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). Bitcoin and cryptocurrency technologies (1st ed., pp. 23-51). Princeton: Princeton University Press.

Szabo, N. (1994). Smart Contracts. Retrieved December 27, 2020, from https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

O'Neal, S. (2020). Tokenization, Explained. Retrieved 25 December 2020, from https://cointelegraph.com/explained/tokenization-explained

OpenZeppelin. (n. d.). ERC721 - OpenZeppelin Docs. Retrieved 23 December 2020, from https://docs.openzeppelin.com/contracts/3.x/erc721

OpenZeppelin. (n. d.). ERC1155 - OpenZeppelin Docs. Retrieved 23 December 2020, from https://docs.openzeppelin.com/contracts/3.x/erc1155

OpenZeppelin. (n. d.). Tokens - OpenZeppelin Docs. Retrieved 26 December 2020, from https://docs.openzeppelin.com/contracts/3.x/tokens

Park, G. (2020). The Epic Games Store infuriated people. Gamers spent $680 million there anyway. Retrieved 25 December 2020, from https://www.washingtonpost.com/video-games/2020/01/14/epic-games-store-infuriated-people-gamers-spent-680-million-there-anyway/

Parker, L. (2017). Decentraland raises $24 million in 35 seconds, leaving retail investors out in the cold. Retrieved December 26, 2020, from https://bravenewcoin.com/insights/decentraland-raises-24-million-in-35-seconds-leaving-retail-investors-out-in-the-cold

Peng, T. (2020). Formula 1 Open Tokenized Crate Sale on Ethereum Blockchain. Retrieved 23 December 2020, from https://cointelegraph.com/news/formula-1-open-tokenized-crate-sale-on-ethereum-blockchain

Pop, C., Prata, M., Antal, M., Cioara, T., Anghel, I., & Salomie, I. (2020). An Ethereum-based implementation of English, Dutch and First-price sealed-bid auctions. In IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP) (pp. 491-497). Cluj-Napoca: IEEE.

Scholten, O., Hughes, N., Deterding, S., Drachen, A., Walker, J., & Zendle, D. (2019). Ethereum Crypto-Games: Mechanics, Prevalence, and Gambling Similarities. In CHI PLAY '19: Proceedings of the Annual Symposium on Computer-Human Interaction in Play (pp.

379-389). Barcelona: Association for Computing Machinery. Retrieved from https://dl.acm.org/doi/10.1145/3311350.3347178

Shurov A., Malevanniy D., Iakushkin O., Korkhov V. (2019) Blockchain Network Threats: The Case of PoW and Ethereum. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11620. Springer, Cham. https://doi.org/10.1007/978-3-030-24296-1_49

Tomic, N. (2018). Economic Model of Microtransactions in Video Games. Journal Of Economic Science Research, 1(1), 17-24. doi: https://doi.org/ 10.30564/jesr.v1i1.439

Zhao, W. (2017). $7 Million Lost in CoinDash ICO Hack. Retrieved December 30, 2020, from https://www.coindesk.com/7-million-ico-hack-results-coindash-refund-offer

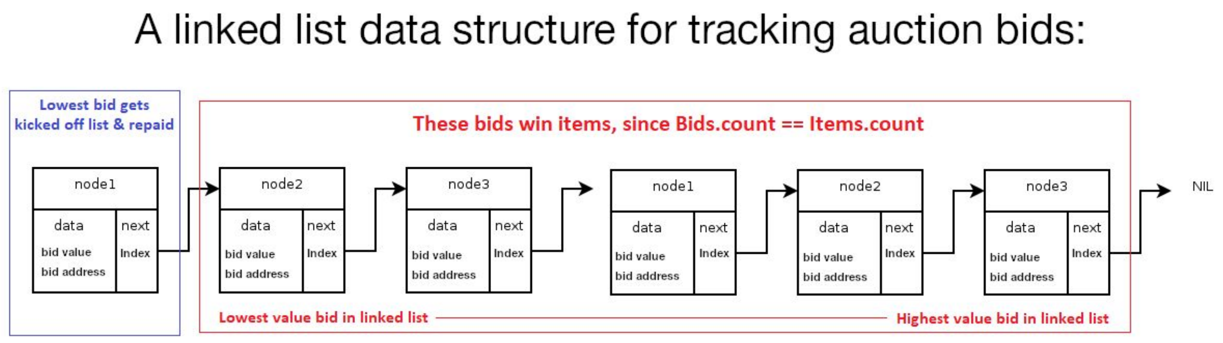
Zhou, Q., Huang, H., Zheng, Z., & Bian, J. (2020). Solutions to Scalability of Blockchain: A Survey. IEEE Access, 8, 16440-16455. doi: 10.1109/access.2020.2967218

Appendix A - Solidity code for ERC1155 auction smart contract

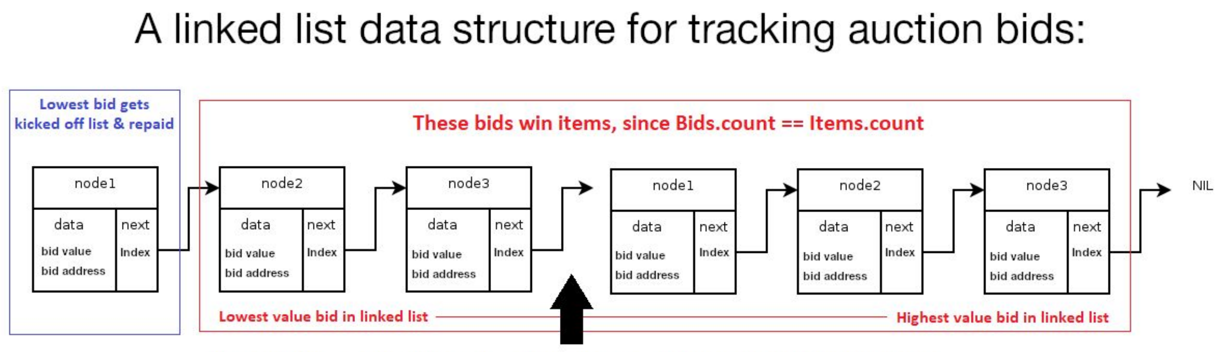
```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import
"https://raw.githubusercontent.com/OpenZeppelin/openzeppelin-contracts/master/contracts/token/ERC1155/ER
C1155.sol"; // ERC1155 standard interface


// This contract issues several classes of spaceships with different rarity

contract GameItems is ERC1155 { // Contract inherits from these classes

    // The game developer can change these values before deploying the contract
    uint256 public constant VESSEL_ID = 0;
    uint256 public constant VESSEL_COUNT = 100;
    uint256 public constant VESSEL_INITIAL_VALUE = 10**17; // Initial value of 0.1 ETH (prices in wei)

    uint256 public constant CRUISER_ID = 1;
    uint256 public constant CRUISER_COUNT = 100;
    uint256 public constant CRUISER_INITIAL_VALUE = 10**17; // Initial value of 0.1 ETH

    uint256 public constant DREADNOUGHT_ID = 2;
    uint256 public constant DREADNOUGHT_COUNT = 50;
    uint256 public constant DREADNOUGHT_INITIAL_VALUE = 5*(10**17); // Initial value of 0.5 ETH

    uint256 public constant BATTLESHIP_ID = 3;
    uint256 public constant BATTLESHIP_COUNT = 5;
    uint256 public constant BATTLESHIP_INITIAL_VALUE = 2*(10**18); // Initial value of 2 ETH

    uint256 public constant DESTROYER_ID = 4;
    uint256 public constant DESTROYER_COUNT = 1;
    uint256 public constant DESTROYER_INITIAL_VALUE = 10**19; // Initial value of 10 ETH

    uint256 public constant AUCTION_END_TIME = 1611230400; /* epoch time (seconds) - Currently set to
01/21/2021 @ 12:00pm (UTC) */

    uint256 public constant MAX_ID = 4;

    /* either we leave the IDs and COUNTs of tokens as constants and create more logic
    (switch statement) to add bids to the correct list or we save these values in a
    structure which adds storage, but removes some bytes from the logic */
    struct BidLink {
        address payable bidder;
        uint256 value;
        uint256 next;
    }

    struct SubAuction {
        uint256 token_count;
        mapping(uint => BidLink) bids;
        uint256 bidcount;
        uint256 head;
        uint256 initial_price;
    }

    address payable private owner;

    bool private wasDistributed;

    SubAuction[5] subauctions;

    constructor() ERC1155("https://game.example/api/item/{id}.json") {
        owner = msg.sender;
        wasDistributed = false;

        _mint(owner, VESSEL_ID, VESSEL_COUNT, "");
        _mint(owner, CRUISER_ID, CRUISER_COUNT, "");
        _mint(owner, DREADNOUGHT_ID, DREADNOUGHT_COUNT, "");
        _mint(owner, BATTLESHIP_ID, BATTLESHIP_COUNT, "");
        _mint(owner, DESTROYER_ID, DESTROYER_COUNT, "");

        /* The struct that stores the bids has a nested mapping of 'bids'. The nested mapping is part of
storage upon declaration and cannot be reassigned in a constructor. Thus, we change each of the values
for each subauctions manually  */

        subauctions[VESSEL_ID].token_count = VESSEL_COUNT;
        subauctions[VESSEL_ID].bidcount = 0;
        subauctions[VESSEL_ID].head = 0;
        subauctions[VESSEL_ID].initial_price = VESSEL_INITIAL_VALUE;

        subauctions[CRUISER_ID].token_count = CRUISER_COUNT;
        subauctions[CRUISER_ID].bidcount = 0;
        subauctions[CRUISER_ID].head = 0;
        subauctions[CRUISER_ID].initial_price = CRUISER_INITIAL_VALUE;

        subauctions[DREADNOUGHT_ID].token_count = DREADNOUGHT_COUNT;
        subauctions[DREADNOUGHT_ID].bidcount = 0;
        subauctions[DREADNOUGHT_ID].head = 0;
        subauctions[DREADNOUGHT_ID].initial_price = DREADNOUGHT_INITIAL_VALUE;

        subauctions[BATTLESHIP_ID].token_count = BATTLESHIP_COUNT;
        subauctions[BATTLESHIP_ID].bidcount = 0;
        subauctions[BATTLESHIP_ID].head = 0;
        subauctions[BATTLESHIP_ID].initial_price = BATTLESHIP_INITIAL_VALUE;

        subauctions[DESTROYER_ID].token_count = DESTROYER_COUNT;
        subauctions[DESTROYER_ID].bidcount = 0;
        subauctions[DESTROYER_ID].head = 0;
        subauctions[DESTROYER_ID].initial_price = DESTROYER_INITIAL_VALUE;

    }

    function distribute() public {
       require(block.timestamp > AUCTION_END_TIME, "Auction has not ended");
       require(wasDistributed == false, "Tokens already distributed");
       wasDistributed = true; /* prevent re-entrancy */


       for (uint256 id = 0; id <= MAX_ID; id++) {
            // Start at head of linked list
            BidLink memory current = subauctions[id].bids[subauctions[id].head];
            for (uint256 i = 0; i < subauctions[id].token_count; i++) {
                // Transfers 1 token per bid to winner
                safeTransferFrom(owner, current.bidder, id, 1, "");
                current = subauctions[id].bids[current.next];
            }
        }
        /* The .transfer() method receives gas from the transaction initiator (2300 gwei) */
        owner.transfer(address(this).balance); // Send contract owner all funds in the smart contract
    }


    function bid(uint256 id) payable public {
        require(id <= MAX_ID, "Id doesn't exist"); // Require that the sent item ID exists
        require(msg.value > subauctions[id].initial_price, "Sent value below initial price");
        // Sent transaction must beat lowest bid to be considered
        /* when max amount of bids were received, the sent transaction must be higher
        than the lowest bid to be considered */
        if (subauctions[id].bidcount >= subauctions[id].token_count
                && msg.value < subauctions[id].bids[subauctions[id].head].value) {
            revert("Sent value below lowest bid");
        }
        /* if auction has already ended, throw error */
        require(block.timestamp < AUCTION_END_TIME, "Auction has ended");




        BidLink memory new_bid = BidLink({bidder:msg.sender, value:msg.value, next:0});
        /* add bids smaller than any other at the end of the list, but only if there is room */
        if ((subauctions[id].bidcount == 0 || msg.value <=
subauctions[id].bids[subauctions[id].head].value)
                && subauctions[id].bidcount < subauctions[id].token_count) {
            new_bid.next = subauctions[id].head;
            subauctions[id].head = subauctions[id].bidcount;
            subauctions[id].bids[subauctions[id].bidcount] = new_bid;
            subauctions[id].bidcount++;
            return;
        }


         /* find the first bid which is higher than the current bid and insert this bid before it if
possible...by adding items in this way our list stays sorted and the head points to the smallest item
(start) */

        /* start at head of linked list (lowest bid of current item id) */
        BidLink memory current_bid = subauctions[id].bids[subauctions[id].head];

        for (uint256 i = 0; i < subauctions[id].bidcount - 1; i++) { /* only for count - 1 since we look
at next value in loop */
            /* comparing to next bid since head value is checked before this */
            if (i == subauctions[id].bidcount - 1
                    || new_bid.value <= subauctions[id].bids[current_bid.next].value) {
                /* to make room for new bid, replace smallest bid (head) and move the head one upwards
*/
                new_bid.next = current_bid.next;
                /* save head for repayment of bid, since bidder gets kicked */
                uint256 oldhead = subauctions[id].head;

                /* for distributing one token, the head remains, pointing to same index (0) */
                if (subauctions[id].token_count > 1) {
                    subauctions[id].head = subauctions[id].bids[subauctions[id].head].next;
                /* set new head to next largest */
                }
                subauctions[id].bids[oldhead] = new_bid;

                // stop re-entrancy vulnerability
                uint amount = subauctions[id].bids[current_bid.next].value - 2300 * tx.gasprice;
                subauctions[id].bids[current_bid.next].value = 0;
                subauctions[id].bids[current_bid.next].bidder.transfer(amount);

                /*  O --> O   to    O --------> O  to  O -        -> O
                       O                 O -´             `-> O -´      */

                current_bid.next = oldhead;

                return;
            }
            else {
                // Set the current bid to the next bid that it links to
                current_bid = subauctions[id].bids[current_bid.next];
            }
        }

    }
}
```

Appendix B - Linked list data structure used to track bids

Data structure:

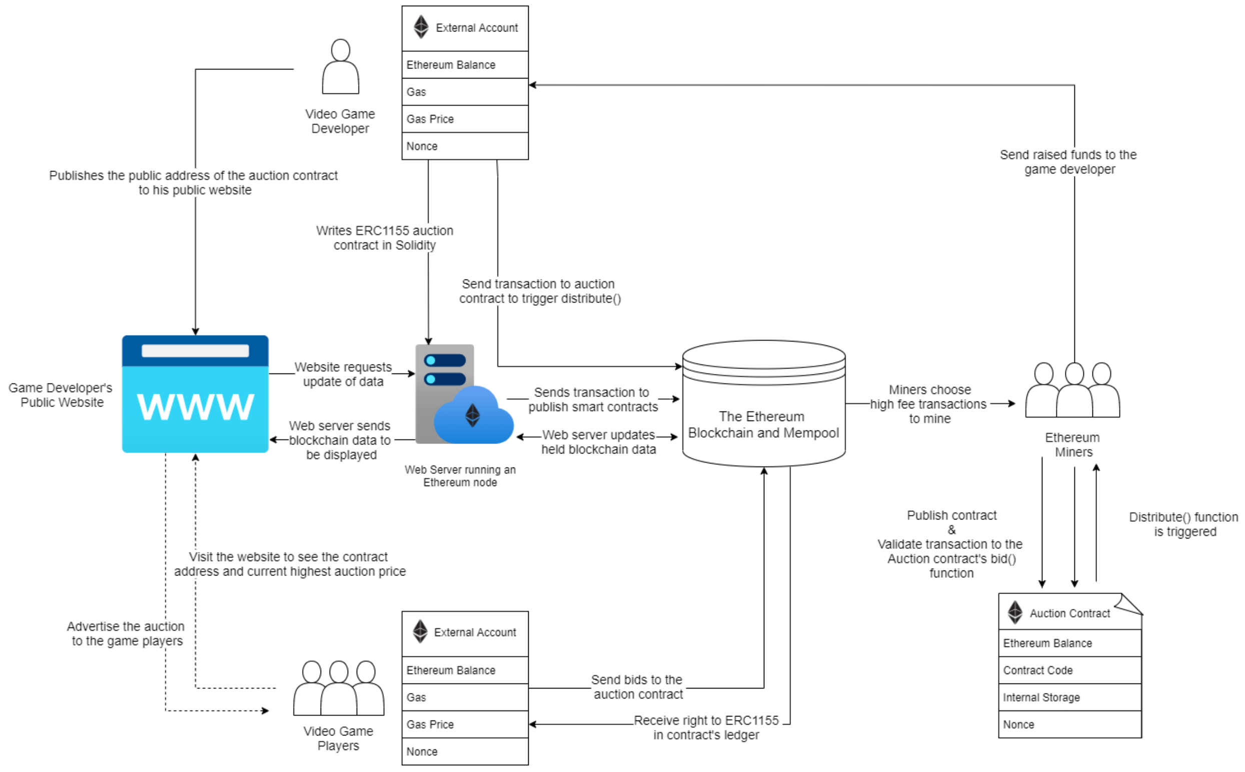

A linked list data structure for tracking auction bids:
Lowest bid gets kicked off list & repaid
These bids win items, since Bids.count == Items.count
node1
node2
node3
node1
node2
node3
data
next
bid value
bid address
Index
NIL
Lowest value bid in linked list
Highest value bid in linked list

New data insertion procedure:


A linked list data structure for tracking auction bids:
Lowest bid gets kicked off list & repaid
These bids win items, since Bids.count == Items.count
node1
node2
node3
node1
node2
node3
data
next
bid value
bid address
Index
NIL
Lowest value bid in linked list
Highest value bid in linked list

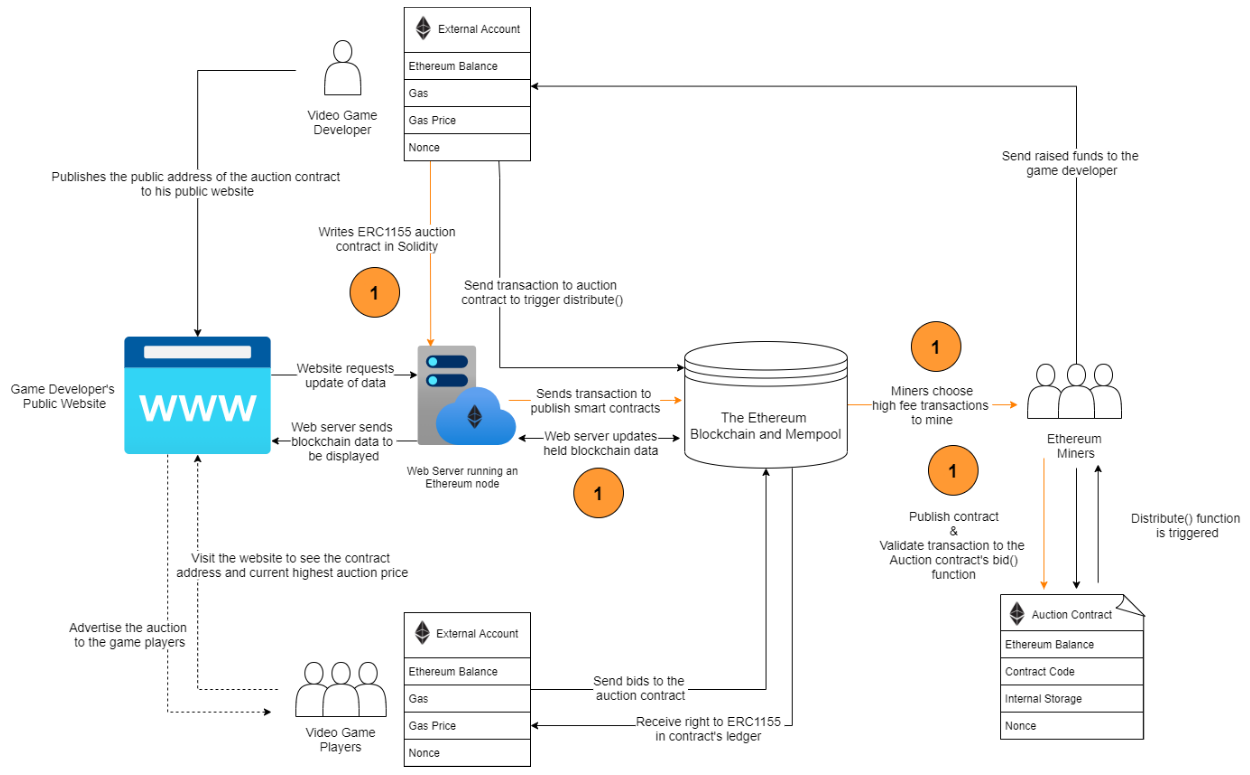
New bids are inserted by pointing the next pointer of the directly lower value bid to the new bid and pointing the next pointer of the new bid to the next highest bid. The lowest bid is kicked off the list afterwards, since there would be more winning bids than items otherwise.

Appendix C - Conceptual Model: ERC1155 smart contract auction

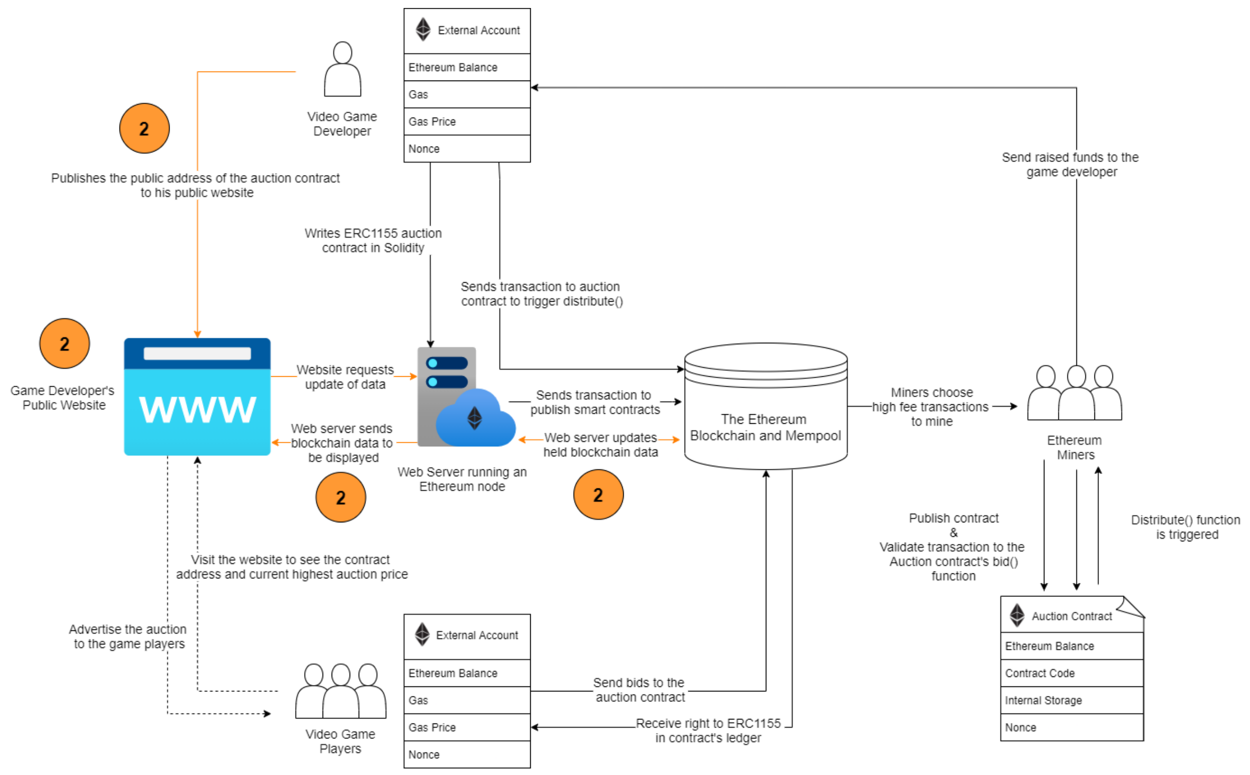

External Account
Ethereum Balance
Gas
Gas Price
Nonce
Video Game Developer
Publishes the public address of the auction contract to his public website
Writes ERC1155 auction contract in Solidity
Send transaction to auction contract to trigger distribute()
Send raised funds to the game developer
Game Developer's Public Website
WWW
Website requests update of data
Web server sends blockchain data to be displayed
Web Server running an Ethereum node
Sends transaction to publish smart contracts
Web server updates held blockchain data
The Ethereum Blockchain and Mempool
Miners choose high fee transactions to mine
Ethereum Miners
Publish contract & Validate transaction to the Auction contract's bid() function
Distribute() function is triggered
Visit the website to see the contract address and current highest auction price
Advertise the auction to the game players
Video Game Players
External Account
Ethereum Balance
Gas
Gas Price
Nonce
Send bids to the auction contract
Receive right to ERC1155 in contract's ledger
Auction Contract
Ethereum Balance
Contract Code
Internal Storage
Nonce

A high definition version of this image can be downloaded at:
https://drive.google.com/file/d/1eeknhWAXcwjCnz6vFf9so_-BSFoiBxaR/view?usp=sharing

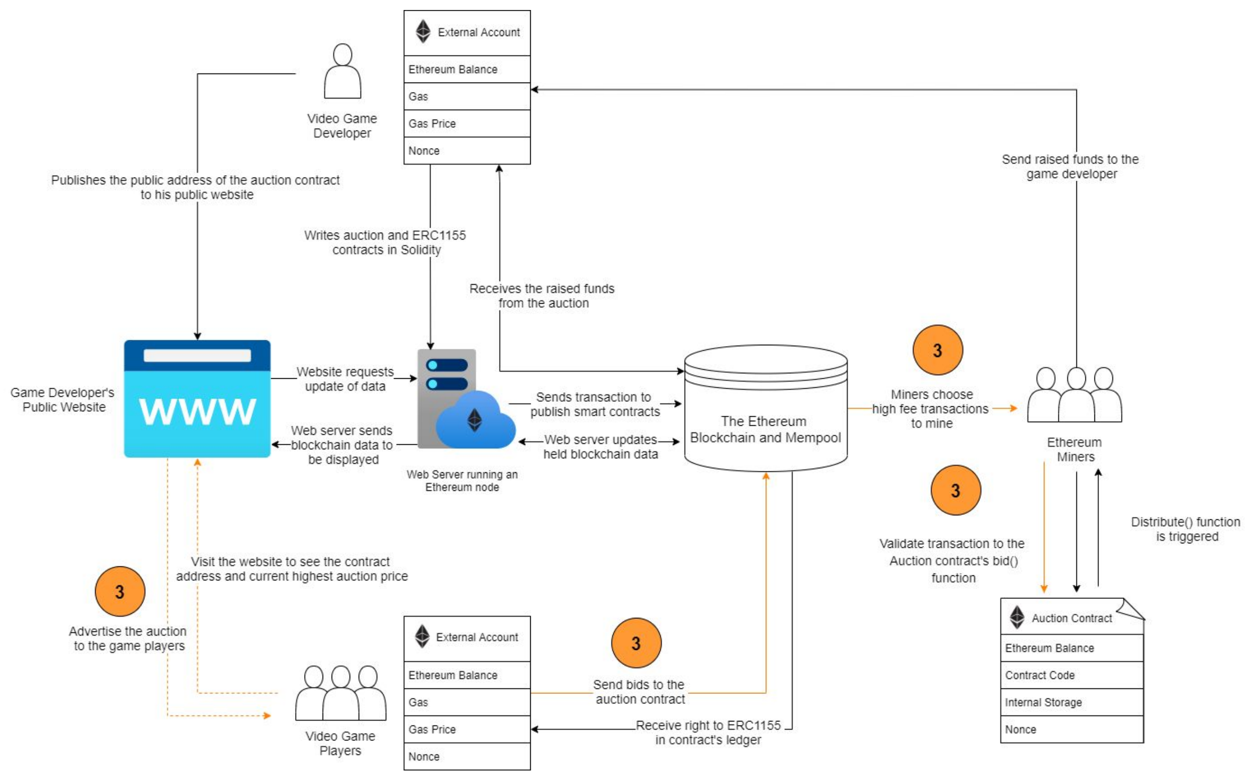
Appendix D - Conceptual Model: Step 1 - Contract creation


External Account
Ethereum Balance
Gas
Gas Price
Nonce
Video Game Developer
Publishes the public address of the auction contract to his public website
Writes ERC1155 auction contract in Solidity
1
Send transaction to auction contract to trigger distribute()
Send raised funds to the game developer
Game Developer's Public Website
WWW
Website requests update of data
Web server sends blockchain data to be displayed
Web Server running an Ethereum node
Sends transaction to publish smart contracts
Web server updates held blockchain data
1
The Ethereum Blockchain and Mempool
1
Miners choose high fee transactions to mine
Ethereum Miners
1
Publish contract & Validate transaction to the Auction contract's bid() function
Distribute() function is triggered
Auction Contract
Ethereum Balance
Contract Code
Internal Storage
Nonce
Visit the website to see the contract address and current highest auction price
Advertise the auction to the game players
Video Game Players
External Account
Ethereum Balance
Gas
Gas Price
Nonce
Send bids to the auction contract
Receive right to ERC1155 in contract's ledger

Appendix E - Conceptual Model: Step 2 - Landing page publication

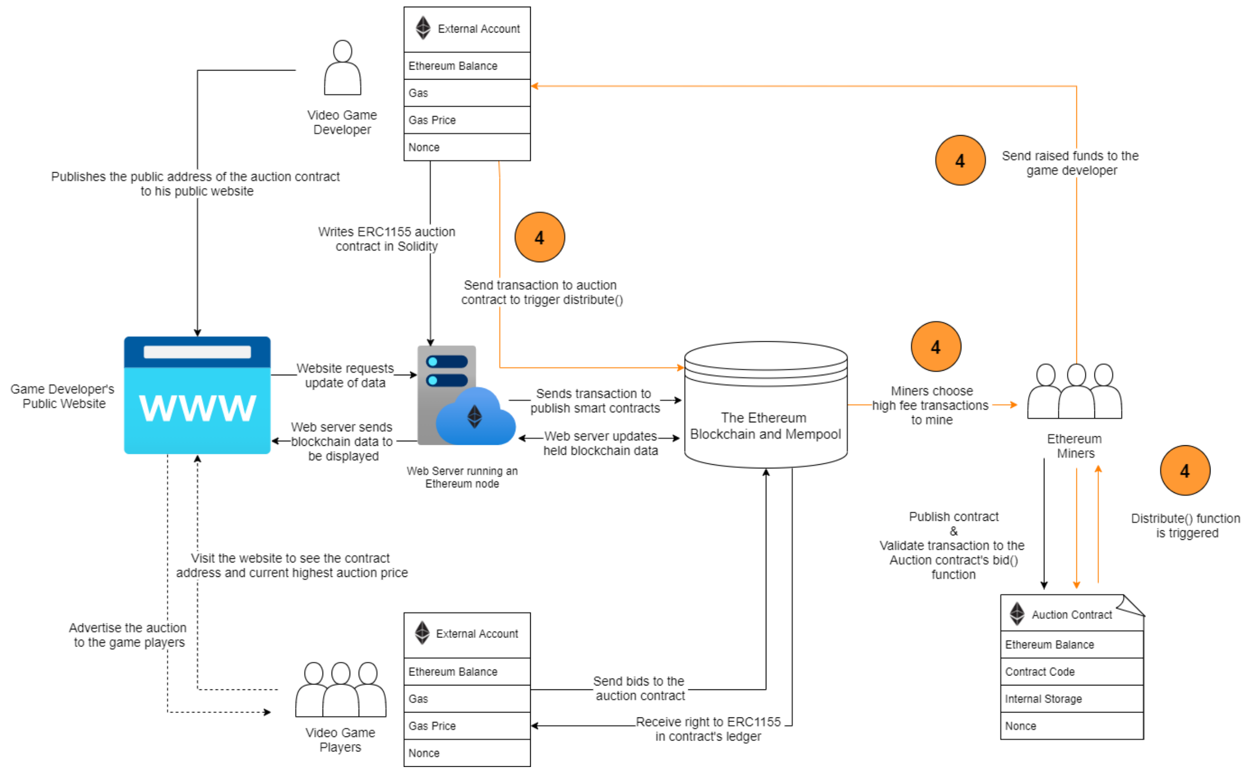

External Account
Ethereum Balance
Gas
Gas Price
Nonce
Video Game Developer
2
Publishes the public address of the auction contract to his public website
Writes ERC1155 auction contract in Solidity
Sends transaction to auction contract to trigger distribute()
Send raised funds to the game developer
2
Game Developer's Public Website
WWW
Website requests update of data
Web server sends blockchain data to be displayed
Sends transaction to publish smart contracts
Web server updates held blockchain data
The Ethereum Blockchain and Mempool
Miners choose high fee transactions to mine
Ethereum Miners
Web Server running an Ethereum node
2
2
Publish contract & Validate transaction to the Auction contract's bid() function
Distribute() function is triggered
Visit the website to see the contract address and current highest auction price
Advertise the auction to the game players
Auction Contract
Ethereum Balance
Contract Code
Internal Storage
Nonce
External Account
Ethereum Balance
Gas
Gas Price
Nonce
Video Game Players
Send bids to the auction contract
Receive right to ERC1155 in contract's ledger

Appendix F - Conceptual Model: Step 3 - Auction starts



External Account
Ethereum Balance
Gas
Gas Price
Nonce
Video Game Developer
Publishes the public address of the auction contract to his public website
Writes auction and ERC1155 contracts in Solidity
Receives the raised funds from the auction
Send raised funds to the game developer
Game Developer's Public Website
WWW
Website requests update of data
Web server sends blockchain data to be displayed
Web Server running an Ethereum node
Sends transaction to publish smart contracts
Web server updates held blockchain data
The Ethereum Blockchain and Mempool
3
Miners choose high fee transactions to mine
Ethereum Miners
3
Validate transaction to the Auction contract's bid() function
Distribute() function is triggered
Auction Contract
Ethereum Balance
Contract Code
Internal Storage
Nonce
3
Advertise the auction to the game players
Visit the website to see the contract address and current highest auction price
Video Game Players
External Account
Ethereum Balance
Gas
Gas Price
Nonce
3
Send bids to the auction contract
Receive right to ERC1155 in contract's ledger

Appendix G - Conceptual Model: Step 4 - Auction ends


External Account
Ethereum Balance
Gas
Gas Price
Nonce
Video Game Developer
Publishes the public address of the auction contract to his public website
Writes ERC1155 auction contract in Solidity
4
Send transaction to auction contract to trigger distribute()
4
Send raised funds to the game developer
Game Developer's Public Website
WWW
Website requests update of data
Web server sends blockchain data to be displayed
Web Server running an Ethereum node
Sends transaction to publish smart contracts
Web server updates held blockchain data
The Ethereum Blockchain and Mempool
4
Miners choose high fee transactions to mine
Ethereum Miners
4
Distribute() function is triggered
Publish contract & Validate transaction to the Auction contract's bid() function
Auction Contract
Ethereum Balance
Contract Code
Internal Storage
Nonce
Visit the website to see the contract address and current highest auction price
Advertise the auction to the game players
Video Game Players
External Account
Ethereum Balance
Gas
Gas Price
Nonce
Send bids to the auction contract
Receive right to ERC1155 in contract's ledger