Design and implementation of NFT-based system

Jiahui Huang[1, *, †] Yuzhuo Shan[2, †] Yining Wang[3, †]

[1]Colloge of information Science and Technology, Beijing University of Chemical Technology, Beijing, China

[2]School of Computer Science and Engineering, Beijing Technology and Business University, Beijing, China

[3]Department of Information, Beijing University of Technology, Beijing, China

*Corresponding author: 2021110087@buct.edu.cn

†These authors contribute equally.

Abstract. Blockchain technology is one of the hottest internet techniques by far, and the NFT is a novel digital collection based on blockchain technology. In 2017, two interesting applications emerged on Ethereum, CryptonPunks, and CryptonKitties, As CryptonKitties is an example, each kitty has a unique DNA number, taking on a different appearance and temperament. These two applications have revolutionary significance for the non-fungible token proposal and practical scenario application. In this experiment, We used the Ethereum platform in the Solidity language to implement a set of digital currency systems based on NFT. It's named CoinCoin.Smart contracts are the way we implement all the functions. Through the smart contract, it can realize its minting, trading, and other functions. The implementation of functions will be simulated on the Remix platform. Mastering NFT's smart contract could be able to provide a new ecosystem for the encryption industry and have a profound impact on the digital asset across multiple industries, such as games, art, and digital assets. Our system can provide a reference for the implementation of the NFT smart contract and bring experiences for the further deeper research.

Keywords: NFT; Smart Contract; Solidity.

1. Introduction

With the rapid application and popularization of blockchain technology, the NFT of digital assets on the blockchain has aroused wide interest from academia and industry[1].2021 is the undisputed "first year of NFT,"[2], With lots of expensive art being sold at auction, new forms of NFTS are coming out. Now, stars, brands, and enterprises have started to get involved in the NFT industry, and the NFT craze has swept the world in a short time[3-6].

NFT means non-fungible tokens, which is a unit of data of blockchain data ledger, each NFT can refer to exclusive digital data. It is based on two standards: Ethereum ERC -721 and ERC -1155. It brings a new expansion of blockchain technology in the field of digital assets and provides ownership with provable uniqueness to the associated digital product. It also has tradable characteristics, liquidity, scarcity, immutability, and other characteristics of digital currency based on blockchain technology, which is why NFT is widely used in art, blockchain games, finance, etc.

Smart contract is a code on the blockchain that can realize the basic functions of NFT. When you deploy the smart contract code to the blockchain, after successful execution, it will store the obtained data on the blockchain module. Its code has the principle that the execution result cannot be changed, and the information is open and transparent. It has functions such as casting, transferring, querying, etc. A blockchain with a good smart contract layer can have a great improvement in its performance.

This experiment will design a set of NFT-based smart contract systems, design and issue an NFT currency "CoinCoin", based on this system [7-8]. The debugging of smart contract writing is implemented on the Ethereum platform. Through this experiment, we preliminarily realized his construction system, feeding system, combat system, trading system, upgrade system, and so on, and defined many properties of our currency, such as ID, DNA, level, and cooldown time.

In the following, we will show the functions and implementation of our system in detail from two aspects: system requirements analysis and the design of the smart contract and the system.

2. System Requirement Analysis

In this part, we are going to have a general introduction to the system construction goals, and give an overview of the system requirements including functional requirements and non-functional requirements [9-10].

2.1. System Construction Goals

This project is a smart contract for a set of issuable and tradable Non-Fungible Token (NFT)-"CoinCoin" written on the Ethereum platform, based on Solidity (5.12 version). The smart contract contains the coin's Genetic System, Ownership System, Attack System, Feeding System, Trading System, Level System, Search System, and other relative functions. The implementation and deployment of its functionality are simulated on the Ethereum platform·

2.2. System Requirements Analysis Overview

2.2.1 Functional Requirements

In the genetic system, each coin has a 16-digit decimal number as its gene code and its last digit as the special envoy logo. Each coin has its own name, and coins are generated according to the name and random number. In the ownership system, everyone can have a coin for free and pay for a coin. In the attack system, it is defined that the owner of one coin can attack the owner of other coins, with a winning rate of 70%. The winning party can obtain another gold coin for its own. In the Feeding system, we stipulate that coins can be fed once a day. And after ten times of feeding, a new coin will be produced for you. In the trading system, we can put our coins into the market for pricing sales and purchases, with taxes and the lowest price. In the level system, the coins can be upgraded by launching an attack and winning, or they can be upgraded by paying extra money as well. In the search system, we can mainly query the name, quantity, grade, and other information of our coins. In addition, there are functions to query the balance and withdraw money. The above is its main function.

2.2.1 Non-functional requirements (security, reliability, ease of use)

This experiment mainly uses a safemath smart contract and ownable smart contract to ensure that the data in the contract does not overflow and whether the coin owner legally owns the coin ownership, so as to prevent noncoin owners from operating on others' gold coins.

3. Design of Smart Contracts and System

In this part, we are going to We will introduce the implementation method of this experimental system in detail. We will start with the standardized contracts and then the smart contracts we wrote. Each module of the system is contained in a different smart contract, we are going to explain how each module implements its own functions in different smart contracts.

3.1. Standardized contract

3.1.1. Smart contract:ERC721

The ERC stands for Ethereum Request for Comment, which represents a protocol proposal submitted by Ethereum developers. The ERC contains technical and organizational considerations and standards. The ERC contains technical and organizational considerations and standards.

3.1.2. Smart Contracts: Ownable and Safemath

An Ownable smart contract is primarily a determination contract that determines whether a function caller is the owner of the contract. The Ownable contract has an owner address and provides

basic authorization control, simplifying the implementation of user permissions through functions. The Ownable smart construct sets the original "owner" of the contract to the sender. Thrown if called by any account other than the owner. It also allows the current owner to transfer control of the contract to the new owner and transfer the address of ownership. The main function of the Safemath smart contract is to perform security checks on mathematics which will be used on the other smart contracts and throw problems when errors are found. The specific function is: two numbers multiply, add, subtract, and throws on overflow. When two numbers are divided, the quotient is truncated and the Solidity automatically throws when dividing by 0.

3.2. Implementation of Smart Contract

3.2.1. An Overview

After introducing the requirements of this NFT smart contract above, the implementation logic of the NFT smart contract is then entered. This experiment is divided into seven smart contracts to realize all the functions of CoinCoin, including CoinFactory, CoinHelper, CoinFeeding, CoinAttack, CoinOwnership, CoinMarket, and Coinore.

3.2.2. CoinFactory

The main function of the CoinFactory contract is to enter the name to create coins, buy coins, and set the price of coins.

The Solidity compiled version of all the smart contracts involved in this experiment is 5.12 and references the ownership smart contract Ownable and the security mathematics smart contract Safemath in the coin factory smart contract. And the coin factory inherits the Owanble contract and references the Safemath contract for uint256 in the contract to ensure that it does not overflow.

There were five main variables in the experiment: dnaDigit, dnaModulus, coolDownTime which was set as 1 days, coinPrice and coinCount which was initially set as 0. There is also a construct that represents the attributes of the coin, which collectively contains the coin's name, gene, level, cooldowntime, number of wins, and losses.

In addition, variables with the same set of the number of digits are defined together to avoid unrelated ones created by the program, so that the program can make full use of internal space and reduce the waste of gas costs. The struct array is defined to hold the contents of the struct. The array number is the ID of the coin.

Internally define three maps to determine the owner of the coins (coinToOwner), the number of coins acquired by the owner (ownerCoinCount), and the number of coins fed by the owner (coinFeedTime). Use mapping to complete.

This smart contract mainly defines five functions, the first is to input a name and create a coin (function createCoin (string memory _name) public{}), when verifying the sender's coin quantity is 0, obtain DNA according to the name, change the last digit of DNA to 0, initialize the coin struct, then complete the creation of the gold coin.The second is the random number function (function _generateRandomDna(string memory _str) private view returns(uint){}). The main function is to create a 16-bit random number based on the coin name and timestamp. The random number created by this function is called the creaateCoin function. The third is the internal contract to create the coin function (function _createCoin (string memory _name,uint _dna) internal{}), which completes the initialization of the coin, i.e. passing in parameters, name, DNA, obtaining the coin ID according to the array number, mapping the coin ID to the sender, mapping the coin owner's coin number +1, the total coin number +1, and finally triggering the event to generate the coin. It is then called in the creaateCoin function.The fourth is buyCoin function (function buyCoin(string memory _name) public payable{}). After verifying that the number of gold coins is greater than 0 and the amount sent is greater than the price of gold coins, random DNA is created. The last bit of DNA is 1, and then gold coins are generated. The final function is to set the gold price function (function setCoinPrice (uint _price) external onlyOwner{}), which the contract owner can change at will. Above are all the functions in the module coinFactory.

3.2.3. CoinHelper

In this module, we implement the contract in “CoinFactory”, define the upgrade cost variable(uint), create the level judgment modifier(aboveLevel), and the operator judgment modifier(onlyOwnerOf), design the functions called “setLevelUpFee”, “levelUp”, “changeName”, “getCoinByOwner”, “triggerCoolDown”, “isReady” and “multiply”. This contract serves for the “CoinFeeding” and the “CoinAttack” modules.

Implementation of this module:

Firstly, we define the upgrade cost as 0.001ether, then create a modifier called “aboveLevel” to verify whether the current coinId exceeds a given level, passing in the “level” and “coinId” variables. After that, another modifier called “onlyOwnerOf” is created, we pass in the “coinId” variable, map the sender of the confirmation message, and ensure that the operator can only be the owner of the current coin. When using these two modifiers to decorate other functions, the code of other functions will replace the last line in the modifier codes which is “_”.

The next step is creating functions that are needed in this contract. We start with the "setLevelUpFee” function, calling the “onlyOwner” modifier in the Ownable contract. This function can only be called externally to provide the contract owner with the function of modifying the upgrade fee. Secondly, we create the “levelUp” function. This function can only be called externally and it makes sure that users can upgrade coin by paying and they need to pass in the “coinId" variable. We use the "payable" modifier to indicate that the function can be paid. The process of this function works as follows: it verifies that the input amount is greater than or equal to the upgrade fee, and no matter how much the user enters, as long as the specified upgrade fee is reached, the coin can only be upgraded by one level. Because the level number is very large, even if the user invests a lot, it cannot bring corresponding large benefits. Therefore, overflow does not need to be considered. Third comes the “changeName” function. If the level of a coin is greater than two levels, the owner can rename it. We use “aboveLevel” and “onlyOwnerOf" modifiers to judge whether the level of the coin is greater than two levels and stipulate that only the owner of the coin can operate this function. After that, we create a function called “getCoinByOwner” which offers users an opportunity to find out how many coins they have owned and the corresponding coinIds. This function is external calling and read-only. We traverse all the data, query all the coins corresponding to the ID of the sender, store them, and then return them at one time. Enter the owner's address and return the unit array. Then create a fixed length array, whose length is the number of the coins owned by the user, declare a counter, and judge whether the address corresponding to each ID number is the owner's coin through the “coinToOwner" mapping function. If so, pass it into the array, and the counter is increased by one. Finally, return the array. Then another two functions are created: "triggerCoolDown” and “isReady”. They are both related to the cooling time. The first function will be used in the “multiply” function which is the most special and the last one of all the functions we create in the “CoinHelper” contract. In the “triggerCoolDown” function, the incoming coin construct can only be called internally. We modify the current structure to ensure that the cooling time is completed at 0 o'clock every day. When it comes to the “isReady” function, which is internal calling and read-only. We pass in the coin construct, judge whether the cooling time is met then continue the operation. Finally, the last one, is the "multiply" function. This function is an internal calling. In case of a coin battle, if the attacker wins, a new coin will be generated for him. At this time, to use this function to get the new coin, we combine the DNA of the winning coin with the DNA of the losing coin, pass in the ID of the current coin and the DNA of the target coin, and modify it with “onlyOwnerOf” to ensure that the caller must be the owner of the current coin. Then create a “myCoin” variable through the coin construct, verify the cooling time, and the target coin’s DNA must be 16 bits. Average the DNA of both sides to generate a new DNA, and the number at the end of the new DNA is set to 9, which serves as a marker, indicating that the coin is generated after the victory of an attack. Call “createCoin”, and set “Jeffery” as the name of the new coin, give it the new DNA, hand it to the winner, and trigger the attacker’s cooling time to limit the time of feeding, combining, and other operations.

3.2.4. CoinFeeding

In this module, we implement the contract in “CoinHelper”, and design the “feed” function.

Implementation of this module:

In the “feed” function, we input the “coinId” variable, modify this function with “onlyOwnerOf” to ensure that only the owner of the coin can call this function, create the temporary local variable “myCoin” of the coin structure, extract the coin structure from the array according to the “coinId”, assign it to “myCoin”, verify the cooling time, use “add (1)” to increase the feeding times by one, trigger the cooling time, and judge the feeding times. If the number of feeding times is more than 10, a new coin will be generated. The DNA of the new coin inherits the DNA of the original coin, and the mantissa is modified to 8, which acts as a marker, indicating that the coin is generated by feeding. Call the “createCoin” function in “CoinFactory” and set the new coin’s name as "Coin'son".

3.2.5. CoinAttack

In this module, we implement the contract in “CoinHelper”, create random number seeds and define the initial winning rate, design the “randMod” function, “setAttackVictoryPosibility” function, and “attack” function.

Implementation of this module:

We create a variable called “randNonce" which is used to count for the random number of seeds. Each time a random number is created, the value of "randNonce” is added by one, and the initial winning rate is set to 70%.

Then we create the “randMod” function, the input variable “modulus” controls the digits of the random numbers, performs "keccak256" hash operation on the timestamp, the address of the sender, and the random number seed, and then divides it by "modulus" to create a random number.

After that, the “setAttackVictoryPosibility” function, it can only be called by the contract owner to modify the winning rate.

At last, the “attack” function, let the IDs of two coins attack each other, judge the outcome according to the victory rate, and give the winner and loser their respectively deserved results. To do these, we assign a value (coinId) to “myCoin” structure and another (targetId) to “enemyCoin” structure, then take a random number less than 100, if the number is less than or equal to the victory rate, then the attacker wins, his victory times plus one, coin level plus one, obtaining a new coin through the “multiply” function in the “CoinHelper” contract, his cooling time is triggered and the failure times of the loser plus one. If the attacker fails, it will add one to the number of his failure times and one to the number of victory times of the enemy, and trigger the cooling time of the attacker.

3.2.6. CoinOwnership, CoinMarket and CoinCore

CoinOwnership. In this module, the owner address is mapped to coinId to realize ownership-based query and exchange. Mainly serve the next market module. This module is able to check the balance, query ownership, and transfer ownership directly and indirectly.

CoinMarket, which is based on the contract of the ownership module, mapping the coinId with the seller and price to realize basic functions such as setting the tax amount and minimum price, querying the number of coins in the market, selling coins, and buying the coin.

Coincore.Inheriting the attack, feeding, and market contracts, the module explains the coin name and abbreviation. By this module, the contract owner can query the contract balance and draw money from the contract.

4. Conclusion

In the current craze of blockchain, the concept of NFT is gradually popularized, and NFT products are gradually popular. Through the research, this paper realizes the development of NFT smart contract, realizes the functions of creation, attack, feeding, trading, and so on, enriches the diversity of non-fungible token smart contract, lays a foundation for realizing more functions and improving the content of contract in the future, and provides a certain reference.

NFT can give works of art uniqueness, scarcity, inseparability, traceability, openness, and transparency, difficulty to tamper with, renewal rights, and many other characteristics that traditional works of art do not have. In recent years, it has become popular and brought great changes to the traditional business model.

We also realize that there are still many points we can improve in these experiments. For example, The function completed in this experiment is only preliminary, and there are still many problems to be considered in terms of details and humanization. Besides, This experiment only involves the design of the smart contract system and has not carried out work on the corresponding front-end, which is what we will strive to achieve later.

In the future, we will explore the application of this smart contract in the field of digital art, online games, and digital currency and finance. We envision our product as a tradable, collectible, artistic NFT currency. It's also entertaining, while completing the currency function, it has the function of the game. We will continue to work towards this goal

References

[1] Wei Liting, Guo Yan, He Mengjiao, et al. Nonhomogeneous Tokens (NFTS): Logic, Applications, and Trends. Transactions of China Electrotechnical Society.doi:10.16110/j.cnki.issn2095-3151.2022.04.009.

[2] Zhang Yong. Research on the current situation of NFT based on blockchain technology. Guizhou Branch of National Computer Network and Information Security Management Center,2022,1672-7274(2022)05-0149-03

[3] Wang Zikai,Zhu Jian,Zhang BoJun,Hu Kai. Research and implementation of parallel method of Blockchain and smart contract. School of Computer Science, Beihang University, BeijingYunnan Key Laboratory of Blockchain Application Technology, Kunming.DOI: 10.11896/jsjkx.210800102

[4] Deng Jianpeng,Li Jianing. The right Certificate of Digital Artwork -- NFT value source, power dilemma and countermeasures, 2022.

[5] Luo Binrui, NFT encryption art characteristics analysis and commercial applications. 1007-9416(2022)07-0215-03.

[6] Zhao Ming,Dong Dazhi.Research on data asset management mechanism based on blockchain technology,Big data.4, 2021.

[7] Guo Quanzhong.NFTS and the Future,A news lover,11,2021.

[8] Yan Chi.NFT and Metaverse regulation path exploration in Digital era.

[9] Lamport, L. , R. Shostak , and M. Pease . "The Byzantine Generals Problem." ACM Transactions on Programming Languages and Systems 4.3(1982).

[10] Yang, W. , Garg, S. , Huang, Z. , and Kang, B. "A Hybrid Consensus Algorithm for Master–slave Blockchain in a Multidomain Conversation System." Expert Systems with Applications (2022).