Cows Gone Mad (CGM Smart Contract) Functions Description For Audit Report

Configuration Settings:

- **Compiler : 0.8.17+commit.8df45f5f**
- **Language : Solidity**
- **EVM Version : Default**
- **Enable Optimization (200) : True**

1. Function: `mint(uint256 quantity)`

- Description: Allows users to mint NFTs during the public sale.
- Inputs:
  - `quantity`: Number of NFTs to mint.
- Relevant Information: Can only be called during the public sale phase (status == 4) and not during any whitelist phase.

2. Function: `crossmint(address _to, uint256 quantity)`

- Description: Allows cross-minting from another contract.
- Inputs:
  - `_to`: Address to mint NFTs for.
  - `quantity`: Number of NFTs to mint.
- Relevant Information: Can only be called by the `crossmintAddress`.

3. Function: `setCrossmintAddress(address _crossmintAddress)`

  - Description: Sets the cross-minting contract address.
  - Inputs:
    - `_crossmintAddress`: Address of the cross-minting contract.
  - Relevant Information: Can only be called by the contract owner.

4. Function: `setBaseURI(string memory baseURI)`

  - Description: Sets the base URI for the NFT metadata.
  - Inputs:
    - `baseURI`: The base URI string.
  - Relevant Information: Can only be called by the contract owner.

5. Function: `revealAllNFTs()`

  - Description: Reveals all NFTs after the sale is over.
  - Relevant Information: Can only be called by the contract owner.

6. Function: `withdrawMoney()`

  - Description: Withdraws the Ether balance from the contract.
  - Relevant Information: Can only be called by the contract owner and is `nonReentrant` to prevent reentrancy attacks.

7. Function: `changePhaseCollectionSize(uint256 _collection)`

  - Description: Changes the collection size for the current phase.
  - Inputs:
    - `_collection`: New collection size for the current phase.
  - Relevant Information: Can only be called by the contract owner.

8. Function: `changePublicMintPrice(uint256 _newPrice)`

  - Description: Changes the public mint price for NFTs.

- Inputs:
  - `_newPrice`: New price for minting NFTs during the public sale.
- Relevant Information: Can only be called by the contract owner.

9. Function: `changeMaxPerTransaction(uint256 q)`

- Description: Changes the maximum NFTs that can be minted per transaction.
- Inputs:
  - `q`: New maximum NFTs allowed per transaction.
- Relevant Information: Can only be called by the contract owner.

10. Function: `changeMaxPerWallet(uint256 q)`

- Description: Changes the maximum NFTs that can be minted per wallet.
- Inputs:
  - `q`: New maximum NFTs allowed per wallet.
- Relevant Information: Can only be called by the contract owner.

11. Function: setStatus(uint256 s)

- Description: Sets the status of the contract.
- Inputs:
- s: The new status to be set.
- Relevant Information: Can only be called by the contract owner.

12. Function: getStatus()

Description: Retrieves the current status of the contract.

Inputs: None.

Returns: The current status as a uint.

Relevant Information: Public function; anyone can call it.

13. Function: getPrice(uint256 _quantity)

Description: Calculates the total price for a specified number of NFTs.

Inputs:

_quantity: The number of NFTs for which the price will be calculated.

Returns: The total price as a uint256.

Relevant Information: Public function; anyone can call it.

14. Function: freeMint(address sendTo, uint quantity)

Description: Mints a specified number of NFTs for free and sends them to the specified address.

Inputs:

sendTo: The address to which the NFTs will be sent.

quantity: The number of NFTs to mint.

Relevant Information: Can only be called by the contract owner.

15.Function: changeAllowBurning(bool _value)

Description: Changes the ability to burn NFTs.

Inputs:

_value: The new boolean value for allowing burning (true to enable, false to disable).

Relevant Information: Can only be called by the contract owner.

16.Function: burn(uint256 tokenId)

Description: Burns a specific NFT.

Inputs:

tokenId: The ID of the NFT to be burned.

Relevant Information: Can only be called by the owner of the NFT, and only if burning is allowed.

17. Function: WL1_InProoflist(bytes32[] memory _proof, address _owner)

Description: Checks if an address is in the whitelist 1 proof list.

Inputs:

_proof: The Merkle proof list.

_owner: The address to check.

Returns: True if the address is in the whitelist 1 proof list, false otherwise.

Relevant Information: Public function; anyone can call it.

18. Function: WL2_InProofList(bytes32[] memory _proof, address _owner)

Description: Checks if an address is in the whitelist 2 proof list.

Inputs:

_proof: The Merkle proof list.

_owner: The address to check.

Returns: True if the address is in the whitelist 2 proof list, false otherwise.

Relevant Information: Public function; anyone can call it.

19.Function: WL3_InProofList(bytes32[] memory _proof, address _owner)

Description: Checks if an address is in the whitelist 3 proof list.

Inputs:

_proof: The Merkle proof list.

_owner: The address to check.

Returns: True if the address is in the whitelist 3 proof list, false otherwise.

Relevant Information: Public function; anyone can call it.

20.Function: setWhiteListMerkleRoot(bytes32 _whiteListMerkleRoot, uint8 _whiteListType)

Description: Sets the Merkle root for a specified whitelist type.

Inputs:

_whiteListMerkleRoot: The new Merkle root.

_whiteListType: The whitelist type (1, 2, or 3).

Relevant Information: Can only be called by the contract owner.

21. Function: getWhiteListPrice(uint256 _quantity, uint8 _whiteListType)

Description: Retrieves the total price for a specified number of NFTs for a given whitelist type.

Inputs:

_quantity: The number of NFTs for which the price will be calculated.

_whiteListType: The whitelist type (1, 2, or 3).

Returns: The total price as a uint256.

Relevant Information: Public function; anyone can call it.

22. Function: setWhiteListMaxMint(uint256 _listMaxMint, uint8 _whiteListType)

Description: Sets the maximum NFTs allowed to be minted per transaction for a given whitelist type.

Inputs:

_listMaxMint: The new maximum NFTs allowed per transaction.

_whiteListType: The whitelist type (1, 2, or 3).

Relevant Information: Can only be called by the contract owner.

23. Function: setWhiteListPerWallet(uint256 _listPerWallet, uint8 _whiteListType)

Description: Updates the maximum number of NFTs that can be minted per wallet for a given whitelist type.

Inputs:

_listPerWallet: New maximum NFTs allowed per wallet for the given whitelist type.

_whiteListType: The type of whitelist for which the maximum NFTs per wallet should be updated.

Relevant Information: Can only be called by the contract owner.

24. Function: setWhiteListAvailable(uint256 _listCollection, uint8 _whiteListType)

Description: Updates the total number of NFTs available for a given whitelist type.

Inputs:

_listCollection: New total number of NFTs available for the given whitelist type.

_whiteListType: The type of whitelist for which the total number of NFTs available should be updated.

Relevant Information: Can only be called by the contract owner.

25. Function: setWhiteItemPrice(uint256 _newPrice, uint8 _whiteListType)

Description: Updates the price for a single NFT for a given whitelist type.

Inputs:

_newPrice: The new price for a single NFT for the given whitelist type.

_whiteListType: The type of whitelist for which the price of a single NFT should be updated.

Relevant Information: Can only be called by the contract owner.

26.Function: mintWhiteListTokens(uint256 _howMany, bytes32[] calldata _proof)

Description: Mints a given number of NFTs for the calling address if they are in the whitelist for the current active sale.

Inputs:

_howMany: The number of NFTs to be minted.

_proof: The Merkle proof for the calling address to be on the whitelist.

Relevant Information: Only callable when the current sale status is active and the calling address is in the whitelist for the current sale.

Finally, the contract includes overrides for on-chain royalties. The functions `setApprovalForAll`, `approve`, `transferFrom`, `safeTransferFrom`, and `safeTransferFrom` with data all override the corresponding functions in the ERC721 standard to ensure that only allowed operators can interact with the NFTs.

Here are the descriptions for the two modifiers:

Modifier: whiteSaleActive

Description: This modifier is used to ensure that the current sale status is either 1, 2, or 3, indicating that the whitelist sale is currently active. This ensures that only those who have been approved for the whitelist sale can mint NFTs during this time.

<u>Modifier: checkList</u>

Description: This modifier is used to ensure that certain conditions are met before minting NFTs during a whitelist sale. The conditions checked include ensuring that the total number of NFTs that will be minted after this transaction will not exceed the maximum collection size, the sender is on the whitelist, the correct amount of ether has been sent for the desired quantity of NFTs, the desired quantity of NFTs does not exceed the maximum allowed per transaction or per wallet, and the NFTs have not already been claimed by the sender. This modifier is used to ensure that the minting process during a whitelist sale is fair and follows the predetermined rules for the sale.

- ***Thank you.***