Date: November, 2 nd, 2021

1. SMART CONTRAC DETAILS:

Smart Contract: 0x7d10b6157c7c577caa62d319dc215209cf2db8c3

Symbol: EXO

Decimals: 18

Network: Binance Smart Chain (BSC)

Max Supply: 1.000.000.000 EXO

Token
EXO is the utility token of the Exohood and has the following use cases:
Governance: EXO token holders will vote on platform parameters to navigate the project's economics and developments
Incentives: EXO tokens are used to incentivize user participation in bounty hunting, referrals and other programs in the future

NFT Suite: EXO tokens are required for NFT minting, staking, auction, trading and more
System Tax: Users who transact with EXO tokens will contribute to a **dividend pool.**

1.1.-Token Specification

Name: Exohood

Symbol: EXO

Network: Binance Smart Chain (BSC)

Spec: BEP-20

Smart Contract Address BSC: 0x7d10b6157c7c577caa62d319dc215209cf2db8c3

Exohood token (EXO) is a BEP-20 token standard, native to the Binance Smart Chain (BSC). EXO offers in the future great utility within the Exohood ecosystem, including representing governance rights in votes on updates or proposals to the protocol. Additionally, Exo tokens in the future we are used as liquidity rewards. The Exohood share of these supply is proportional to the amount of liquidity provided to the protocol a maximum cap of 120 EXO per block and a minimum supply of 1 EXO per block is in place.

EXO is the native utility token that is used for:
Governance: EXO token holders will vote on platform parameters to navigate the project's economics and developments.

- Incentives: EXO tokens are used to incentivize user participation in bounty hunting, referrals and other programs in the future.
- NFT Suite: EXO tokens are required for NFT minting, staking, auction, trading and more.
- System Tax: Users who transact with EXO tokens will contribute to a dividend pool.
- Decentralized Exchange: Exohood building a fully decentralized exchange and powered by the Community Governance.
- Liquidity: Allows users to earn rewards by providing liquidity using the EXO token.

1.2 Token Allocation

Total Supply (100%): 1,000,000,000 EXO

Token Name: EXO

Community: 60.00% of the total token supply

Team: 21.00% of the total token supply

Investors: 18.20% of the total token supply

Advisors: 0.80% of the total token supply

1.3.- Token Public Contract

```
contract Consts
uint public constant TOKEN_DECIMALS = 18;
uint8 public constant TOKEN_DECIMALS_UINT8 = 18;
uint public constant TOKEN_DECIMAL_MULTIPLIER = 10 ** TOKEN_DECIMALS;
string public constant TOKEN_NAME = "Exohood";
string public constant TOKEN_SYMBOL = "EXO";
bool public constant PAUSED = false;
address public constant TARGET_USER = 0xcA387DDAfF5B41C18320145788Ee5C2A53226e9e;
bool public constant CONTINUE_MINTING = true;
address[4] memory addresses = address(0x8778f4d4ea6ae699dc5f97ac318bdb3d9bbc20a400),address(0x600a2ccc7c8779af245dec3cae9d430ffac2749100),address(0x156dd746a820b7e4c6eab9e40ce3a4b2318c820100),address(0x0594d6ac776f46837f2ecc16cd6b8f4a616eed4b00)];
uint[4] memory amounts = [uint(8000000000000000000000000),uint(182000000000000000000000000),uint(210000000000000000000000000),uint(600000000000000000000000000)];
uint64[4] memory freezes = [uint64(0),uint64(0),uint64(0),uint64(0)];
Copied!
```

1.4.- Tokenomics Distribution

Community: 0x0594d6ac776f46837f2ecc16cd6b8f4a616eed4b

Team: 0x156dd746a820b7e4c6eab9e40ce3a4b2318c8201

Investors: 0x600a2ccc7c8779af245dec3cae9d430ffac27491

Advisors: 0x8778f4d4ea6ae699dc5f97ac318bdb3d9bbc20a4

Contract Mint Finished

- **Max Supply:** 1.000.000.000 EXO
- **View Source:** BscScan

1.5 Contract Mint Finished

Max Supply: 1.000.000.000 EXO

Transaction Details

https://bscscan.com/tx/0x00471731b7b47d35edd45c0ec115b6a0eb3d3b37a975b64b0220cd854d639a1b#eventlog

1.6.- Source Code

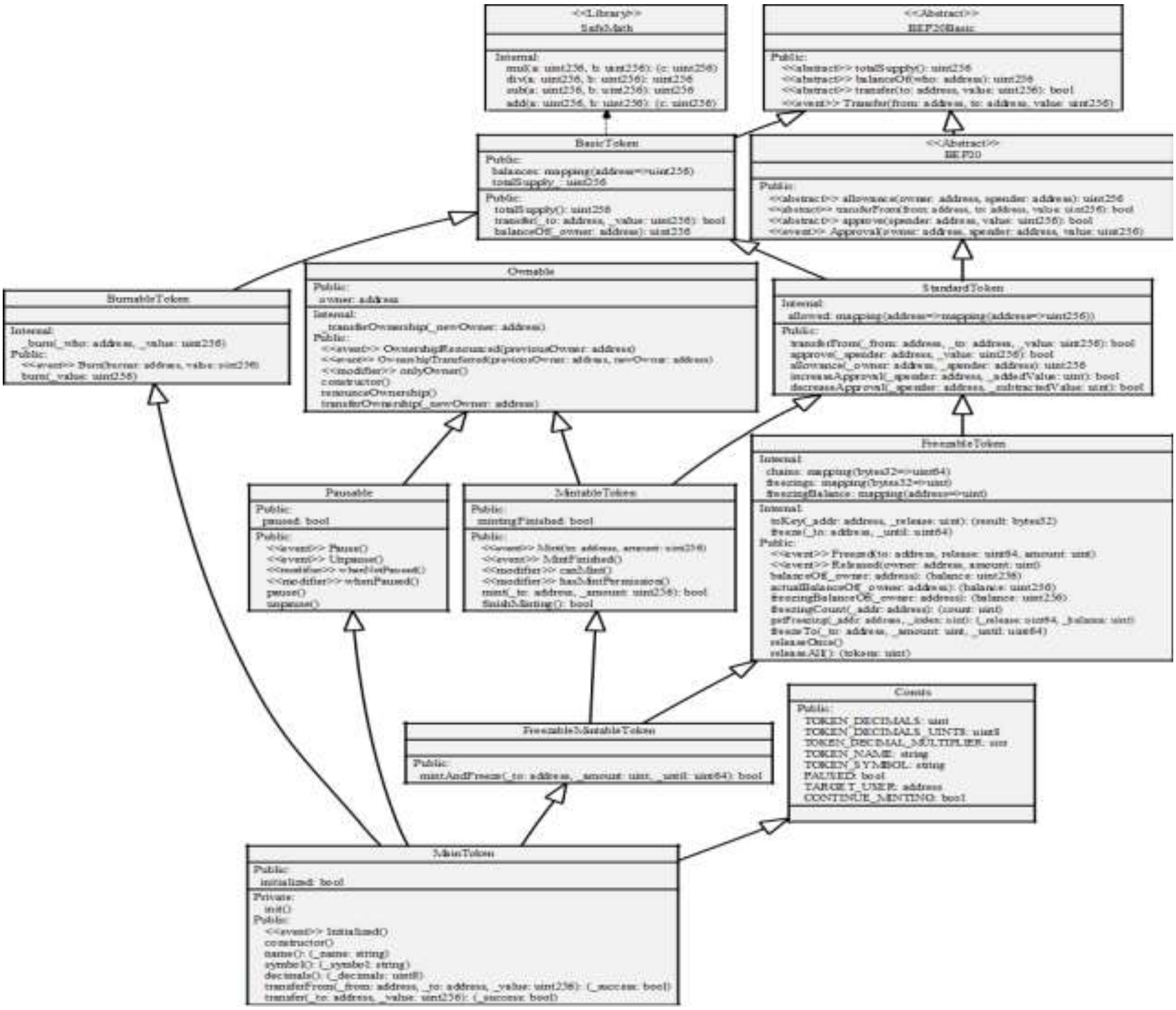
```
/**
 * @title Mintable token
 * @dev Simple BEP20 Token example, with mintable token creation
 * @dev Issue: * https://github.com/OpenZeppelin/openzeppelin-solidity/issues/120
 * Based on code by TokenMarketNet:
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
 */
contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = false;

  modifier canMint() {
    require(!mintingFinished);
    _;
  }
  modifier hasMintPermission() {
    require(msg.sender == owner);
    _;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
```

1.7 SIMILAR CONTRACTS CODE

RESULT: NO MATCH , THE PROGRAMMED CODE WAS PROGRAMMED FOR EXOHOOD AND IS NOT A COPY OF ANOTHER CODE.
SOURCE: https://bscscan.com/find-similar-contracts?a=0x7d10b6157c7c577caa62d319dc215209cf2db8c3

2.- TESTING

2.1.- Diagram for Contract


<<Library>>
SafeMath
Internal:
mul(a: uint256, b: uint256): (c: uint256)
div(a: uint256, b: uint256): uint256
sub(a: uint256, b: uint256): uint256
add(a: uint256, b: uint256): (c: uint256)
<<Abstract>>
BEP20Basic
Public:
<<abstract>> totalSupply(): uint256
<<abstract>> balanceOf(who: address): uint256
<<abstract>> transfer(to: address, value: uint256): bool
<<event>> Transfer(from: address, to: address, value: uint256)
BasicToken
Public:
balances: mapping(address=>uint256)
totalSupply_: uint256
Public:
totalSupply(): uint256
transfer(_to: address, _value: uint256): bool
balanceOf(_owner: address): uint256
<<Abstract>>
BEP20
Public:
<<abstract>> allowance(owner: address, spender: address): uint256
<<abstract>> transferFrom(from: address, to: address, value: uint256): bool
<<abstract>> approve(spender: address, value: uint256): bool
<<event>> Approval(owner: address, spender: address, value: uint256)
BurnableToken
Internal:
_burn(_who: address, _value: uint256)
Public:
<<event>> Burn(burner: address, value: uint256)
burn(_value: uint256)
Ownable
Public:
owner: address
Internal:
_transferOwnership(_newOwner: address)
Public:
<<event>> OwnershipRenounced(previousOwner: address)
<<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
<<modifier>> onlyOwner()
constructor()
renounceOwnership()
transferOwnership(_newOwner: address)
StandardToken
Internal:
allowed: mapping(address=>mapping(address=>uint256))
Public:
transferFrom(_from: address, _to: address, _value: uint256): bool
approve(_spender: address, _value: uint256): bool
allowance(_owner: address, _spender: address): uint256
increaseApproval(_spender: address, _addedValue: uint): bool
decreaseApproval(_spender: address, _subtractedValue: uint): bool
Pausable
Public:
paused: bool
Public:
<<event>> Pause()
<<event>> Unpause()
<<modifier>> whenNotPaused()
<<modifier>> whenPaused()
pause()
unpause()
MintableToken
Public:
mintingFinished: bool
Public:
<<event>> Mint(to: address, amount: uint256)
<<event>> MintFinished()
<<modifier>> canMint()
<<modifier>> hasMintPermission()
mint(_to: address, _amount: uint256): bool
finishMinting(): bool
FreezableToken
Internal:
chains: mapping(bytes32=>uint64)
freezings: mapping(bytes32=>uint)
freezingBalance: mapping(address=>uint)
Internal:
toKey(_addr: address, _release: uint): (result: bytes32)
freeze(_to: address, _until: uint64)
Public:
<<event>> Freezed(to: address, release: uint64, amount: uint)
<<event>> Released(owner: address, amount: uint)
balanceOf(_owner: address): (balance: uint256)
actualBalanceOf(_owner: address): (balance: uint256)
freezingBalanceOf(_owner: address): (balance: uint256)
freezingCount(_addr: address): (count: uint)
getFreezing(_addr: address, _index: uint): (_release: uint64, _balance: uint)
freezeTo(_to: address, _amount: uint, _until: uint64)
releaseOnce()
releaseAll(): (tokens: uint)
Consts
Public:
TOKEN_DECIMALS: uint
TOKEN_DECIMALS_UINT8: uint8
TOKEN_DECIMAL_MULTIPLIER: uint
TOKEN_NAME: string
TOKEN_SYMBOL: string
PAUSED: bool
TARGET_USER: address
CONTINUE_MINTING: bool
FreezableMintableToken
Public:
mintAndFreeze(_to: address, _amount: uint, _until: uint64): bool
MainToken
Public:
initialized: bool
Private:
init()
Public:
<<event>> Initialized()
constructor()
name(): (_name: string)
symbol(): (_symbol: string)
decimals(): (_decimals: uint8)
transferFrom(_from: address, _to: address, _value: uint256): (_success: bool)
transfer(_to: address, _value: uint256): (_success: bool)

1. **SUCCESS:** The check that advisor tokens are thawed all at once, and by 1/30 every month for 30 months, as reserveAndTeam tokens.

2. **SUCCESS** Performing the defrosting function several times in one month with subsequent verification showing that only the first defrost was triggered.

3. **SUCCESS** The Scenario distribution of tokens, their defrosting for 30 months. The Verification shows that all tokens are defrosted correctly: at the specified time and in the right amount.

4. **SUCCESS** Defrost the tokens for the ReserveAndTeam address, then transfer from this token address and again defrost. The check show that the number of defrosted tokens does not depend on the current balance on the defrosted address.

5. **IMPROVE NEEDS:** An attempt to distribute more than 1 billion tokens. An error should have occurred at some point. There were not errors, and tokens were distributed (Important.6)

6. **IMPROVE NEEDS:** The test checking that total Supply corresponds to the actual number of tokens issued before distribution (Important.4)

7. **IMPROVE NEEDS:** The attempt to execute batchAsssignTokens several times. The test that all tokens have been distributed (Important.10)

2.2.- CODE QUALITY

/** * @title Mintable token * @dev Simple BEP20 Token example, with mintable token creation * @dev Issue: * https://github.com/OpenZeppelin/openzeppelin-solidity/issues/120 * Based on code by TokenMarketNet: https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol */ contract MintableToken is StandardToken, Ownable { event Mint(address indexed to, uint256 amount); event MintFinished(); bool public mintingFinished = false; modifier canMint() { require(!mintingFinished); _; } modifier hasMintPermission() { require(msg.sender == owner); _; } /** * @dev Function to mint tokens * @param _to The address that will receive the minted tokens. * @param _amount The amount of tokens to mint. * @return A boolean that indicates if the operation was successful. */

1. There is no need to define functions that return public contract state variables

2. The constant modifier in functions is recommended to limit to pure/view.

3. It is not recommend using camelCase together with under_score in the name of a variable.

4. It is not recommend using getBlockTimestamp since Blockchain timestamp can be obtained by means of web3 without using a contract.

5. TestNaviToken.js: Tests inside it() must be independent of each other. If one of the tests is removed or replaced, the result of the automatic tests should not be changed.

3.- FINAL RESULT

1. **SUCCESS** The check of defrosting the advisor tokens.

2. **SUCCESS** The attempt to perform the defrost function several times in one month and check that only the first defrost has worked.

3. **SUCCESS** The distribution of tokens, their defrosting for 30 months. Check that all tokens are defrosted correctly: at the specified time and in the right amount.

4. **SUCCESS** The defrost of the tokens address, then their transfer from the address and re-defrost. The check that the number of defrosted tokens is independent of the current balance on the defrostable address.

5. **SUCCESS** Distribution of more than 1.000.000.000 tokens..

6. **SUCCESS** The check that totalSupply corresponds to the actual number of tokens produced before the distribution.

7. **SUCCESS** Try to execute batchAsssignTokens several times. Verify that all tokens have been distributed successfully.

8. **SUCCESS** Distribution of tokens after 7 months, when it is already possible to perform defrosting.

9. **SUCCESS** The check that even after 40 months it defrosts even after 40 months the correct number of tokens.

SUCCESS NO MORE TOKEN CAN BE ISSUED DUE TO EVENT 1.5 THAT THE MINT CONTRACT HAS BEEN TERMINATED SO THE NUMBER OF TOKEN FOR THE LIFE OF THE CONTRACT WILL BE 1,000,000

November, 2 nd, 2021

,

APPROVED

INTERCER