# String Magnetic Protocol

catalogue

[String Magnetic Protocol 1](#_Toc31519)

[0x0 Project background 3](#_Toc20067)

[0x1 Preface 4](#_Toc17040)

[0x 2 Major issues in the current blockchain ecology 6](#_Toc3722)

[0x20 The problem of fair competition 6](#_Toc6597)

[0x21 classic Token price determinants 7](#_Toc12426)

[0x 21 Classic inflation-price incompatibility paradox 8](#_Toc28939)

[0x22 string magnetic protocol price algorithm scheme 8](#_Toc22154)

[0x23, the underlying formula 9](#_Toc20657)

[0x24 for the interpretation 10](#_Toc17219)

[0x3 string magnetic ecology 11](#_Toc26029)

[. 0x30 Node mining ecological plan 11](#_Toc30850)

[0x31 GameFi 12](#_Toc22653)

[0x32 string magnetic protocol financial ecology 13](#_Toc6924)

[0x33 string magnetic war game 13](#_Toc26634)

[0x34 NFT 15](#_Toc20094)

[0X35, a one-key coin issue 17](#_Toc20622)

[0x 36.ZK zero-knowledge anonymity protocol 18](#_Toc4358)

[0x4 Project timeline 21](#_Toc1749)

[0x5 contract code 22](#_Toc7736)

[0x6 Token Contract code 49](#_Toc21066)

[0x7 Reference 60](#_Toc22867)

[0x71.Fabian Vogelsteller 60](#_Toc25777)

[0x72.A Hands-On Tutorial 60](#_Toc666)

[0x73.ERC-20 (Ethereum 60](#_Toc17540)

[0x74. "How to Create an NFT 60](#_Toc22722)

[0x75.The Rise of Non-Fungible Tokens 60](#_Toc18244)

[0x76.Non-Fungible Tokens: 60](#_Toc3711)



## 0x0 Project background

This technical white paper explains some of the design logic and economic models behind the String Magnetic Protocol (String Magnetic Protocol) core contract. It's being used to solve the entire multi-chain ecology. After Web3, humans will enter a whole new digital age. String magnetic protocol (SMP) will become the digital currency dominating the economy, and its mining technology and smart contracts bind to a new revolutionary approach that allows every CPU to participate. As technology continues to advance, traditional cryptocurrency mining has become more centralized, and only miners with large amounts of dedicated hardware and huge computing power can benefit. However, the mining technique of the string magnetic protocol has completely changed this situation.

The mining technique of string magnetic protocol is based on a new algorithm that utilizes the special properties of string magnetic structure. Each CPU can be bound to the string magnetic smart contract and participate in the mining process of virtual coins.

To ensure fairness and security, the mining process of the string magnetic protocol requires binding to smart contracts. A smart contract is a computer program that includes the rules and conditions for mining. In order to protect users' privacy and data security, string magnetic virtual currency mining technology also uses the concept of zero knowledge proof. The zero-knowledge proof allows participants not to disclose any specific information about a statement without proving when it is true. This means that each CPU can indicate that it has contributed to the mining process by providing the necessary proof, without leaking its specific mining data.

As time went on, the value of string magnetic virtual coins rose, and more and more people began to join the mining network. People gradually realize that this mining technology based on string magnetic virtual currency will bring a new economy to mankind

## 0x1 Preface

"Bitcoin" is the successful implementation of the p2p e-cash concept. Both professionals and the general public are beginning to appreciate the convenient combination of public transactions and work certificates as a trust model. Today, the user base of e-cash is growing steadily growing; consumers are attracted by low fees and the anonymity of e-cash provision, and merchants value their projected and scattered emissions. Bitcoin effectively demonstrates that e-cash can be as simple as paper money and as convenient as a credit card.

However, Bitcoin does not complete the smart contracts like Ethereum, as a smart contract can complete a lot of blockchain functions. The string magnetic protocol is built on the Ethereum smart contract, which can solve the incompatible contradictions in the blockchain through the ecology. It can be both inflated and deflationary. Through GAMEFI, WEB 3 ecosystem is destroyed to achieve deflation, inflation is achieved through the node system, the deflation part is destroyed, and the inflation part is released to the node.

Blockchain string magnetic theory is realized according to the logic of physics. According to the most basic computing power of cpu, it can be regarded as string vibration. After everyone enters the mining, it is fair, because the computing power of cpu is regarded as equal vibration. Magnetic theory is like quantum entanglement, and the computing power generated by each individual mining machine is related to all other mining machines. To achieve the fair goal of computing power through the verification of cpu computing power and smart contract.

And the string magnetic protocol will solve the problem of bitcoin mining injustice, smart contracts can not be bound to mining machines. The string magnetic protocol is bound to the machine through the smart contract EVM, and the introduction of this string magnetic protocol mining technology makes the mining process more fair and inclusive. Each CPU has the opportunity to participate in the mining of virtual coins and get corresponding accordingly. As a result, the security of the entire network is also enhanced, because no single entity can control most of the network's computing power.

This fictitious scenario can make the mining process more fair and safe. The unique code of the equipment ensures that only legitimate equipment can participate in mining, and that each equipment can receive benefits corresponding to its computing power and participation time. Indeed, binding each user to a unique ID protocol is a way to increase the fairness of mining. This approach ensures that each user involved in mining has a separate identity and their revenue are bound to their unique ID.

Through this unique ID binding method, we can effectively prevent malicious behavior and cheating. Each user is tracked and identified, so that any violations can be traced back to its unique ID, thus maintaining the security and impartiality of the entire mining network. Moreover, the unique ID binding protocol can also provide a more precise revenue distribution. Depending on the unique ID of each user, smart contracts or other algorithms can be used to calculate its contribution and distribute the mining revenue accordingly. This helps to ensure that each user receives a fair return consistent with their actual contribution.

It should be noted that although a unique ID binding protocol can increase fairness and security, other factors such as computing power distribution, network effects and economic incentives still need to be considered. The design of mining systems requires considering multiple factors to achieve a more impartial, efficient and sustainable mining ecosystem.

In practice, ensuring that each user has a unique ID and binding the protocol may involve challenges in user authentication, key management, and data privacy. Therefore, when designing and implementing this mining method, the security and user privacy protection issues need to be carefully considered, and corresponding measures should be taken to solve these problems.

## 0x 2 Major issues in the current blockchain ecology

### 0x20 The problem of fair competition

Energy consumption: Bitcoin mining requires a lot of electricity to solve mathematical problems, which leads to huge energy consumption. Electricity costs are lower in some areas, while others are more expensive. This puts miners who have access to electricity at a lower cost at an unfair advantage in the mining competition.

Demand for specialized hardware: Bitcoin mining currently relies heavily on specially designed ASIC chips. These chips are highly computationally efficient and dedicated, allowing miners with these hardware to reap more mining returns at a lower cost. However, ordinary consumers have difficulty obtaining these special hardware, which leads to unequal competition among miners.

Pool concentration: As bitcoin mining is becoming more difficult, miners tend to join the pool and increase their chances of getting rewards by sharing their workload and rewards. However, a few large mining pools control most of the mining computing power, making it difficult for individual miners of those who do not join these pools to have a fair chance.

These problems make bitcoin mining less fair to some extent. It should be noted that Bitcoin is originally designed not to pursue absolute fairness, but to achieve security and reliability through a decentralized consensus mechanism.

### 0x21 classic Token price determinants

The price of a token is determined by several factors, including:

Supply and demand: The basic economic principles of supply and demand are the main factor determining the price of a token. If demand for tokens is high and supply is limited, prices could rise. If demand is low and supply is high, prices may fall.

Sentiment: The overall perception and attitude towards tokens will also have a significant impact on their prices. Positive news and developments can increase sentiment and push up prices, while negative news and events can lower sentiment and lower prices.

Adoption and use: The wider the adoption and use of tokens, the higher the value may be. Tokens with strong use cases, a large active user base, and strong partnerships are more likely to add value over time.

Competition: The presence of similar or competing tokens also affects the price of the token. A new tokens with similar functions and use cases could reduce demand for the original tokens, resulting to a lower price.

Regulatory environment: The regulatory environment can also play a role in determining the price of tokens. If the token is subject to restrictive regulations, it may reduce demand and lower prices, while favorable regulations may increase demand and raise prices.

These are some of the key factors that may affect the price of a token, the cryptocurrency market is highly volatile and unpredictable, and token prices may be influenced by many other factors.

Tokens weigh the supply and demand relationship of typical models based on tokens, and consider the influence of domestic factors (such as blockchain financial growth rate, inflation rate, interest rate, etc.) and the overall market value factors.

### 0x 21 Classic inflation-price incompatibility paradox

In classical economics, moderate inflation can stimulate the economy, but excessive inflation can cause violent fluctuations in the capital market and cause chaos in the capital market. According to market rules, the greater the inflation, the lower the price. This has been fully reflected in the digital currency market. The total inflation of many coins cannot be controlled, leading to a lower price spiral.

String magnetic protocols solve the classic inflation value incompatibility paradox through smart contracts. More algorithms can solve the problem of inflationary and deflation. The inflation part is given the weight to the node user. Through the combination of mining and Gamefi, the whole system locks in liquidity, reduces the positive correlation between inflation and price, makes inflation compatible with price, and forms a closed loop of ecological economy.

### 0x22 string magnetic protocol price algorithm scheme

Algorithmic price formula refers to the mathematical equation used to determine how the Token price should be established as conditions or other factors change. Depending on the assets assessed and the method of analysis, different models may be used.

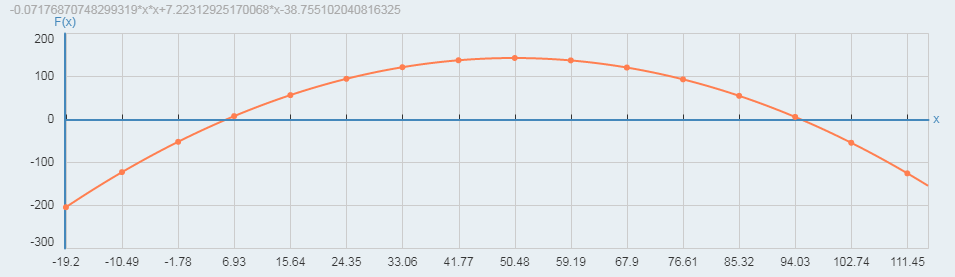
String magnetic protocol is a service as a platform. In order to enable many projects to have an ecosystem, it provides project side services to destroy the String Magnetic Protocol Token through the contract. The destroyed portion is permanently driven into the black hole address address (0). The chord magnetic protocol contract releases the same proportion of Token within a certain period of time according to the destruction ratio, and this part is issued as the certificate of interest of the node.

### 0x23, the underlying formula

Token, Total Output Calculation:



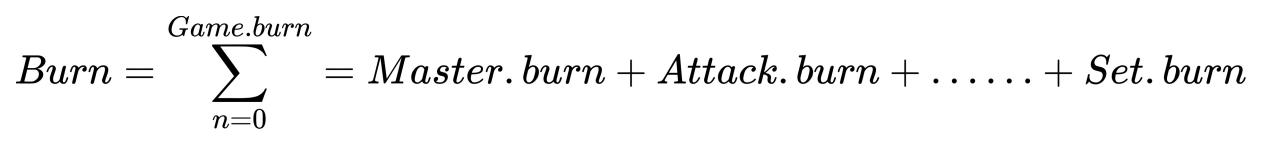
Fit the Token total amount curve:



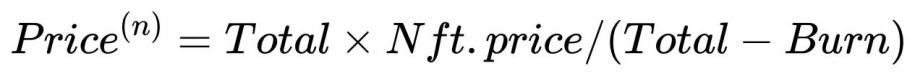
X standard: total output

Y standard: total flow volume

Token combustion calculation:



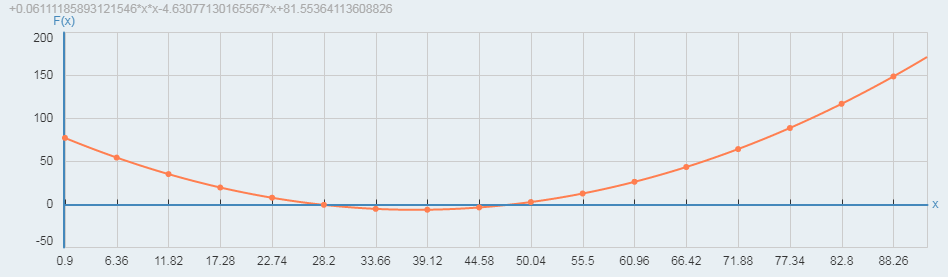
Token Price curve



Fit the price curve:

X mark: Nft market value

Y standard: Token value



### 0x24 for the interpretation

Token parameter:

Name: String Magnetic Protocol

Total amount: 210,000,000

symbol:SMP

Numbers: 12

Computing power output: 210,000,000 (100%)

Destruction: 105,000,000 (50%)

explain:

String Magnetic Protocol Is a decentralized transaction and wealth management agreement based on the standard ERC 20. Support mining machine mining, liquidity mining, DAO community, and other fund support functions. By providing the underlying technical support of blockchain, SMP will cooperate with multiple project parties in the game, which will provide game platforms and solutions for many project parties, which can empower the global game ecosystem. SMP is the world's first fully decentralized game platform, and also the first game public chain application platform bound to mining machines. Its design products are all based on Ethereum smart contract, not a pseudo-centralized platform. Community-driven multi-chain compatible Game and asset management protocols. Revenue fully into burn destruction. It uses immutable smart contracts, and deflation burns up to 50 percent of the total (about 105,000,000).100% of the revenue generated is distributed to the mining rights users. The founding team did not process the revenue artificially, and realized the autonomous strategy of "code for rules and contracts determine distribution".

String magnetic proof of equity: the blockchain equity certificate is the mining reward in the classic blockchain, in the file storage blockchain mode is the storage certificate, and in the string magnetic protocol is the string magnetic proof of interest, that is, the income generated by the certification part held is based on the total amount of ecological destruction distributed according to the weight. Simply short, how much the whole ecology is destroyed, how much will be produced. Token Circulation = Token ecological output = Token ecological destruction, Token Total =2.1B-Number of ecological destruction. Token total circulation = mine-out-destruction.

## 0x3 string magnetic ecology

### . 0x30 node mining machine ecology

Genesis mining: the string magnetic protocol creation node mining is created at the start of the project. Nodes assign their weights according to the number of donations. Everyone is fair, because the computing power of the CPU is regarded as equally vibration. Magnetic theory is like quantum entanglement, and the computing power generated by each individual mining machine is related to all other mining machines. Through the CPU computing power and smart contract to the verification of computing power to achieve the purpose of fair.

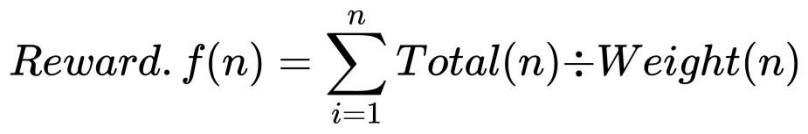
Super mining: After the creation node is started, any user can use the idle computer equipment to mine. Super mining coin creation world mining income has been reduced, because there is no bound mining machine equipment and identity ID verification. But if there is a strong equipment group as the base, then the production volume is also very large.

Total Token mining: 105,000,000.

Out speed: per minute.

Wheel: 5.

Block yield formula:



explain:

The initial node for the creation node will enjoy the initial project ecological reward. For example, if you get 10 percent of the weight, then you will get 10 percent of the net creation mining production. When the production of destruction reaches 10% of the total. The total amount of node mining 105,000,000 and game mining 105,000,000 is 210,000,000, and 50% of the mine is destroyed at the same time. Reached 100% zero pre-dig, 100% held by the miners.

### 0x31 GameFi

Combined with Game (Game) Finance (Finance) Mining (Mining)

GameFi The ecosystem uses cryptocurrencies, non-homogeneous tokens (NFT), and blockchain technology to create virtual games.

The game of string magnetic ecology consists of the following several subjects.

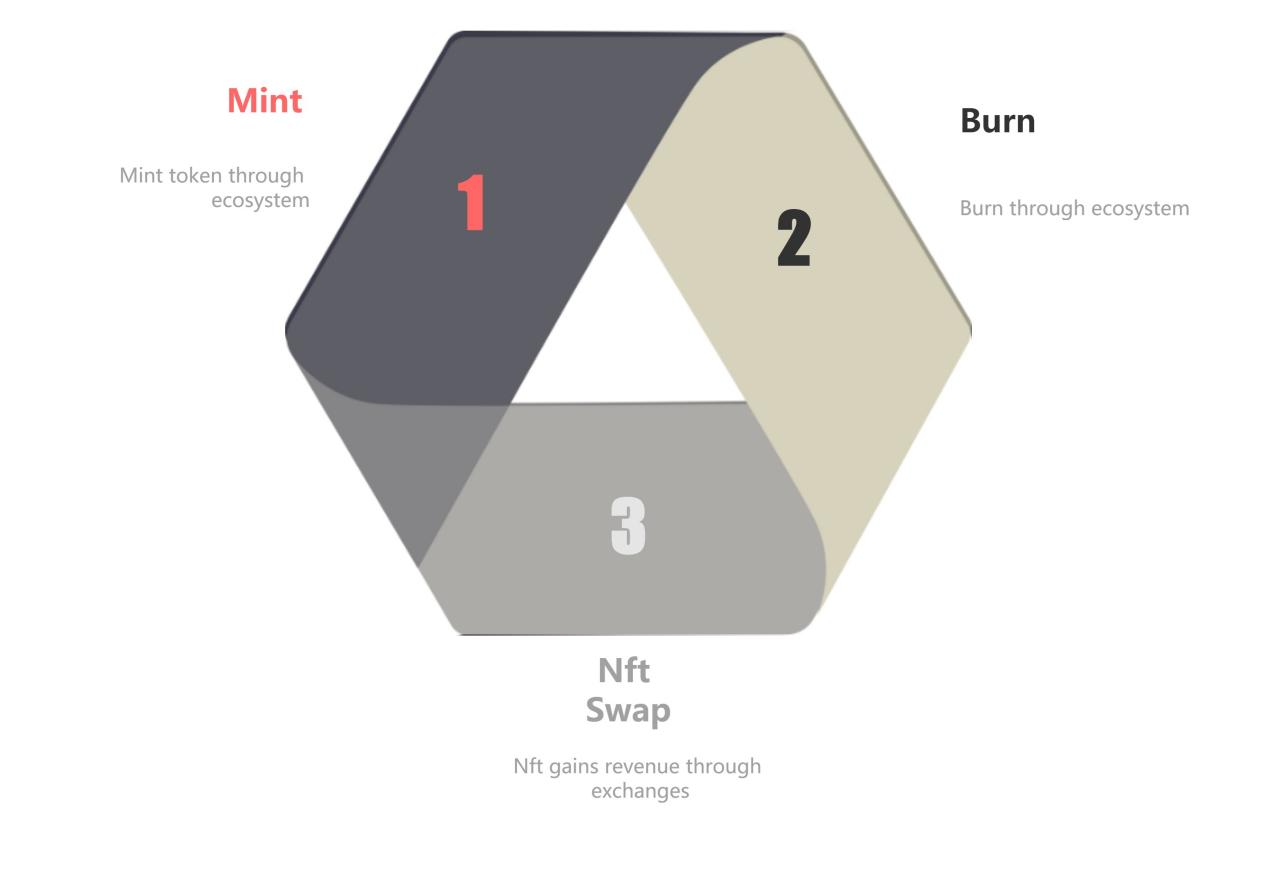
User: Play and trade virtual assets in games using the GameFi ecosystem. Make money in the play.

Token: The ecology of string magnetic protocols uses SMP, which can be used as virtual assets in games and other ecological applications.

String magnetic smart contracts: smart contracts used in the ecosystem to support game trading and economic models. Let the game destruction and game additional issuance to achieve a certain balance.

Architecture platform: a developer platform that provides technical support for the ecosystem and develops new features.

Through these elements, the string magnetic protocol ecosystem tries to solve incompatible economic models and combine results with smart contracts into the gaming space, providing a stable, transparent, and credible closed loop of economic gaming.



### 0x32 string magnetic protocol financial ecology

It mainly consists of three parts:

Coin casting: release the income of node users through coin casting, and coin casting is actually increasing the circulation part.

Burning: Burning off the Token through the ecosystem. The amount of combustion can be calculated and adjusted according to the ecological situation.

NFT economy: Buy NFT through game player configuration and buy back some SMPs.

Through these three parts, the whole ecology forms the economic closed loop to avoid malicious inflation and excessive deflation.

### 0x33 string magnetic war game

The string magnetic national war game ecology only provides basic consumption as a game platform, while the game itself requires other project parties to enter the whole ecology. There are two main tokens in the platform, namely String Magnetic Protocol Token (SMP) and SMP NFT.

Token (n) (Token of the participating ecological project party)

Aggression / defense force

SMP to add the game base attribute token

The 1-point energy = 1 token

1 token = 1 point of attack or defense

(Token (n)) As the in-game currency, the anchor attribute price is 1 USDT: Token (n) =1 base attribute, which also has a very good means of acquisition.

Entering the magnetic land war system, each adventurer is sheltered by the kingdom energy, adding different country tokens depending on the country the player is in.

scene

1. Gameplay

There are many countries on the magical crypto continent. As a warrior of the country, players can summon the red dragon, Thai tower, archangel, nine-headed dragon and other summoning beasts for adventure. Players can play the role of a castle master and build powerful forces through expeditions around the world. Players can also go to other countries, communicate with different forces, bring down all kinds of powerful enemies, and build an immortal legend by winning an epic war.

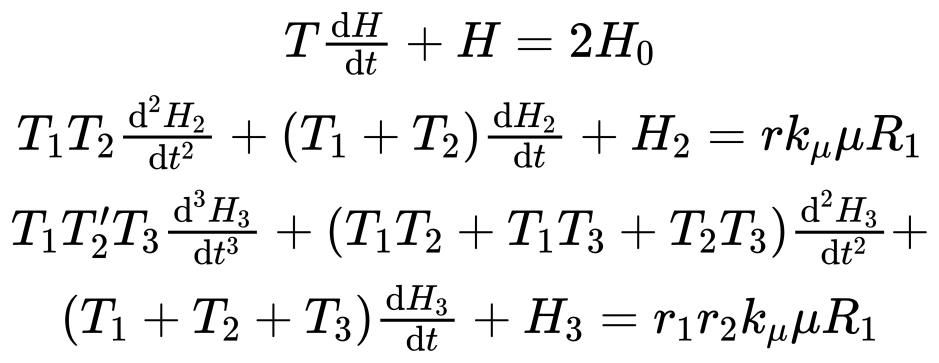
2. Income from Token (n)

User revenue is determined based on the user attack power and the country base and NFT bonus,

Output country token = (attack force / (country base) Xnft plus) \* min

If the attack force is greater than the defense force of the other side, it can rob the other side did not get the income token

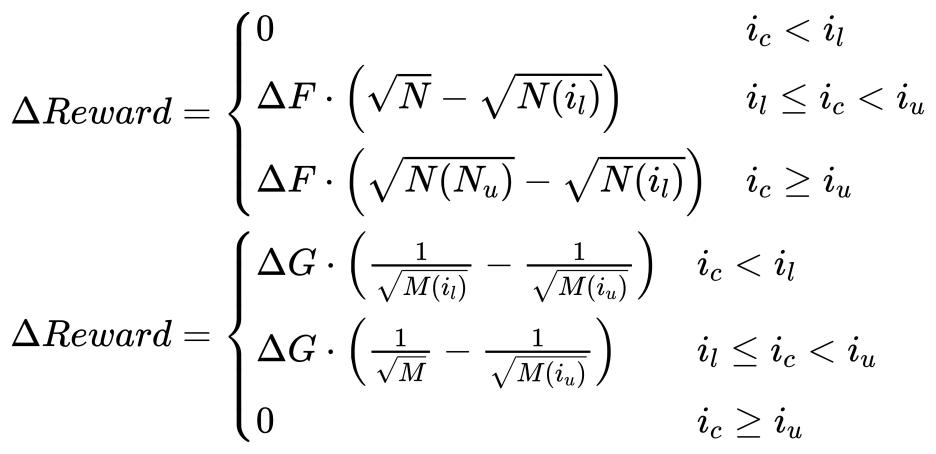
3. Calculation formula of deflation T-H attenuation:



4. Globalized ecological king

In order to defeat other countries, each country produces a king, and the king earns twice as much as the other players, and wins the title by challenging the king to defeat the king.

5. Game inflation formula:



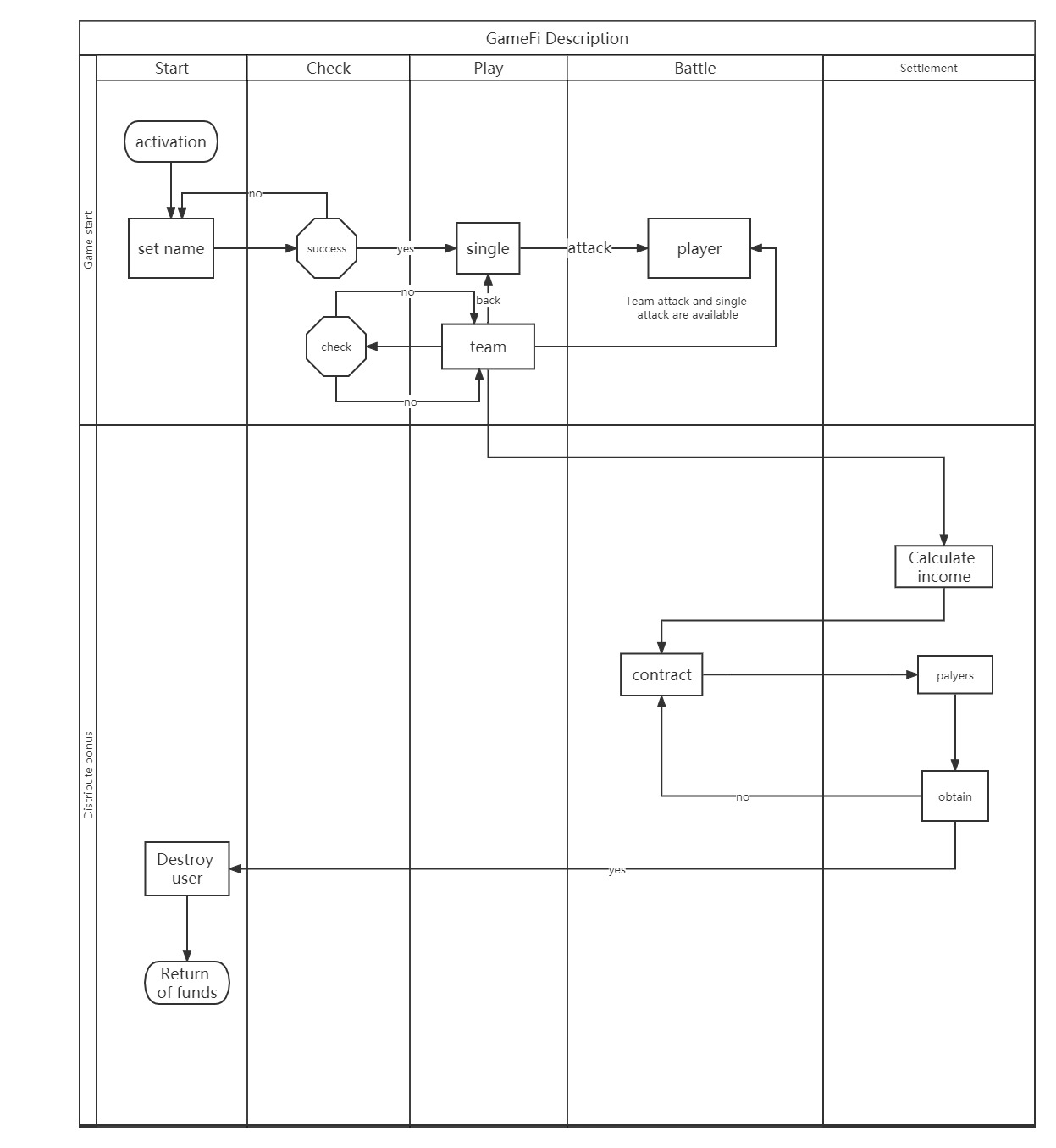
### 0x34 NFT

The limited version of the NFT is attached to the game.

1. Users can go to the NFT exchange to buy NFT items to improve the player's attributes.

2. The NFT equipment of string magnetic war is limited edition, adding three ordinary attributes and two special attributes. The attack force defense was increased by percentage, and the attack cooldown was reduced by N seconds. The special attribute is remote attack, time travel.

Game flow chart



NFT endorses the chord-magnetic economy ecology. NFT sales are all the main coins on the chain. After the sales, SMP tokens are bought back regularly to solve the problem of ecological price and inflation conflict.

NFT parameter list

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | Num | PRICE | ATT | DEF | TIME | STUNT |
| VALKYRIE | 400 | 2 | 2 | 0 | 0 | NULL |
| ENRAGED CYCLOPIA | 400 | 2 | 1 | 2 | 0 | NULL |
| ANCIENT TREANT | 100 | 4 | 2 | 6 | 0 | NULL |
| ABBOT | 100 | 4 | 0 | 4 | 200 | CHAT |
| GRIM REAPER | 100 | 10 | 13 | 2 | 0 | NULL |
| EMERALD DRAGON | 50 | 20 | 10 | 10 | 1500 | CHAT |
| SIMURGH | 50 | 20 | 20 | 5 | 0 | DISTANCE SPACE |
| SPECTRAL DRAGON | 50 | 20 | 10 | 35 | 100 | SPACE |
| ANCIENT BEHEMOTH | 50 | 20 | 30 | 1 | 100 | DISTANCE |
| DARK HYDRA | 50 | 20 | 20 | 20 | 100 | SPACE |
| TITAN | 50 | 20 | 10 | 40 | 100 | SPACE |
| ARDENT DRAGON | 30 | 30 | 40 | 20 | 100 | DISTANCE SPACE |
| CELESTIAL | 20 | 50 | 50 | 50 | 400 | ALL |
| BLACK DRAGON | 20 | 50 | 60 | 30 | 400 | ALL |

1. NFT to to properties. After mining, users can purchase some limited edition NFT to add mining. When you hold an NFT, the contract pool and the mine pool will add some additional revenue based on your mining power. NFT has acceleration and gain effect on mining.

Special NFT for the mining machine

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | NUM | REWARD | PRICE |
| CPU | 50 | 10% | 2 |
| GPU | 50 | 20% | 3.5 |
| QUANTUM | 10 | 30% | 5 |
| QUARK | 10 | 40% | 6.5 |

Note: NFT pricing is the force conversion value, and the ore gain will be adjusted regularly according to the agreement

### 0X35, a one-key coin issue

One-click creation coin function (TOKEN) will be available on the Ecological DAPP.

Ecology is committed to popularizing the blockchain, allowing ordinary users to create their own coins for only a small cost. Users only need to pay a little ecological TOKEN to complete the function of one-click coin issuance.

### 0x 36.ZK zero-knowledge anonymity protocol

A Zero-knowledge proof (Zero-Knowledge Proof) is a technique used to prove certain facts without the need to reveal any useful information. It can be used to prove that a person has a particular identity, the authenticity of an item, etc. The importance of zero-knowledge proof is that it can prove a fact while protecting privacy. String magnetic protocols provide an anonymous trading system for ecology. Through anonymous transactions, users' privacy can be better protected.

The zero-knowledge proof of anonymous transactions is a technique used to protect the privacy of both parties to the transaction. It allows the parties to the transaction to prove the legality of the transaction, but they need not disclose any identifying information about themselves. The technology is often used in the cryptocurrency domain to protect the privacy of both sides of the transaction. With zero-knowledge proof, the parties can prove that they have enough money to complete the transaction, but do not need to reveal their identity. In this way, it can protect the privacy of both parties and ensure the security of the transaction. ZK zero-knowledge Anonymous Agreement contract generates a pair of public and private key receipts to users for anonymous transactions. According to the public and private key pairing, users can conduct various Token transactions, such as USDT, ETH, BUSD, etc.

The logic is as follows:



The logic is as follows:

Kate sends out the transaction. Billy checks each passing transaction using its public key (a, b) and computes P'= Hs (aR) G + b. If Kate includes Billy as a transaction of the recipient, a R = arG = rA and P'=P. Billy can recover the corresponding one-time private key: x = Hs (aR) + b, so P = xG. He can use this output at any time by signing transactions with x.



Thus, Billy obtained the incoming payments associated with the one-time public key

It is immovable to the audience. Some additional notes:

When Billy "identifies" his transaction (see step 3), he actually uses only half of the private information: (a, B). This pair of keys, also called a tracking key, can be passed to a third party (Jim). Billy could commission her with the new deal. Billy doesn't need to explicitly trust Jim, because she can't recover the one-time key p without a full Billy private key (a, b). This approach is useful when Billy lacks bandwidth or computing power (smartphones, hardware wallets, etc.). If Kate wants to prove that she sent the transaction to Billy's address, she can disclose r or use any type of zero-knowledge agreement to prove that she knows r (for example by signing the transaction with r). If Billy wants to have an audit-compatible address, all incoming transactions can be linked and he can publish the trace key or use a truncated address. This address represents only one public key B, and the rest of the protocol is exported as follows: a = Hs (B) and a = Hs) G. In both cases, each person was able to "identify" all of Billy's transactions, but of course nobody could use the funds in them without the secret key b.

## 0x4 Project timeline



Fourth quarter of 2022: Complete the economic model feasibility demonstration of the SMP project.

First quarter of 2023: Complete the recruitment of the creation node.

Q 22023: Complete the Gamefi String Magnetic National Wargame Core Smart Contract Code.

Q 32023: String Magnetic National War game launch.

Q 42023: Zero-knowledge anonymity protocol completed.

Q 1,2024:..............................

## 0x5 contract code

The NFT contract code

// File: openzeppelin-contracts-master/contracts/introspection/IERC165.sol

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.2;

/\*\*

\* @dev Interface of the ERC165 standard, as defined in the

\* https://eips.ethereum.org/EIPS/eip-165[EIP].

\*

\* Implementers can declare support of contract interfaces, which can then be

\* queried by others ({ERC165Checker}).

\*

\* For an implementation, see {ERC165}.

\*/

interface IERC165 {

/\*\*

\* @dev Returns true if this contract implements the interface defined by

\* `interfaceId`.See the corresponding

\* https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]

\* to learn more about how these ids are created.

\*

\* This function call must use less than 30 000 gas.

\*/

function supportsInterface(bytes4 interfaceId) external view returns (bool);

}

// File: openzeppelin-contracts-master/contracts/token/ERC1155/IERC1155.sol

pragma solidity ^0.6.2;

/\*\*

\* @dev Required interface of an ERC1155 compliant contract, as defined in the

\* https://eips.ethereum.org/EIPS/eip-1155[EIP].

\*

\* \_Available since v3.1.\_

\*/

interface IERC1155 is IERC165 {

/\*\*

\* @dev Emitted when `value` tokens of token type `id` are transferred from `from` to `to` by `operator`.

\*/

event TransferSingle(address indexed operator, address indexed from, address indexed to, uint256 id, uint256 value);

/\*\*

\* @dev Equivalent to multiple {TransferSingle} events, where `operator`, `from` and `to` are the same for all

\* transfers.

\*/

event TransferBatch(address indexed operator, address indexed from, address indexed to, uint256[] ids, uint256[] values);

/\*\*

\* @dev Emitted when `account` grants or revokes permission to `operator` to transfer their tokens, according to

\* `approved`.

\*/

event ApprovalForAll(address indexed account, address indexed operator, bool approved);

/\*\*

\* @dev Emitted when the URI for token type `id` changes to `value`, if it is a non-programmatic URI.

\*

\* If an {URI} event was emitted for `id`, the standard

\* https://eips.ethereum.org/EIPS/eip-1155#metadata-extensions[guarantees] that `value` will equal the value

\* returned by {IERC1155MetadataURI-uri}.

\*/

event URI(string value, uint256 indexed id);

/\*\*

\* @dev Returns the amount of tokens of token type `id` owned by `account`.

\*

\* Requirements:

\*

\* - `account` cannot be the zero address.

\*/

function balanceOf(address account, uint256 id) external view returns (uint256);

/\*\*

\* @dev xref:ROOT:erc1155.adoc#batch-operations[Batched] version of {balanceOf}.

\*

\* Requirements:

\*

\* - `accounts` and `ids` must have the same length.

\*/

function balanceOfBatch(address[] calldata accounts, uint256[] calldata ids) external view returns (uint256[] memory);

/\*\*

\* @dev Grants or revokes permission to `operator` to transfer the caller's tokens, according to `approved`,

\*

\* Emits an {ApprovalForAll} event.

\*

\* Requirements:

\*

\* - `operator` cannot be the caller.

\*/

function setApprovalForAll(address operator, bool approved) external;

/\*\*

\* @dev Returns true if `operator` is approved to transfer ``account``'s tokens.

\*

\* See {setApprovalForAll}.

\*/

function isApprovedForAll(address account, address operator) external view returns (bool);

/\*\*

\* @dev Transfers `amount` tokens of token type `id` from `from` to `to`.

\*

\* Emits a {TransferSingle} event.

\*

\* Requirements:

\*

\* - `to` cannot be the zero address.

\* - If the caller is not `from`, it must be have been approved to spend ``from``'s tokens via {setApprovalForAll}.

\* - `from` must have a balance of tokens of type `id` of at least `amount`.

\* - If `to` refers to a smart contract, it must implement {IERC1155Receiver-onERC1155Received} and return the

\* acc SMP ance magic value.

\*/

function safeTransferFrom(address from, address to, uint256 id, uint256 amount, bytes calldata data) external;

/\*\*

\* @dev xref:ROOT:erc1155.adoc#batch-operations[Batched] version of {safeTransferFrom}.

\*

\* Emits a {TransferBatch} event.

\*

\* Requirements:

\*

\* - `ids` and `amounts` must have the same length.

\* - If `to` refers to a smart contract, it must implement {IERC1155Receiver-onERC1155BatchReceived} and return the

\* acc SMP ance magic value.

\*/

function safeBatchTransferFrom(address from, address to, uint256[] calldata ids, uint256[] calldata amounts, bytes calldata data) external;

}

// File: openzeppelin-contracts-master/contracts/token/ERC1155/IERC1155MetadataURI.sol

pragma solidity ^0.6.2;

/\*\*

\* @dev Interface of the optional ERC1155MetadataExtension interface, as defined

\* in the https://eips.ethereum.org/EIPS/eip-1155#metadata-extensions[EIP].

\*

\* \_Available since v3.1.\_

\*/

interface IERC1155MetadataURI is IERC1155 {

/\*\*

\* @dev Returns the URI for token type `id`.

\*

\* If the `\{id\}` substring is present in the URI, it must be replaced by

\* clients with the actual token type ID.

\*/

function uri(uint256 id) external view returns (string memory);

}

// File: openzeppelin-contracts-master/contracts/token/ERC1155/IERC1155Receiver.sol

pragma solidity ^0.6.2;

/\*\*

\* \_Available since v3.1.\_

\*/

interface IERC1155Receiver is IERC165 {

/\*\*

@dev Handles the receipt of a single ERC1155 token type.This function is

called at the end of a `safeTransferFrom` after the balance has been updated.

To acc SMP the transfer, this must return

`bytes4(keccak256("onERC1155Received(address,address,uint256,uint256,bytes)"))`

(i.e.0xf23a6e61, or its own function selector).

@param operator The address which initiated the transfer (i.e.msg.sender)

@param from The address which previously owned the token

@param id The ID of the token being transferred

@param value The amount of tokens being transferred

@param data Additional data with no specified format

@return `bytes4(keccak256("onERC1155Received(address,address,uint256,uint256,bytes)"))` if transfer is allowed

\*/

function onERC1155Received(

address operator,

address from,

uint256 id,

uint256 value,

bytes calldata data

)

external

returns(bytes4);

/\*\*

@dev Handles the receipt of a multiple ERC1155 token types.This function

is called at the end of a `safeBatchTransferFrom` after the balances have

been updated.To acc SMP the transfer(s), this must return

`bytes4(keccak256("onERC1155BatchReceived(address,address,uint256[],uint256[],bytes)"))`

(i.e.0xbc197c81, or its own function selector).

@param operator The address which initiated the batch transfer (i.e.msg.sender)

@param from The address which previously owned the token

@param ids An array containing ids of each token being transferred (order and length must match values array)

@param values An array containing amounts of each token being transferred (order and length must match ids array)

@param data Additional data with no specified format

@return `bytes4(keccak256("onERC1155BatchReceived(address,address,uint256[],uint256[],bytes)"))` if transfer is allowed

\*/

function onERC1155BatchReceived(

address operator,

address from,

uint256[] calldata ids,

uint256[] calldata values,

bytes calldata data

)

external

returns(bytes4);

}

// File: openzeppelin-contracts-master/contracts/GSN/Context.sol

pragma solidity ^0.6.2;

/\*

\* @dev Provides information about the current execution context, including the

\* sender of the transaction and its data.While these are generally available

\* via msg.sender and msg.data, they should not be accessed in such a direct

\* manner, since when dealing with GSN meta-transactions the account sending and

\* paying for execution may not be the actual sender (as far as an application

\* is concerned).

\*

\* This contract is only required for intermediate, library-like contracts.

\*/

abstract contract Context {

function \_msgSender() internal view virtual returns (address payable) {

return msg.sender;

}

function \_msgData() internal view virtual returns (bytes memory) {

this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691

return msg.data;

}

}

// File: openzeppelin-contracts-master/contracts/introspection/ERC165.sol

pragma solidity ^0.6.2;

/\*\*

\* @dev Implementation of the {IERC165} interface.

\*

\* Contracts may inherit from this and call {\_registerInterface} to declare

\* their support of an interface.

\*/

contract ERC165 is IERC165 {

/\*

\* bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7

\*/

bytes4 private constant \_INTERFACE\_ID\_ERC165 = 0x01ffc9a7;

/\*\*

\* @dev Mapping of interface ids to whether or not it's supported.

\*/

mapping(bytes4 => bool) private \_supportedInterfaces;

constructor () internal {

// Derived contracts need only register support for their own interfaces,

// we register support for ERC165 itself here

\_registerInterface(\_INTERFACE\_ID\_ERC165);

}

/\*\*

\* @dev See {IERC165-supportsInterface}.

\*

\* Time complexity O(1), guaranteed to always use less than 30 000 gas.

\*/

function supportsInterface(bytes4 interfaceId) public view override returns (bool) {

return \_supportedInterfaces[interfaceId];

}

/\*\*

\* @dev Registers the contract as an implementer of the interface defined by

\* `interfaceId`.Support of the actual ERC165 interface is automatic and

\* registering its interface id is not required.

\*

\* See {IERC165-supportsInterface}.

\*

\* Requirements:

\*

\* - `interfaceId` cannot be the ERC165 invalid interface (`0xffffffff`).

\*/

function \_registerInterface(bytes4 interfaceId) internal virtual {

require(interfaceId != 0xffffffff, "ERC165: invalid interface id");

\_supportedInterfaces[interfaceId] = true;

}

}

// File: openzeppelin-contracts-master/contracts/math/SafeMath.sol

pragma solidity ^0.6.2;

/\*\*

\* @dev Wrappers over Solidity's arithmetic operations with added overflow

\* checks.

\*

\* Arithmetic operations in Solidity wrap on overflow.This can easily result

\* in bugs, because programmers usually assume that an overflow raises an

\* error, which is the standard behavior in high level programming languages.

\* `SafeMath` restores this intuition by reverting the transaction when an

\* operation overflows.

\*

\* Using this library instead of the unchecked operations eliminates an entire

\* class of bugs, so it's recommended to use it always.

\*/

library SafeMath {

/\*\*

\* @dev Returns the addition of two unsigned integers, reverting on

\* overflow.

\*

\* Counterpart to Solidity's `+` operator.

\*

\* Requirements:

\*

\* - Addition cannot overflow.

\*/

function add(uint256 a, uint256 b) internal pure returns (uint256) {

uint256 c = a + b;

require(c >= a, "SafeMath: addition overflow");

return c;

}

/\*\*

\* @dev Returns the subtraction of two unsigned integers, reverting on

\* overflow (when the result is negative).

\*

\* Counterpart to Solidity's `-` operator.

\*

\* Requirements:

\*

\* - Subtraction cannot overflow.

\*/

function sub(uint256 a, uint256 b) internal pure returns (uint256) {

return sub(a, b, "SafeMath: subtraction overflow");

}

/\*\*

\* @dev Returns the subtraction of two unsigned integers, reverting with custom message on

\* overflow (when the result is negative).

\*

\* Counterpart to Solidity's `-` operator.

\*

\* Requirements:

\*

\* - Subtraction cannot overflow.

\*/

function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b <= a, errorMessage);

uint256 c = a - b;

return c;

}

/\*\*

\* @dev Returns the multiplication of two unsigned integers, reverting on

\* overflow.

\*

\* Counterpart to Solidity's `\*` operator.

\*

\* Requirements:

\*

\* - Multiplication cannot overflow.

\*/

function mul(uint256 a, uint256 b) internal pure returns (uint256) {

// Gas optimization: this is cheaper than requiring 'a' not being zero, but the

// benefit is lost if 'b' is also tested.

// See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522

if (a == 0) {

return 0;

}

uint256 c = a \* b;

require(c / a == b, "SafeMath: multiplication overflow");

return c;

}

/\*\*

\* @dev Returns the integer division of two unsigned integers.Reverts on

\* division by zero.The result is rounded towards zero.

\*

\* Counterpart to Solidity's `/` operator.Note: this function uses a

\* `revert` opcode (which leaves remaining gas untouched) while Solidity

\* uses an invalid opcode to revert (consuming all remaining gas).

\*

\* Requirements:

\*

\* - The divisor cannot be zero.

\*/

function div(uint256 a, uint256 b) internal pure returns (uint256) {

return div(a, b, "SafeMath: division by zero");

}

/\*\*

\* @dev Returns the integer division of two unsigned integers.Reverts with custom message on

\* division by zero.The result is rounded towards zero.

\*

\* Counterpart to Solidity's `/` operator.Note: this function uses a

\* `revert` opcode (which leaves remaining gas untouched) while Solidity

\* uses an invalid opcode to revert (consuming all remaining gas).

\*

\* Requirements:

\*

\* - The divisor cannot be zero.

\*/

function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b > 0, errorMessage);

uint256 c = a / b;

// assert(a == b \* c + a % b); // There is no case in which this doesn't hold

return c;

}

/\*\*

\* @dev Returns the remainder of dividing two unsigned integers.(unsigned integer modulo),

\* Reverts when dividing by zero.

\*

\* Counterpart to Solidity's `%` operator.This function uses a `revert`

\* opcode (which leaves remaining gas untouched) while Solidity uses an

\* invalid opcode to revert (consuming all remaining gas).

\*

\* Requirements:

\*

\* - The divisor cannot be zero.

\*/

function mod(uint256 a, uint256 b) internal pure returns (uint256) {

return mod(a, b, "SafeMath: modulo by zero");

}

/\*\*

\* @dev Returns the remainder of dividing two unsigned integers.(unsigned integer modulo),

\* Reverts with custom message when dividing by zero.

\*

\* Counterpart to Solidity's `%` operator.This function uses a `revert`

\* opcode (which leaves remaining gas untouched) while Solidity uses an

\* invalid opcode to revert (consuming all remaining gas).

\*

\* Requirements:

\*

\* - The divisor cannot be zero.

\*/

function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b != 0, errorMessage);

return a % b;

}

}

// File: openzeppelin-contracts-master/contracts/utils/Address.sol

pragma solidity ^0.6.2;

/\*\*

\* @dev Collection of functions related to the address type

\*/

library Address {

/\*\*

\* @dev Returns true if `account` is a contract.

\*

\* [IMPORTANT]

\* ====

\* It is unsafe to assume that an address for which this function returns

\* false is an externally-owned account (EOA) and not a contract.

\*

\* Among others, `isContract` will return false for the following

\* types of addresses:

\*

\*  - an externally-owned account

\*  - a contract in construction

\*  - an address where a contract will be created

\*  - an address where a contract lived, but was destroyed

\* ====

\*/

function isContract(address account) internal view returns (bool) {

// This method relies on extcodesize, which returns 0 for contracts in

// construction, since the code is only stored at the end of the

// constructor execution.

uint256 size;

// solhint-disable-next-line no-inline-assembly

assembly { size := extcodesize(account) }

return size > 0;

}

/\*\*

\* @dev Replacement for Solidity's `transfer`: sends `amount` wei to

\* `recipient`, forwarding all available gas and reverting on errors.

\*

\* https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost

\* of certain opcodes, possibly making contracts go over the 2300 gas limit

\* imposed by `transfer`, making them unable to receive funds via

\* `transfer`.{sendValue} removes this limitation.

\*

\* https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].

\*

\* IMPORTANT: because control is transferred to `recipient`, care must be

\* taken to not create reentrancy vulnerabilities.Consider using

\* {ReentrancyGuard} or the

\* https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].

\*/

function sendValue(address payable recipient, uint256 amount) internal {

require(address(this).balance >= amount, "Address: insufficient balance");

// solhint-disable-next-line avoid-low-level-calls, avoid-call-value

(bool success, ) = recipient.call{ value: amount }("");

require(success, "Address: unable to send value, recipient may have reverted");

}

/\*\*

\* @dev Performs a Solidity function call using a low level `call`.A

\* plain`call` is an unsafe replacement for a function call: use this

\* function instead.

\*

\* If `target` reverts with a revert reason, it is bubbled up by this

\* function (like regular Solidity function calls).

\*

\* Returns the raw returned data.To convert to the expected return value,

\* use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].

\*

\* Requirements:

\*

\* - `target` must be a contract.

\* - calling `target` with `data` must not revert.

\*

\* \_Available since v3.1.\_

\*/

function functionCall(address target, bytes memory data) internal returns (bytes memory) {

return functionCall(target, data, "Address: low-level call failed");

}

/\*\*

\* @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with

\* `errorMessage` as a fallback revert reason when `target` reverts.

\*

\* \_Available since v3.1.\_

\*/

function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {

return functionCallWithValue(target, data, 0, errorMessage);

}

/\*\*

\* @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],

\* but also transferring `value` wei to `target`.

\*

\* Requirements:

\*

\* - the calling contract must have an ETH balance of at least `value`.

\* - the called Solidity function must be `payable`.

\*

\* \_Available since v3.1.\_

\*/

function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {

return functionCallWithValue(target, data, value, "Address: low-level call with value failed");

}

/\*\*

\* @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but

\* with `errorMessage` as a fallback revert reason when `target` reverts.

\*

\* \_Available since v3.1.\_

\*/

function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {

require(address(this).balance >= value, "Address: insufficient balance for call");

require(isContract(target), "Address: call to non-contract");

// solhint-disable-next-line avoid-low-level-calls

(bool success, bytes memory returndata) = target.call{ value: value }(data);

return \_verifyCallResult(success, returndata, errorMessage);

}

/\*\*

\* @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],

\* but performing a static call.

\*

\* \_Available since v3.3.\_

\*/

function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {

return functionStaticCall(target, data, "Address: low-level static call failed");

}

/\*\*

\* @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],

\* but performing a static call.

\*

\* \_Available since v3.3.\_

\*/

function functionStaticCall(address target, bytes memory data, string memory errorMessage) internal view returns (bytes memory) {

require(isContract(target), "Address: static call to non-contract");

// solhint-disable-next-line avoid-low-level-calls

(bool success, bytes memory returndata) = target.staticcall(data);

return \_verifyCallResult(success, returndata, errorMessage);

}

/\*\*

\* @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],

\* but performing a delegate call.

\*

\* \_Available since v3.3.\_

\*/

function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory) {

return functionDelegateCall(target, data, "Address: low-level delegate call failed");

}

/\*\*

\* @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],

\* but performing a delegate call.

\*

\* \_Available since v3.3.\_

\*/

function functionDelegateCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {

require(isContract(target), "Address: delegate call to non-contract");

// solhint-disable-next-line avoid-low-level-calls

(bool success, bytes memory returndata) = target.delegatecall(data);

return \_verifyCallResult(success, returndata, errorMessage);

}

function \_verifyCallResult(bool success, bytes memory returndata, string memory errorMessage) private pure returns(bytes memory) {

if (success) {

return returndata;

} else {

// Look for revert reason and bubble it up if present

if (returndata.length > 0) {

// The easiest way to bubble the revert reason is using memory via assembly

// solhint-disable-next-line no-inline-assembly

assembly {

let returndata\_size := mload(returndata)

revert(add(32, returndata), returndata\_size)

}

} else {

revert(errorMessage);

}

}

}

}

// @dev Implementation for different URIs for every token

contract TokenURI {

// mapping for token URIs

mapping(uint256 => string) private \_tokenURIs;

function \_tokenURI(uint256 tokenId) internal view returns (string memory) {

return \_tokenURIs[tokenId];

}

function \_setTokenURI(uint256 tokenId, string memory tokenUri) virtual internal {

\_tokenURIs[tokenId] = tokenUri;

}

}

// File: openzeppelin-contracts-master/contracts/token/ERC1155/ERC1155.sol

pragma solidity ^0.6.2;

/\*\*

\*

\* @dev Implementation of the basic standard multi-token.

\* See https://eips.ethereum.org/EIPS/eip-1155

\* Originally based on code by Enjin: https://github.com/enjin/erc-1155

\*

\* \_Available since v3.1.\_

\*/

contract ERC1155 is Context, ERC165, IERC1155, IERC1155MetadataURI, TokenURI {

using SafeMath for uint256;

using Address for address;

// Mapping from token ID to account balances

mapping (uint256 => mapping(address => uint256)) private \_balances;

// Mapping from account to operator approvals

mapping (address => mapping(address => bool)) private \_operatorApprovals;

// Used as the URI for all token types by relying on ID substitution, e.g.https://token-cdn-domain/{id}.json

string private \_uri;

/\*

\*     bytes4(keccak256('balanceOf(address,uint256)')) == 0x00fdd58e

\*     bytes4(keccak256('balanceOfBatch(address[],uint256[])')) == 0x4e1273f4

\*     bytes4(keccak256('setApprovalForAll(address,bool)')) == 0xa22cb465

\*     bytes4(keccak256('isApprovedForAll(address,address)')) == 0xe985e9c5

\*     bytes4(keccak256('safeTransferFrom(address,address,uint256,uint256,bytes)')) == 0xf242432a

\*     bytes4(keccak256('safeBatchTransferFrom(address,address,uint256[],uint256[],bytes)')) == 0x2eb2c2d6

\*

\*     => 0x00fdd58e ^ 0x4e1273f4 ^ 0xa22cb465 ^

\*        0xe985e9c5 ^ 0xf242432a ^ 0x2eb2c2d6 == 0xd9b67a26

\*/

bytes4 private constant \_INTERFACE\_ID\_ERC1155 = 0xd9b67a26;

/\*

\*     bytes4(keccak256('uri(uint256)')) == 0x0e89341c

\*/

bytes4 private constant \_INTERFACE\_ID\_ERC1155\_METADATA\_URI = 0x0e89341c;

/\*\*

\* @dev See {\_setURI}.

\*/

constructor (string memory uri) public {

\_setURI(uri);

// register the supported interfaces to conform to ERC1155 via ERC165

\_registerInterface(\_INTERFACE\_ID\_ERC1155);

// register the supported interfaces to conform to ERC1155MetadataURI via ERC165

\_registerInterface(\_INTERFACE\_ID\_ERC1155\_METADATA\_URI);

}

/\*\*

\* @dev See {IERC1155MetadataURI-uri}.

\*

\* This implementation returns the same URI for \*all\* token types.It relies

\* on the token type ID substitution mechanism

\* https://eips.ethereum.org/EIPS/eip-1155#metadata[defined in the EIP].

\*

\* Clients calling this function must replace the `\{id\}` substring with the

\* actual token type ID.

\*/

function uri(uint256 id) external view virtual override returns (string memory) {

return \_tokenURI(id);

}

/\*\*

\* @dev See {IERC1155-balanceOf}.

\*

\* Requirements:

\*

\* - `account` cannot be the zero address.

\*/

function balanceOf(address account, uint256 id) public view override returns (uint256) {

require(account != address(0), "ERC1155: balance query for the zero address");

return \_balances[id][account];

}

/\*\*

\* @dev See {IERC1155-balanceOfBatch}.

\*

\* Requirements:

\*

\* - `accounts` and `ids` must have the same length.

\*/

function balanceOfBatch(

address[] memory accounts,

uint256[] memory ids

)

public

view

override

returns (uint256[] memory)

{

require(accounts.length == ids.length, "ERC1155: accounts and ids length mismatch");

uint256[] memory batchBalances = new uint256[](accounts.length);

for (uint256 i = 0; i < accounts.length; ++i) {

require(accounts[i] != address(0), "ERC1155: batch balance query for the zero address");

batchBalances[i] = \_balances[ids[i]][accounts[i]];

}

return batchBalances;

}

/\*\*

\* @dev See {IERC1155-setApprovalForAll}.

\*/

function setApprovalForAll(address operator, bool approved) public virtual override {

require(\_msgSender() != operator, "ERC1155: setting approval status for self");

\_operatorApprovals[\_msgSender()][operator] = approved;

emit ApprovalForAll(\_msgSender(), operator, approved);

}

/\*\*

\* @dev See {IERC1155-isApprovedForAll}.

\*/

function isApprovedForAll(address account, address operator) public view override returns (bool) {

return \_operatorApprovals[account][operator];

}

/\*\*

\* @dev See {IERC1155-safeTransferFrom}.

\*/

function safeTransferFrom(

address from,

address to,

uint256 id,

uint256 amount,

bytes memory data

)

public

virtual

override

{

require(to != address(0), "ERC1155: transfer to the zero address");

require(

from == \_msgSender() || isApprovedForAll(from, \_msgSender()),

"ERC1155: caller is not owner nor approved"

);

address operator = \_msgSender();

\_beforeTokenTransfer(operator, from, to, \_asSingletonArray(id), \_asSingletonArray(amount), data);

\_balances[id][from] = \_balances[id][from].sub(amount, "ERC1155: insufficient balance for transfer");

\_balances[id][to] = \_balances[id][to].add(amount);

emit TransferSingle(operator, from, to, id, amount);

\_doSafeTransferAcc SMP anceCheck(operator, from, to, id, amount, data);

}

/\*\*

\* @dev See {IERC1155-safeBatchTransferFrom}.

\*/

function safeBatchTransferFrom(

address from,

address to,

uint256[] memory ids,

uint256[] memory amounts,

bytes memory data

)

public

virtual

override

{

require(ids.length == amounts.length, "ERC1155: ids and amounts length mismatch");

require(to != address(0), "ERC1155: transfer to the zero address");

require(

from == \_msgSender() || isApprovedForAll(from, \_msgSender()),

"ERC1155: transfer caller is not owner nor approved"

);

address operator = \_msgSender();

\_beforeTokenTransfer(operator, from, to, ids, amounts, data);

for (uint256 i = 0; i < ids.length; ++i) {

uint256 id = ids[i];

uint256 amount = amounts[i];

\_balances[id][from] = \_balances[id][from].sub(

amount,

"ERC1155: insufficient balance for transfer"

);

\_balances[id][to] = \_balances[id][to].add(amount);

}

emit TransferBatch(operator, from, to, ids, amounts);

\_doSafeBatchTransferAcc SMP anceCheck(operator, from, to, ids, amounts, data);

}

/\*\*

\* @dev Sets a new URI for all token types, by relying on the token type ID

\* substitution mechanism

\* https://eips.ethereum.org/EIPS/eip-1155#metadata[defined in the EIP].

\*

\* By this mechanism, any occurrence of the `\{id\}` substring in either the

\* URI or any of the amounts in the JSON file at said URI will be replaced by

\* clients with the token type ID.

\*

\* For example, the `https://token-cdn-domain/\{id\}.json` URI would be

\* interpreted by clients as

\* `https://token-cdn-domain/000000000000000000000000000000000000000000000000000000000004cce0.json`

\* for token type ID 0x4cce0.

\*

\* See {uri}.

\*

\* Because these URIs cannot be meaningfully represented by the {URI} event,

\* this function emits no events.

\*/

function \_setURI(string memory newuri) internal virtual {

\_uri = newuri;

}

/\*\*

\* @dev Creates `amount` tokens of token type `id`, and assigns them to `account`.

\*

\* Emits a {TransferSingle} event.

\*

\* Requirements:

\*

\* - `account` cannot be the zero address.

\* - If `to` refers to a smart contract, it must implement {IERC1155Receiver-onERC1155Received} and return the

\* acc SMP ance magic value.

\*/

function \_mint(address account, uint256 tokenId, uint256 amount, string memory tokenUri, bytes memory data) internal virtual {

require(account != address(0), "ERC1155: mint to the zero address");

address operator = \_msgSender();

\_beforeTokenTransfer(operator, address(0), account, \_asSingletonArray(tokenId), \_asSingletonArray(amount), data);

\_balances[tokenId][account] = \_balances[tokenId][account].add(amount);

\_setTokenURI(tokenId, tokenUri);

emit TransferSingle(operator, address(0), account, tokenId, amount);

\_doSafeTransferAcc SMP anceCheck(operator, address(0), account, tokenId, amount, data);

}

/\*\*

\* @dev Internal function to set the token URI for a given token.

\* Reverts if the token ID does not exist.

\* @param tokenId uint256 ID of the token to set its URI

\* @param tokenUri string URI to assign

\*/

function \_setURI( uint256 tokenId,  string memory tokenUri) internal virtual {

\_setTokenURI(tokenId, tokenUri);

}

function \_setTokenURI(uint256 tokenId, string memory tokenUri) internal override {

super.\_setTokenURI(tokenId, tokenUri);

}

/\*\*

\* @dev xref:ROOT:erc1155.adoc#batch-operations[Batched] version of {\_mint}.

\*

\* Requirements:

\*

\* - `ids` and `amounts` must have the same length.

\* - If `to` refers to a smart contract, it must implement {IERC1155Receiver-onERC1155BatchReceived} and return the

\* acc SMP ance magic value.

\*/

function \_mintBatch(address to, uint256[] memory ids, uint256[] memory amounts, bytes memory data) internal virtual {

require(to != address(0), "ERC1155: mint to the zero address");

require(ids.length == amounts.length, "ERC1155: ids and amounts length mismatch");

address operator = \_msgSender();

\_beforeTokenTransfer(operator, address(0), to, ids, amounts, data);

for (uint i = 0; i < ids.length; i++) {

\_balances[ids[i]][to] = amounts[i].add(\_balances[ids[i]][to]);

}

emit TransferBatch(operator, address(0), to, ids, amounts);

\_doSafeBatchTransferAcc SMP anceCheck(operator, address(0), to, ids, amounts, data);

}

/\*\*

\* @dev Destroys `amount` tokens of token type `id` from `account`

\*

\* Requirements:

\*

\* - `account` cannot be the zero address.

\* - `account` must have at least `amount` tokens of token type `id`.

\*/

function \_burn(address account, uint256 id, uint256 amount) internal virtual {

require(account != address(0), "ERC1155: burn from the zero address");

address operator = \_msgSender();

\_beforeTokenTransfer(operator, account, address(0), \_asSingletonArray(id), \_asSingletonArray(amount), "");

\_balances[id][account] = \_balances[id][account].sub(

amount,

"ERC1155: burn amount exceeds balance"

);

emit TransferSingle(operator, account, address(0), id, amount);

}

/\*\*

\* @dev xref:ROOT:erc1155.adoc#batch-operations[Batched] version of {\_burn}.

\*

\* Requirements:

\*

\* - `ids` and `amounts` must have the same length.

\*/

function \_burnBatch(address account, uint256[] memory ids, uint256[] memory amounts) internal virtual {

require(account != address(0), "ERC1155: burn from the zero address");

require(ids.length == amounts.length, "ERC1155: ids and amounts length mismatch");

address operator = \_msgSender();

\_beforeTokenTransfer(operator, account, address(0), ids, amounts, "");

for (uint i = 0; i < ids.length; i++) {

\_balances[ids[i]][account] = \_balances[ids[i]][account].sub(

amounts[i],

"ERC1155: burn amount exceeds balance"

);

}

emit TransferBatch(operator, account, address(0), ids, amounts);

}

/\*\*

\* @dev Hook that is called before any token transfer.This includes minting

\* and burning, as well as batched variants.

\*

\* The same hook is called on both single and batched variants.For single

\* transfers, the length of the `id` and `amount` arrays will be 1.

\*

\* Calling conditions (for each `id` and `amount` pair):

\*

\* - When `from` and `to` are both non-zero, `amount` of ``from``'s tokens

\* of token type `id` will be  transferred to `to`.

\* - When `from` is zero, `amount` tokens of token type `id` will be minted

\* for `to`.

\* - when `to` is zero, `amount` of ``from``'s tokens of token type `id`

\* will be burned.

\* - `from` and `to` are never both zero.

\* - `ids` and `amounts` have the same, non-zero length.

\*

\* To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].

\*/

function \_beforeTokenTransfer(

address operator,

address from,

address to,

uint256[] memory ids,

uint256[] memory amounts,

bytes memory data

)

internal virtual

{ }

function \_doSafeTransferAcc SMP anceCheck(

address operator,

address from,

address to,

uint256 id,

uint256 amount,

bytes memory data

)

private

{

if (to.isContract()) {

try IERC1155Receiver(to).onERC1155Received(operator, from, id, amount, data) returns (bytes4 response) {

if (response != IERC1155Receiver(to).onERC1155Received.selector) {

revert("ERC1155: ERC1155Receiver rejected tokens");

}

} catch Error(string memory reason) {

revert(reason);

} catch {

revert("ERC1155: transfer to non ERC1155Receiver implementer");

}

}

}

function \_doSafeBatchTransferAcc SMP anceCheck(

address operator,

address from,

address to,

uint256[] memory ids,

uint256[] memory amounts,

bytes memory data

)

private

{

if (to.isContract()) {

try IERC1155Receiver(to).onERC1155BatchReceived(operator, from, ids, amounts, data) returns (bytes4 response) {

if (response != IERC1155Receiver(to).onERC1155BatchReceived.selector) {

revert("ERC1155: ERC1155Receiver rejected tokens");

}

} catch Error(string memory reason) {

revert(reason);

} catch {

revert("ERC1155: transfer to non ERC1155Receiver implementer");

}

}

}

function \_asSingletonArray(uint256 element) private pure returns (uint256[] memory) {

uint256[] memory array = new uint256[](1);

array[0] = element;

return array;

}

}

pragma solidity ^0.6.2;

contract NFTTokens is ERC1155 {

address public governance;

uint256 public nftCount;

modifier onlyGovernance() {

require(msg.sender == governance, "only governance can call this");

\_;

}

constructor(address governance\_) public ERC1155("MateXgame") {

governance = governance\_;

nftCount = 0;

}

function addNewcard(uint256 initialSupply,string calldata \_tokenUri) external onlyGovernance {

nftCount++;

uint256 nftTokenClassId = nftCount;

\_mint(msg.sender, nftTokenClassId, initialSupply, \_tokenUri, "");

}

function getMetaverseCount() public view  returns(uint256) {

return nftCount;

}

function setURI( uint256 tokenId,string calldata tokenUri) external onlyGovernance {

\_setURI(tokenId, tokenUri);

}

}

## 0x6 Token Contract code

/\*\*

\*Submitted for verification at https://etherscan.io/ on 2022-03-25

\*/

pragma solidity 0.5.17;

interface IERC20 {

/\*\*

\* @dev Returns the amount of tokens in existence.

\*/

function totalSupply() external view returns (uint256);

/\*\*

\* @dev Returns the token decimals.

\*/

function decimals() external view returns (uint8);

/\*\*

\* @dev Returns the token symbol.

\*/

function symbol() external view returns (string memory);

/\*\*

\* @dev Returns the token name.

\*/

function name() external view returns (string memory);

/\*\*

\* @dev Returns the bep token owner.

\*/

function getOwner() external view returns (address);

/\*\*

\* @dev Returns the amount of tokens owned by `account`.

\*/

function balanceOf(address account) external view returns (uint256);

/\*\*

\* @dev Moves `amount` tokens from the caller's account to `recipient`.

\* Returns a boolean value indicating whether the operation succeeded.

\* Emits a {Transfer} event.

\*/

function transfer(address recipient, uint256 amount) external returns (bool);

/\*\*

\* @dev Returns the remaining number of tokens that `spender` will be

\* allowed to spend on behalf of `owner` through {transferFrom}.This is

\* zero by default.

\* This value changes when {approve} or {transferFrom} are called.

\*/

function allowance(address \_owner, address spender) external view returns (uint256);

/\*\*

\* @dev Sets `amount` as the allowance of `spender` over the caller's tokens.

\*

\* Returns a boolean value indicating whether the operation succeeded.

\* IMPORTANT: Beware that changing an allowance with this method brings the risk

\* that someone may use both the old and the new allowance by unfortunate

\* transaction ordering.One possible solution to mitigate this race

\* condition is to first reduce the spender's allowance to 0 and set the

\* desired value afterwards:

https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729

\* Emits an {Approval} event.

\*/

function approve(address spender, uint256 amount) external returns (bool);

/\*\*

\* @dev Moves `amount` tokens from `sender` to `recipient` using the

\* allowance mechanism.`amount` is then deducted from the caller's

\* allowance.

\* Returns a boolean value indicating whether the operation succeeded.

\* Emits a {Transfer} event.

\*/

function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

/\*\*

\* @dev Emitted when `value` tokens are moved from one account (`from`) to

\* another (`to`).

\* Note that `value` may be zero.

\*/

event Transfer(address indexed from, address indexed to, uint256 value);

/\*\*

\* @dev Emitted when the allowance of a `spender` for an `owner` is set by

\* a call to {approve}.`value` is the new allowance.

\*/

event Approval(address indexed owner, address indexed spender, uint256 value);

}

/\*

\* @dev Provides information about the current execution context, including the

\* sender of the transaction and its data.While these are generally available

\* via msg.sender and msg.data, they should not be accessed in such a direct

\* manner, since when dealing with GSN meta-transactions the account sending and

\* paying for execution may not be the actual sender (as far as an application

\* is concerned).

\* This contract is only required for intermediate, library-like contracts.

\*/

contract Context {

// Empty internal constructor, to prevent people from mistakenly deploying

// an instance of this contract, which should be used via inheritance.

constructor () internal { }

function \_msgSender() internal view returns (address payable) {

return msg.sender;

}

function \_msgData() internal view returns (bytes memory) {

this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691

return msg.data;

}

}

/\*\*

\* @dev Wrappers over Solidity's arithmetic operations with added overflow

\* checks.

\* Arithmetic operations in Solidity wrap on overflow.This can easily result

\* in bugs, because programmers usually assume that an overflow raises an

\* error, which is the standard behavior in high level programming languages.

\* `SafeMath` restores this intuition by reverting the transaction when an

\* operation overflows.

\* Using this library instead of the unchecked operations eliminates an entire

\* class of bugs, so it's recommended to use it always.

\*/

library SafeMath {

/\*\*

\* @dev Returns the addition of two unsigned integers, reverting on

\* overflow.

\* Counterpart to Solidity's `+` operator.

\* Requirements:

\* - Addition cannot overflow.

\*/

function add(uint256 a, uint256 b) internal pure returns (uint256) {

uint256 c = a + b;

require(c >= a, "SafeMath: addition overflow");

return c;

}

/\*\*

\* @dev Returns the subtraction of two unsigned integers, reverting on

\* overflow (when the result is negative).

\* Counterpart to Solidity's `-` operator.

\* Requirements:

\* - Subtraction cannot overflow.

\*/

function sub(uint256 a, uint256 b) internal pure returns (uint256) {

return sub(a, b, "SafeMath: subtraction overflow");

}

/\*\*

\* @dev Returns the subtraction of two unsigned integers, reverting with custom message on

\* overflow (when the result is negative).

\* Counterpart to Solidity's `-` operator.

\* Requirements:

\* - Subtraction cannot overflow.

\*/

function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b <= a, errorMessage);

uint256 c = a - b;

return c;

}

/\*\*

\* @dev Returns the multiplication of two unsigned integers, reverting on

\* overflow.

\* Counterpart to Solidity's `\*` operator.

\* Requirements:

\* - Multiplication cannot overflow.

\*/

function mul(uint256 a, uint256 b) internal pure returns (uint256) {

// Gas optimization: this is cheaper than requiring 'a' not being zero, but the

// benefit is lost if 'b' is also tested.

// See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522

if (a == 0) {

return 0;

}

uint256 c = a \* b;

require(c / a == b, "SafeMath: multiplication overflow");

return c;

}

/\*\*

\* @dev Returns the integer division of two unsigned integers.Reverts on

\* division by zero.The result is rounded towards zero.

\* Counterpart to Solidity's `/` operator.Note: this function uses a

\* `revert` opcode (which leaves remaining gas untouched) while Solidity

\* uses an invalid opcode to revert (consuming all remaining gas).

\* Requirements:

\* - The divisor cannot be zero.

\*/

function div(uint256 a, uint256 b) internal pure returns (uint256) {

return div(a, b, "SafeMath: division by zero");

}

/\*\*

\* @dev Returns the integer division of two unsigned integers.Reverts with custom message on

\* division by zero.The result is rounded towards zero.

\*

\* Counterpart to Solidity's `/` operator.Note: this function uses a

\* `revert` opcode (which leaves remaining gas untouched) while Solidity

\* uses an invalid opcode to revert (consuming all remaining gas).

\*

\* Requirements:

\* - The divisor cannot be zero.

\*/

function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

// Solidity only automatically asserts when dividing by 0

require(b > 0, errorMessage);

uint256 c = a / b;

// assert(a == b \* c + a % b); // There is no case in which this doesn't hold

return c;

}

/\*\*

\* @dev Returns the remainder of dividing two unsigned integers.(unsigned integer modulo),

\* Reverts when dividing by zero.

\* Counterpart to Solidity's `%` operator.This function uses a `revert`

\* opcode (which leaves remaining gas untouched) while Solidity uses an

\* invalid opcode to revert (consuming all remaining gas).

\* Requirements:

\* - The divisor cannot be zero.

\*/

function mod(uint256 a, uint256 b) internal pure returns (uint256) {

return mod(a, b, "SafeMath: modulo by zero");

}

/\*\*

\* @dev Returns the remainder of dividing two unsigned integers.(unsigned integer modulo),

\* Reverts with custom message when dividing by zero.

\* Counterpart to Solidity's `%` operator.This function uses a `revert`

\* opcode (which leaves remaining gas untouched) while Solidity uses an

\* invalid opcode to revert (consuming all remaining gas).

\* Requirements:

\* - The divisor cannot be zero.

\*/

function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b != 0, errorMessage);

return a % b;

}

}

/\*\*

\* @dev Contract module which provides a basic access control mechanism, where

\* there is an account (an owner) that can be granted exclusive access to

\* specific functions.

\* By default, the owner account will be the one that deploys the contract.This

\* can later be changed with {transferOwnership}.

\* This module is used through inheritance.It will make available the modifier

\* `onlyOwner`, which can be applied to your functions to restrict their use to

\* the owner.

\*/

contract Ownable is Context {

address private \_owner;

event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

/\*\*

\* @dev Initializes the contract setting the deployer as the initial owner.

\*/

constructor () internal {

address msgSender = \_msgSender();

\_owner = msgSender;

emit OwnershipTransferred(address(0), msgSender);

}

/\*\*

\* @dev Returns the address of the current owner.

\*/

function owner() public view returns (address) {

return \_owner;

}

/\*\*

\* @dev Throws if called by any account other than the owner.

\*/

modifier onlyOwner() {

require(\_owner == \_msgSender(), "Ownable: caller is not the owner");

\_;

}

/\*\*

\* @dev Leaves the contract without owner.It will not be possible to call

\* `onlyOwner` functions anymore.Can only be called by the current owner.

\* NOTE: Renouncing ownership will leave the contract without an owner,

\* thereby removing any functionality that is only available to the owner.

\*/

function renounceOwnership() public onlyOwner {

emit OwnershipTransferred(\_owner, address(0));

\_owner = address(0);

}

/\*\*

\* @dev Transfers ownership of the contract to a new account (`newOwner`).

\* Can only be called by the current owner.

\*/

function transferOwnership(address newOwner) public onlyOwner {

\_transferOwnership(newOwner);

}

/\*\*

\* @dev Transfers ownership of the contract to a new account (`newOwner`).

\*/

function \_transferOwnership(address newOwner) internal {

require(newOwner != address(0), "Ownable: new owner is the zero address");

emit OwnershipTransferred(\_owner, newOwner);

\_owner = newOwner;

}

}

contract ERC20SMP is Context, IERC20, Ownable {

using SafeMath for uint256;

mapping (address => uint256) private \_balances;

mapping (address => mapping (address => uint256)) private \_allowances;

uint256 private \_totalSupply;

uint8 public \_decimals;

string public \_symbol;

string public \_name;

constructor() public {

\_name = "SMP ";

\_symbol = "SMP ";

\_decimals = 6;

\_totalSupply = 210000000 \* 1e 6;

\_balances[msg.sender] = \_totalSupply;

emit Transfer(address(0), msg.sender, \_totalSupply);

}

/\*\*

\* @dev Returns the bep token owner.

\*/

function getOwner() external view returns (address) {

return owner();

}

/\*\*

\* @dev Returns the token decimals.

\*/

function decimals() external view returns (uint8) {

return \_decimals;

}

/\*\*

\* @dev Returns the token symbol.

\*/

function symbol() external view returns (string memory) {

return \_symbol;

}

/\*\*

\* @dev Returns the token name.

\*/

function name() external view returns (string memory) {

return \_name;

}

/\*\*

\* @dev See {ERC20-totalSupply}.

\*/

function totalSupply() external view returns (uint256) {

return \_totalSupply;

}

/\*\*

\* @dev See {ERC20-balanceOf}.

\*/

function balanceOf(address account) external view returns (uint256) {

return \_balances[account];

}

/\*\*

\* @dev See {ERC20-transfer}.

\* Requirements:

\* - `recipient` cannot be the zero address.

\* - the caller must have a balance of at least `amount`.

\*/

function transfer(address recipient, uint256 amount) external returns (bool) {

\_transfer(\_msgSender(), recipient, amount);

return true;

}

/\*\*

\* @dev See {ERC20-allowance}.

\*/

function allowance(address owner, address spender) external view returns (uint256) {

return \_allowances[owner][spender];

}

/\*\*

\* @dev See {ERC20-approve}.

\*

\* Requirements:

\* - `spender` cannot be the zero address.

\*/

function approve(address spender, uint256 amount) external returns (bool) {

\_approve(\_msgSender(), spender, amount);

return true;

}

/\*\*

\* @dev See {ERC20-transferFrom}.

\*

\* Emits an {Approval} event indicating the updated allowance.This is not

\* required by the EIP.See the note at the beginning of {ERC20};

\* Requirements:

\* - `sender` and `recipient` cannot be the zero address.

\* - `sender` must have a balance of at least `amount`.

\* - the caller must have allowance for `sender`'s tokens of at least

\* `amount`.

\*/

function transferFrom(address sender, address recipient, uint256 amount) external returns (bool) {

\_transfer(sender, recipient, amount);

\_approve(sender, \_msgSender(), \_allowances[sender][\_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));

return true;

}

/\*\*

\* @dev Atomically increases the allowance granted to `spender` by the caller.

\*

\* This is an alternative to {approve} that can be used as a mitigation for

\* problems described in {ERC20-approve}.

\*

\* Emits an {Approval} event indicating the updated allowance.

\*

\* Requirements:

\*

\* - `spender` cannot be the zero address.

\*/

function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {

\_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].add(addedValue));

return true;

}

function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {

\_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));

return true;

}

function \_transfer(address sender, address recipient, uint256 amount) internal {

require(sender != address(0), "ERC20: transfer from the zero address");

require(recipient != address(0), "ERC20: transfer to the zero address");

\_balances[sender] = \_balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");

\_balances[recipient] = \_balances[recipient].add(amount);

emit Transfer(sender, recipient, amount);

}

/\*\*

\* @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.

\*

\* This is internal function is equivalent to `approve`, and can be used to

\* e.g.set automatic allowances for certain subsystems, etc.

\*

\* Emits an {Approval} event.

\*

\* Requirements:

\*

\* - `owner` cannot be the zero address.

\* - `spender` cannot be the zero address.

\*/

function \_approve(address owner, address spender, uint256 amount) internal {

require(owner != address(0), "ERC20: approve from the zero address");

require(spender != address(0), "ERC20: approve to the zero address");

\_allowances[owner][spender] = amount;

emit Approval(owner, spender, amount);

}

/\*\*

\* @dev Destroys `amount` tokens from `account`.`amount` is then deducted

\* from the caller's allowance.

\*

\* See {\_burn} and {\_approve}.

}

## 0x7 Reference

0x71.Fabian Vogelsteller and Vitalik Buterin.Nov.2015.

<https://eips.ethereum.org/EIPS/eip-20>

0x72.A Hands-On Tutorial for Zero-Knowledge Proofs.

<https://en.wikipedia.org/wiki/Zero-knowledge_proof>

0x73.ERC-20 (Ethereum Consultation 20) was proposed by Fabian Vogelsteller in November 2015.

<https://ethereum.org/zh/developers/docs/standards/tokens/erc-20/>

0x74. "How to Create an NFT – Simply Explained".Eduwab.May 19, 2022. Retrieved June 3, 2022.

0x75.The Rise of Non-Fungible Tokens : Exploring the Future of Digital Collectibles" by Chris Burns.

0x76.Non-Fungible Tokens: A Survey" by C.Szegedi et al.