Cats Metaverse

0x0000001 Introduce

1、smartBCH Cats introduce

smartBCH

Started in December 2020, smartBCH is a sidechain for Bitcoin Cash and has an aim to explore new ideas and unlock novel possibilities.By developing optimized, high-throughput and hardware-friendly libraries compatible with the de facto standards of smart contracts, DeFi applications can be easily migrated into Bitcoin Cash's ecosystem and run fluently at low cost.In March 2021, smartBCH raised 1000 BCH from the Bitcoin Cash community with a Flipstarter campaign.The funds will be used for product and business development.

cats

Hey there pretty PussyCat, nice to meet you!Let me tell you a little story about $CATS & Bitcoin Cash.

Once upon a time, there was a cryptocurrency called Bitcoin Cash..The Bitcoin Cash community grew bigger and stronger every day.People were building all kinds of cool things and applications.But then one day a team of developers thought of a cool idea to create a new side chain.A new side-chain where Bitcoin Cash enthusiast could run EVM DApps, And so SmartBCH was born!

2、Lack of games

There's not a complete, decentralized blockchain game on the smartBCH network.

3. Coping strategies

The SKY team will jointly join the cats community to deploy the decentralized game Cats Metaverse project in the smartBCH network, provide decentralized games to the smartBCH public chain, and help in the construction of the smartBCH ecology.

0x0000002 Cats Metaverse

Project overview

Cats Metaverse is a pvp blockchain game combined with NFT.

In the world of the metaverse, every life needs energy crystals to maintain, and in order to survive, there is no peace here, only enemies.Players will play the adventurer and the other players in the Mettaverse.You will play a cold-blooded interstellar killer who can do to capture other adventuurers and dominate the meta-universe.Fight for the Mettaverse with other players.You will play a cold-blooded interstellar killer who can do is to grab other adventurers and dominate the meta-universe.

Game play instructions

PVP pattern:

energy system:

Entering the metaverse, each adventurer is sheltered by stellar energy, adding different energy crystals depending on the player.

1. Combat status: This mode can attack any other game, and the player increases the most energy per unit of time.Get energy crystals = (Attack force / 80,000) * min.

2. Defense state: This mode can defend against other people's attacks.The average amount of energy the player takes per unit of time.Get energy crystals = (Defence / 160,000) * min.

3. Protection state: This state can be protected by the stars, and the player cannot be attacked.The player becomes protected when attacked by other players.Get energy crystals = (Defence / 160,000) * min.

Charging system:

Charging the corresponding energy crystal, you can obtain the corresponding attack force defense force.

Combat System (PVP):

The adventurers in the metaverse can survive, combat, and peace.

1. Combat state: It can attack people and be easy to be attacked by others.When you attack more than the other side and the other person is not in a protective state, you can attack the other side and steal the energy crystals (the attacker gets 80% of the crystals and 20% into the stellar energy system).

2. Defense status: Defense force * 2, unable to attack anyone.But when others attack, they need to be twice as defensive.

3. Protection status: Unable cannot attack anyone, neither can attack you.

NFT system:

1. Users can buy NFT equipment.

2. The NFT equipment of metauniverse Hero is limited edition, adding three attributes.Attack defense increased by percentage, and attack cooltime was reduced by N seconds.

recovery system:

User's attack and defense power can be converted into energy crystals for recycling.Users gain 80% and 20% into the stellar energy system.

Energy crystal: It can be extracted to the user address at any time.

Recovery rule: It can not be recovered during combat, protection cooling time.

0x0000003 The economic model

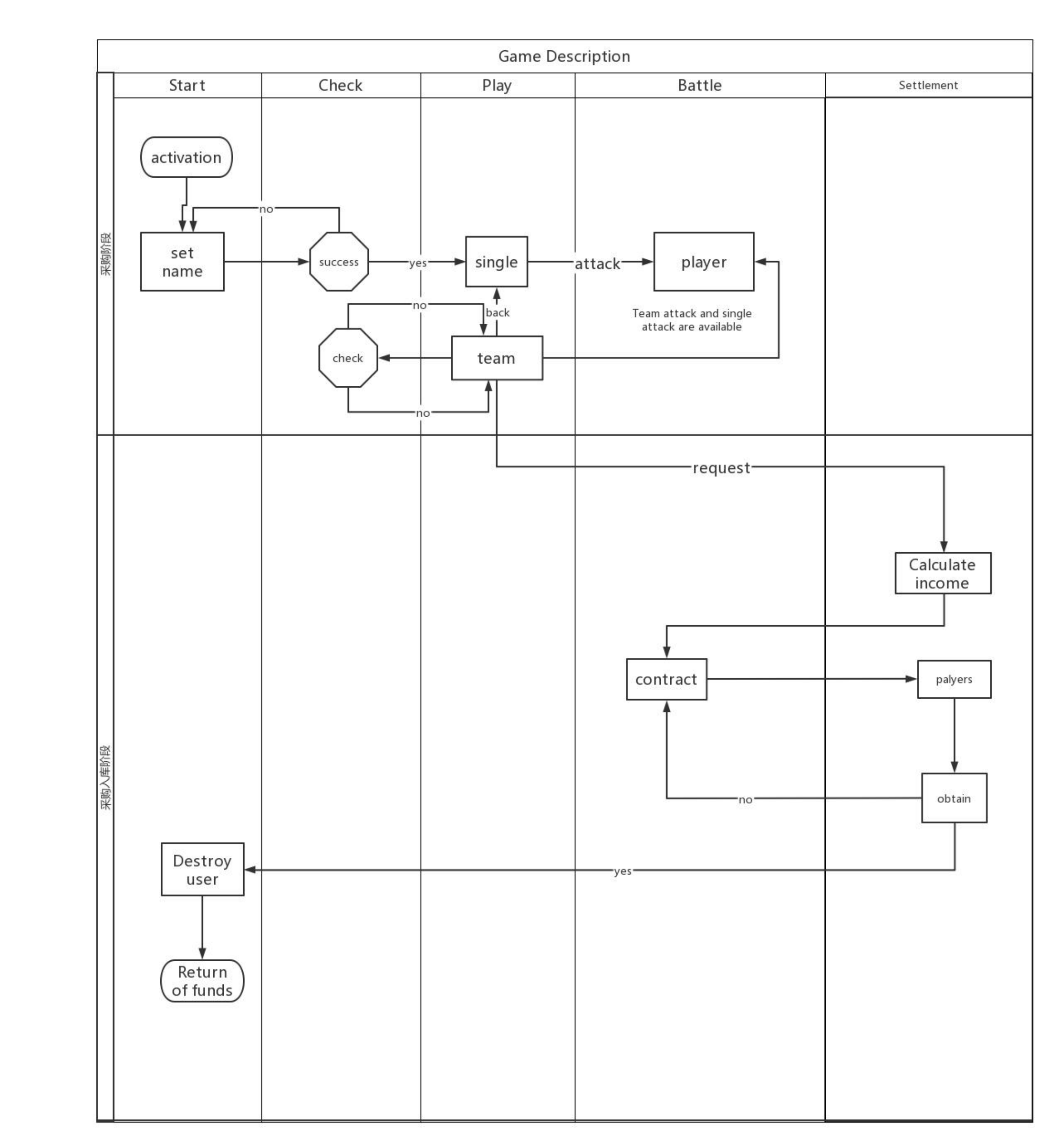
Re-purchase strategy:

1、Half of the pre-sale NFT amount is used to repurchase the cats token for game rewards.

2、The game has a halving mechanism.Halve the production in four months.Just like Bitcoin.Annual half to monthly half.

3. The NFT is all in a limited edition.

Cats Metaverse design principle



0x0000004

NFT smart contract

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

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

/**
* @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
* acceptance 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
* acceptance 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 ^0.6.0;


/**
* _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 accept 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 accept 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 ^0.6.0;

/*
* @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 ^0.6.0;


/**
* @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 ^0.6.0;

/**
* @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 ^0.6.0;


/**
*
* @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);

_doSafeTransferAcceptanceCheck(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);

_doSafeBatchTransferAcceptanceCheck(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
* acceptance 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);

_doSafeTransferAcceptanceCheck(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 _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
* acceptance 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);

_doSafeBatchTransferAcceptanceCheck(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 _doSafeTransferAcceptanceCheck(
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 _doSafeBatchTransferAcceptanceCheck(
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 ^0.6.0;

contract NFTTokens is ERC1155 {
address public governance;
uint256 public brkCount;

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

_;
}

constructor(address governance_) public ERC1155("iloveyou") {
governance = governance_;
brkCount = 0;
}

function addNewcard(uint256 initialSupply,string calldata _tokenUri) external onlyGovernance {
brkCount++;
uint256 brkTokenClassId = brkCount;
_mint(msg.sender, brkTokenClassId, initialSupply, _tokenUri, "");
}
function getMetaverseCount() public view  returns(uint256) {
return brkCount;
}

}
```

Development process:

August 2021: Cats Metaverse contract development, the underlying contract interaction

November 2021: Conduct evaluation and performance test, and code modification

December 2021: Cats Metaverse technical verification

January 2022: Development and verification of built-in contracts.

February 2022: Test the network deployment, build a large-scale test network based on the community, and conduct all-round testing

March 2022: Main online line, expansion, and data transactions

April 2022: Cats Metaverse global community ecology was initially completed

May 2022: Game NFT Expansion Package DC Universe Complete.

August 2022: Improve the Cats Metaverse global ecological development tasks successively.

Team:Sky

Friday, December 10th, 2021