Sapienza
Università di Roma

Department of Computer Science

MeltyFi

Blockchain and Distributed Ledger Technologies

Professors:
Claudio Di Ciccio

Students:
Vincenzo Imperati
Benigno Ansanelli
Andrea Princic

Academic Year 2022/2023

Contents

1 Preface

1.1 Brief presentation of the protocol

The **MeltyFi protocol** presented in the report proposes an innovative solution for **peer-to-pool lending and borrowing with NFT collateral**. Thanks to its structure and **independence from off-chain factors** such as the floor price of NFTs, MeltyFi allows borrowers to easily obtain loans **without the risk of involuntary liquidation of the NFT**, allowing them to obtain liquidity without risking the loss of their NFT. Additionally, the MeltyFi protocol allows lenders to use their capital to **provide liquidity for loans through a lottery system**. For this use of capital, lenders are obviously rewarded, and in the event that the loan they funded is not repaid, they have the opportunity to win the NFT used as collateral proportional to the capital they provided for that loan. If the loan is repaid, the lenders are obviously returned the capital they invested. Those who also repay a loan are rewarded with an amount equal to the interest paid to the protocol.

In summary, MeltyFi guarantees for all users an easy access to a decentralized system of lending and borrowing with NTFs collateral, independent of external factors, and encourages users to provide liquidity and repay loans in order to be rewarded for their correct behavior in proportion to their personal capital invested in the protocol. MeltyFi represents an effective solution for **"making the illiquid liquid"** and offers **benefits to both borrowers and lenders**.

1.2 Outline of the report

Descrivere brevemente di cosa parla ogni sezione citandole in ordine

1.3 Team members and main responsibilities

Descrivere brevemente quali sono state le nostre personali responsabilita'

2 Background

2.1 Blockchain technology

Blockchain is a technology that enables the creation of a **distributed, immutable and secure ledger of transactions**. It relies on a network of nodes that collaborate to **maintain a coherent and up-to-date record of transactions**, known as the ”blockchain”. Each transaction is inserted into a ”block” (with others transactions) that is added to the existing chain, creating a history of all transactions made on the network.

The security of the blockchain is ensured through the use of public key cryptography to sign transactions and a **decentralized consensus system** to validate transactions and add new blocks to the chain. The consensus system used depends on the specific blockchain implementation, but the most common ones are Proof of Work (PoW) and Proof of Stake (PoS).

One of the main features of the blockchain is its **transparency**, as all transactions are public and visible to all network participants. However, it is possible to use cryptography to maintain user anonymity and ensure the privacy of transactions.

The blockchain has a wide range of applications, from cryptocurrencies to supply chain management, from electronic voting to decentralized finance (DeFi) applications. The flexibility of blockchain technology allows the creation of smart contracts, which enable the automation of complex processes through the execution of secure and tamper-proof code on the network.

In addition, the blockchain offers greater security compared to centralized systems as it does **not rely on a single entity or authority** for its operation and **is not subject to single point of failure**. This makes the blockchain particularly suitable for scenarios where it is important to ensure the security of transactions and the immutability of data.

2.1.1 Ethereum blockchain

Ethereum is a Proof of Stake blockchain that uses the Ether (ETH) as its native coin [1]. It is the second-largest blockchain per market cap, with more than 180 billion US dollars [2], just below Bitcoin. One of the reasons for its success is the introduction of smart contracts, pieces of code that are decentralized and execute directly on-chain. Smart contracts allow decentralized applications (DApp) and are fundamental to the so-called Web 3.0.

2.1.2 Smart contracts

Smart contracts are computer programs that run on the blockchain and can automate the process of negotiating. Due to their decentralized nature, smart contracts are considered **trustworthy and secure**, as their **code runs uncensorably** and **cannot be changed** once posted on the blockchain. This makes them particularly suitable for handling transactions and forging contractual relationships where transparency and impartiality are important.

Another advantage of smart contracts is their ability to **eliminate intermediaries**, thus reducing costs and time to negotiate. Furthermore, smart contracts can be used to create incentives for correct user behavior, as the conditions of the contract can be automated in order to penalize users who do not comply with their obligations. Furthermore, smart contracts can be used to create decentralized governance systems, where decisions are made transparently and democratically by all network users.

2.1.3 Tocken ERC standard

Ethereum smart contracts can be used to create different types of tokens, each of which has specific functions and uses. Let's examine the three main types of Ethereum tokens: ERC-20, ERC-721 and ERC-1155, and explain their characteristics and applications.

ERC-20 ERC-20 tokens are the most popular Ethereum token and are the norm for token creation on the platform. They are designed to be **interchangeable** and to adhere to an interoperability standard, which makes them easy to implement in different DApps and to trade on decentralized marketplaces. ERC20 tokens are typically used to represent units of a certain resource, such as loyalty points or gaming tokens.

ERC-721 ERC-721 tokens, by contrast, represent **unique, non-fungible assets**, such as works of art or real estate. These tokens are used to represent goods that have intrinsic value or are rare or unique. We will analyze them better later.

ERC-1155 ERC-1155 tokens represent **a combination of the first two types of tokens**, ERC-20 and ERC-721, and allow you to manage both fungible and non-fungible assets within a single contract. These tokens are particularly suitable for building decentralized games or representing a collection of assets. Furthermore, they are often used as tickets and are extremely functional because they adapt to the needs of both ERC-20s and ERC-721s.

2.1.4 Summary

Blockchain technology and smart contracts have the potential to revolutionize several sectors, from finance to healthcare, through the supply chain and energy. For example, blockchain technology can be used to create more efficient and transparent financial markets, while smart contracts can be used to automate supply chain management and ensure product traceability.

In conclusion, blockchain technology and smart contracts represent an important step forward in the development of decentralized and transparent systems, which can be used to automate and guarantee the security of contractual relationships.

2.2 Application domain

This section describes the basic concepts and phenomena that will allow us to more clearly understand the purpose of the MeltyFi protocol and the problems it solves. Here we will analyze the phenomenon of NFTs, their value and the manipulations of the latter. We will define what a lending and borrowing platform is and we will analyze platforms of this kind that make use of NFTs to allow you to obtain loans.

2.2.1 NFTs

Non-fungible tokens(NFTs) have become popular as unique and non-interchangeable units of data that signify ownership of associated digital items, such as images, music, or videos. **Token "ownership" is recorded and tracked on a blockchain** [3].

The market for NFTs, transferrable and unique digital assets on public blockchains, has received widespread attention and experienced strong growth since early 2021. Prominent examples of NFTs, such as the **artist Beeple** selling a piece of digital art for $69 million [4] or **Twitter CEO Jack Dorsey** auctioning off his first-ever tweet for $2.9 million [5], show that NFTs have received mainstream attention and represent a popular application in FinTech and the cryptocurrency ecosystem.

NFTs are **unique certificates of authenticity on blockchains** that are usually issued by the creators of the underlying assets. These assets can be digital or physical in nature. Fungible goods such as money or trade goods can be exchanged for goods of the same kind. By contrast, non-fungible items cannot be exchanged for a similar good because their value exceeds the actual material value. Examples from the analogue world include items of artistic or historical significance, or rare trading cards, all of which have a long history of trading in auctions and other marketplaces. In the digital world, it has so far been difficult to trade and auction non-fungible goods, as their authenticity was hard to verify. NFTs now pave the way for the digitization and trade of unique values on the internet [6].

The Etherum blockchain has welcomed the phenomenon of NFTs and it is precisely on this blockchain that the most famous and valuable NFTs reside [7].

2.2.2 NFT floor price

Above all due to the non-tangibility and novelty of this type of asset, it is difficult to decipher the true value and therefore define the right price. The floor price is the most used and most established price indicator to determine the value of an NFT. In general, NFT floor prices are an attempt by market participants to gather information about the fair market value of an NFT project at the collection level. This helps focus an NFT buyer's decision-making and analysis by eliminating factors in the collection such as rarity, traits, and more. [8].

The floor price, however, is a **parameter external to the blockchain**, and therefore as such it is **subject to manipulation** for various purposes. This makes it even more difficult to correctly determine the true value of NFTs, contributing more to increasing their volatility'. **Two phenomena** of floor price manipulation are described below. Both of these methods intentionally mislead prospective NFT buyers into believing that the fair market value of an NFT is the new floor price when in reality the price is not a result of natural demand. This is harder to factor in compared to other NFT floor price factors, and requires NFT buyers to do their due diligence on NFT ownership metrics, market sales, project community, and more.

Sweeping the floor Some NFT collections, often those which are low-priced, can be subject to price manipulation in the form of mass buying. Referred to as "sweeping the floor" in NFT communities, groups or wealthy individuals can make a concentrated effort to raise the floor price. In this scenario, the floor price is defined as the lowest-priced NFT in a collection as priced by marketplaces.

Wash trading Another way to manipulate the price is through wash trading, where an individual trades their own NFTs. Simply put, a group or individual with enough NFTs can manipulate the price by listing their own NFTs on a marketplace for a more expensive price, and then buying them to artificially inflate the price [9].

2.2.3 Lending and borrowing platform

collateral liquidation pool

describe to me what Lending and borrowing platform is in defi as in the ethereum blockchain. in detail using scientific terms and citing examples of decentralized lending and borrowing protocols on ethereum. what are the strengths and weaknesses

2.2.4 NFTs as loan collateral

Nowadays NFTs are used as collateral to secure loans.

2.2.5 Peer-to-pool lending and borrowing with NFTs collateral

Definition In Peer-to-Pool lending, multiple lenders provide liquidity into a pool, and the borrower take the liquidity from that pool. These pools algorithmically set a threshold for the floor price of the NFT. If it falls below the threshold, is automatically liquidated.

Pro So it's easy to achieve loan.

Con Remember though that the floor price of an NFT is very volatile, so one could see its dear NFT liquidated for a sudden variation in the floor price.

Existing protocols

2.2.6 Peer-to-peer lending and borrowing with NFTs collateral

Definition Another way is peer-to-peer lending. The lender and borrower agree on the NFT's value, the length of the term, and the amount of interest. At the end of the expiry date, if the borrower can't repay the loan in time, the NFT is sent to the lender's wallet as collateral for the unpaid amount.

Pro With this protocol, there's no need for floor price, the price is decided by the two peers. This also implies no sudden liquidation.

Con It's difficult to achieve an agreement, and a single lender has to have all the money for the loan.

Existing protocols

2.2.7 Summary

3 Presentation of the context

3.1 Aim of MeltyFi protocol

meltyfi is a peer-to-pool lending and borrowing platform with nfts collaterals designed in a new different way. There's no floor price dependence, so no liquidation risk. This because the borrower creates a lottery, and put as prize of the lottery his NFT. The lottery tickets, in our case, the wonka bars, are sold to multiple lenders, in this way the borrower obtains liquidity.

If the borrower repays the loan before the set expire date, his nft is sent back to him and lenders are refounded. if borrower does not repay, at the expire date his nft is sent to the lottery winner.

is also easy, for the borrower, achieve loan thanks to peer-to-pool design, where there are multiple lenders and no a single one. meltify reward with the choco chips the borrowers that repay loans and all the lenders that buy wonka bars

3.2 Use case scenarios

Let's now jump into the flow of the protocol in the two possible scenarios.

3.2.1 Borrower repay loan

In the first scenario, the borrower applies for the loan and before the expiry date the borrower repays the loan. The borrower sends their NFT to meltyfi, which creates a lottery and holds the NFT for its duration. Lenders that are interested in the lottery can buy tickets related to the lottery. 5% of the value is kept in the treasury, while the remaining 95% is given to the borrower as liquidity. A wonka bar is minted for the lender.

Before the expiry date, the borrower returns the liquidity to close the lottery. The borrower is awarded with Choco Chips and the NFT is returned to them. Now, lenders can melt their wonka bars to be awarded with Choco Chips, and refunded of their investment.

3.2.2 Borrower does not repay loan

In the second scenario, the borrower applies for the loan but doesn't repay it at the end. As before, borrower creates a lottery and lenders buy wonka bars.

But this time, at the expiry date, the borrower doesn't repay the loan so the oracle extracts a winning ticket (the golden ticket). The winner's wonka bar is burnt and they are awarded with Choco Chips and the NFT. As before, lenders can melt their wonka bars and be awarded with Choco Chips.

3.3 Why using a blockchain

4 Software architecture

4.1 Tools used

4.2 Protocol blueprint

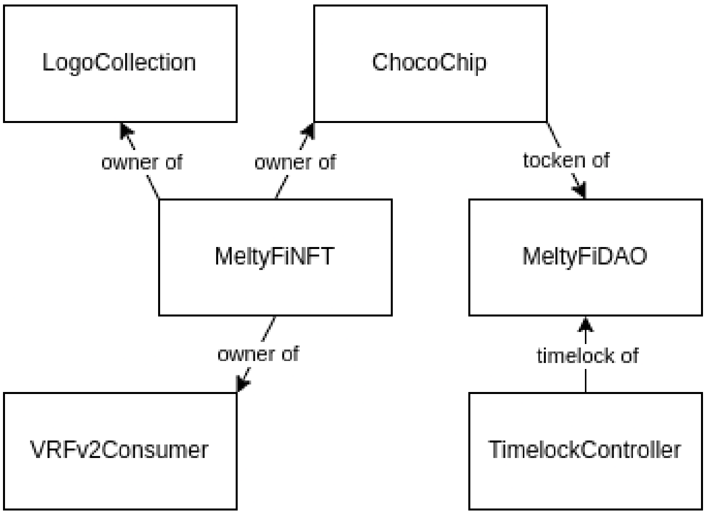
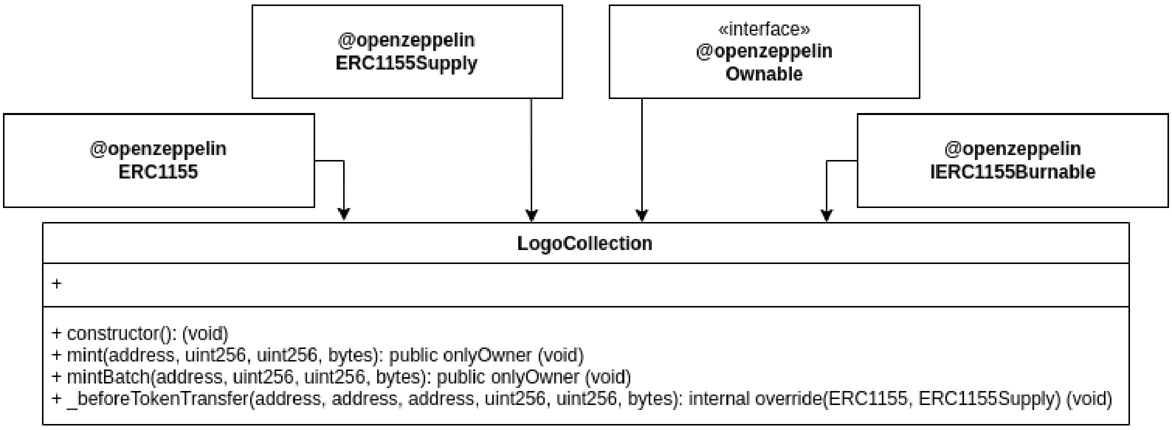


Figure 1: Protocol blueprint

4.3 LogoCollection



```
+ constructor(): (void)
+ mint(address, uint256, uint256, bytes): public onlyOwner (void)
+ mintBatch(address, uint256, uint256, bytes): public onlyOwner (void)
+ _beforeTokenTransfer(address, address, address, uint256, uint256, bytes): internal override(ERC1155, ERC1155Supply) (void)
```

Figure 2: LogoCollection.sol class diagram

4.4 ChocoChip

Allow the Choco Chip holders to take part in the modification and development decisions of MeltyFi. This is possible thanks to Choco Chip and MeltyFi DAO. Borrowers who repay the loans are given 1 $CHOC for every Finney of interest paid. Lenders are given 1 $CHOC for every Finney spent on Wonka Bars. For example, Choco Chip holders will be able to vote to stake the Ether in the DAO treasury and distribute staking rewords among Choco Chip holders.

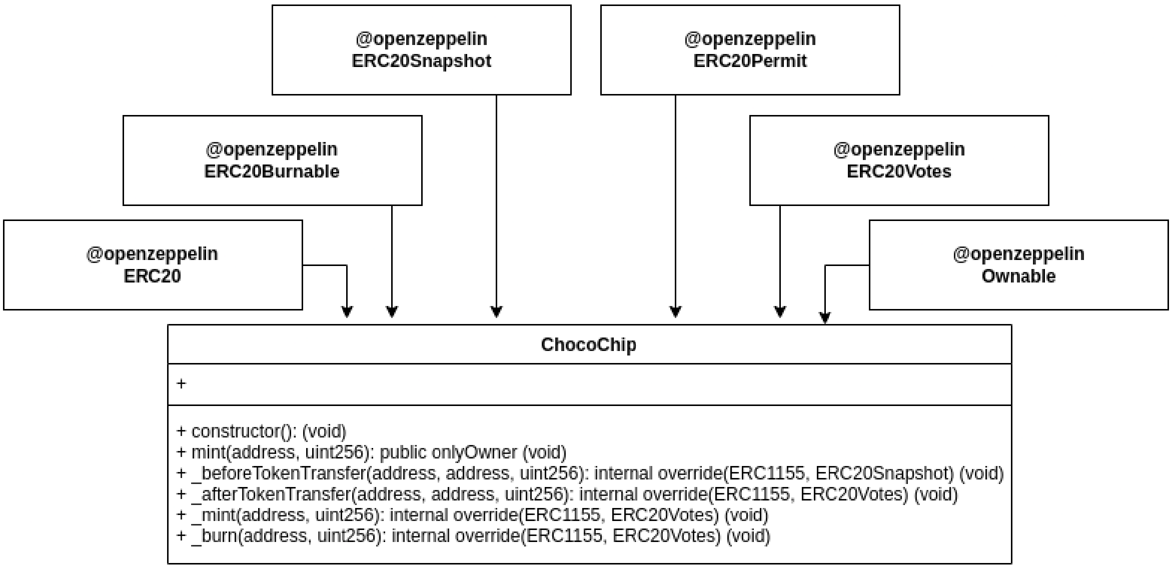

@openzeppelin
ERC20Snapshot
@openzeppelin
ERC20Permit
@openzeppelin
ERC20Burnable
@openzeppelin
ERC20Votes
@openzeppelin
ERC20
@openzeppelin
Ownable
ChocoChip
+
+ constructor(): (void)
+ mint(address, uint256): public onlyOwner (void)
+ _beforeTokenTransfer(address, address, uint256): internal override(ERC1155, ERC20Snapshot) (void)
+ _afterTokenTransfer(address, address, uint256): internal override(ERC1155, ERC20Votes) (void)
+ _mint(address, uint256): internal override(ERC1155, ERC20Votes) (void)
+ _burn(address, uint256): internal override(ERC1155, ERC20Votes) (void)

Figure 3: ChocoChip.sol class diagram

4.5 TimelockController

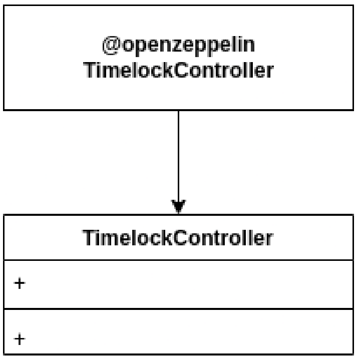


Figure 4: TimelockController.sol class diagram

4.6 MeltyFiDAO

4.7 VRFv2Consumer

4.8 MeltyFiNFT

MeltyFiNFT is the contract that runs the core functionality of the MeltyFi protocol. It manages the creation, cancellation and conclusion of lotteries, as well as the sale and refund of WonkaBars for each lottery, and also reward good users with ChocoChips. The contract allows users to create a lottery by choosing their NFT to put as lottery

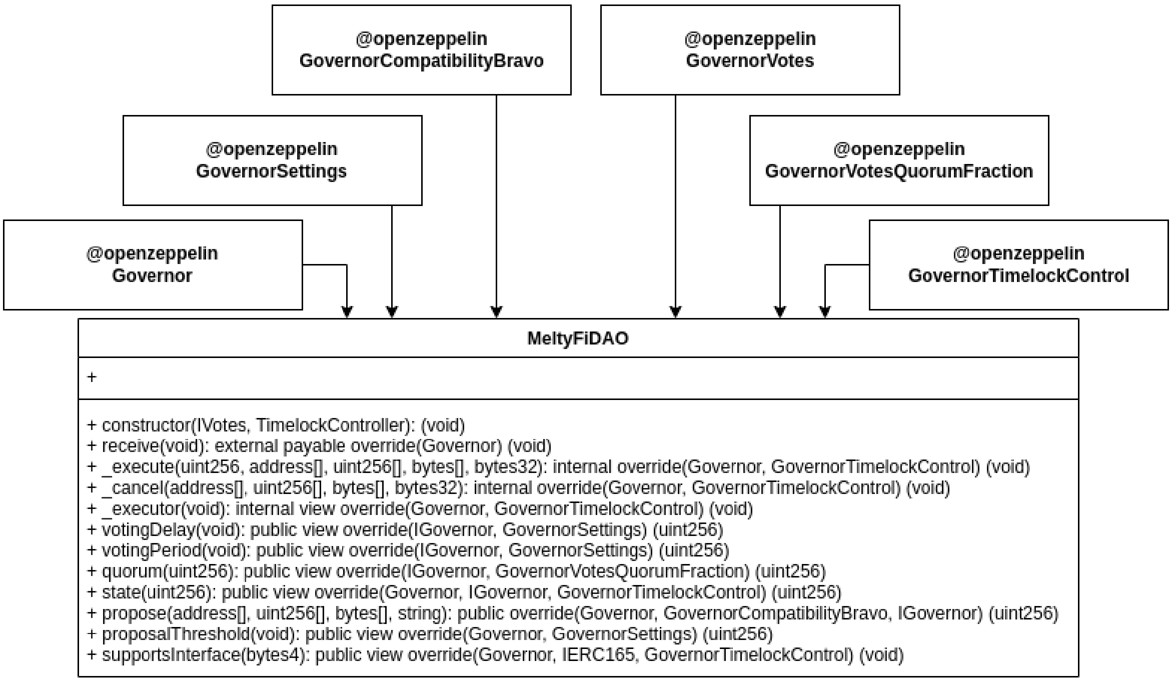

@openzeppelin
GovernorCompatibilityBravo
@openzeppelin
GovernorVotes
@openzeppelin
GovernorSettings
@openzeppelin
GovernorVotesQuorumFraction
@openzeppelin
Governor
@openzeppelin
GovernorTimelockControl
MeltyFiDAO
+
+ constructor(IVotes, TimelockController): (void)
+ receive(void): external payable override(Governor) (void)
+ _execute(uint256, address[], uint256[], bytes[], bytes32): internal override(Governor, GovernorTimelockControl) (void)
+ _cancel(address[], uint256[], bytes[], bytes32): internal override(Governor, GovernorTimelockControl) (void)
+ _executor(void): internal view override(Governor, GovernorTimelockControl) (void)
+ votingDelay(void): public view override(IGovernor, GovernorSettings) (uint256)
+ votingPeriod(void): public view override(IGovernor, GovernorSettings) (uint256)
+ quorum(uint256): public view override(IGovernor, GovernorVotesQuorumFraction) (uint256)
+ state(uint256): public view override(Governor, IGovernor, GovernorTimelockControl) (uint256)
+ propose(address[], uint256[], bytes[], string): public override(Governor, GovernorCompatibilityBravo, IGovernor) (uint256)
+ proposalThreshold(void): public view override(Governor, GovernorSettings) (uint256)
+ supportsInterface(bytes4): public view override(Governor, IERC165, GovernorTimelockControl) (void)

Figure 5: MeltyFiDAO.sol class diagram

prize, setting an expiration date and defining a price in Ether for each WonkaBar sold. When a lottery is created, the contract will be able to mint a fixed amount of WonkaBars (setted by lottery owner) for the lottery. These WonkaBars are sold to users interested in participating in the lottery and money raised are sent to the lottery owner (less some fees). Once the expiration date is reached, the contract selects a random WonkaBar holder as the winner, who receives the prize NFT. Plus every wonkabar holder is rewarded with ChocoCips. If the lottery is cancelled by the owner beafore the expiration date, the contract refunds WonkaBars holders with Ether of the lottery owners. Plus every wonkabar holder is rewarded with ChocoCips.

4.8.1 createLottery

This public function creates a new lottery. The function raises error in the following cases:

- Error if the caller is not the owner of the prize.
- Error if the maximum number of Wonka Bars for sale is greater that the upper bound.

The function parameters are the following:

- **duration**: The duration of the lottery, in seconds.
- **prizeContract**: The contract that holds the prize for this lottery.
- **prizeTokenId**: The token ID of the prize for this lottery.

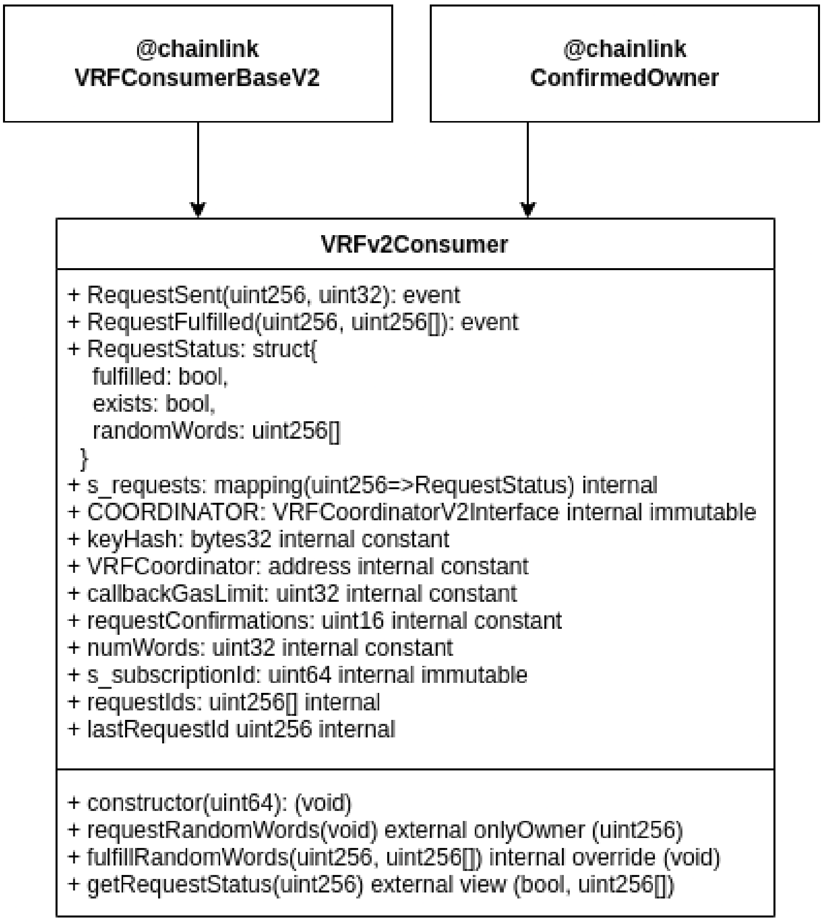

@chainlink
VRFConsumerBaseV2
@chainlink
ConfirmedOwner
VRFv2Consumer
+ RequestSent(uint256, uint32): event
+ RequestFulfilled(uint256, uint256[]): event
+ RequestStatus: struct{
fulfilled: bool,
exists: bool,
randomWords: uint256[]
}
+ s_requests: mapping(uint256=>RequestStatus) internal
+ COORDINATOR: VRFCoordinatorV2Interface internal immutable
+ keyHash: bytes32 internal constant
+ VRFCoordinator: address internal constant
+ callbackGasLimit: uint32 internal constant
+ requestConfirmations: uint16 internal constant
+ numWords: uint32 internal constant
+ s_subscriptionId: uint64 internal immutable
+ requestIds: uint256[] internal
+ lastRequestId uint256 internal
+ constructor(uint64): (void)
+ requestRandomWords(void) external onlyOwner (uint256)
+ fulfillRandomWords(uint256, uint256[]) internal override (void)
+ getRequestStatus(uint256) external view (bool, uint256[])

Figure 6: VRFv2Consumer.sol class diagram

- **wonkaBarPrice**: The price of a Wonka Bar in this lottery.
- **wonkaBarsMaxSupply**: The maximum number of Wonka Bars that can be sold in this lottery.

The function return the ID of the new lottery.

```
function createLottery(
    uint256 duration,
    IERC721 prizeContract,
    uint256 prizeTokenId,
    uint256 wonkaBarPrice,
    uint256 wonkaBarsMaxSupply
) public returns (uint256)
{
    /// The maximum number of Wonka Bars for sale must not be greater than the upper bound
    /// The maximum number of Wonka Bars for sale must not be lower than the lower bound
```

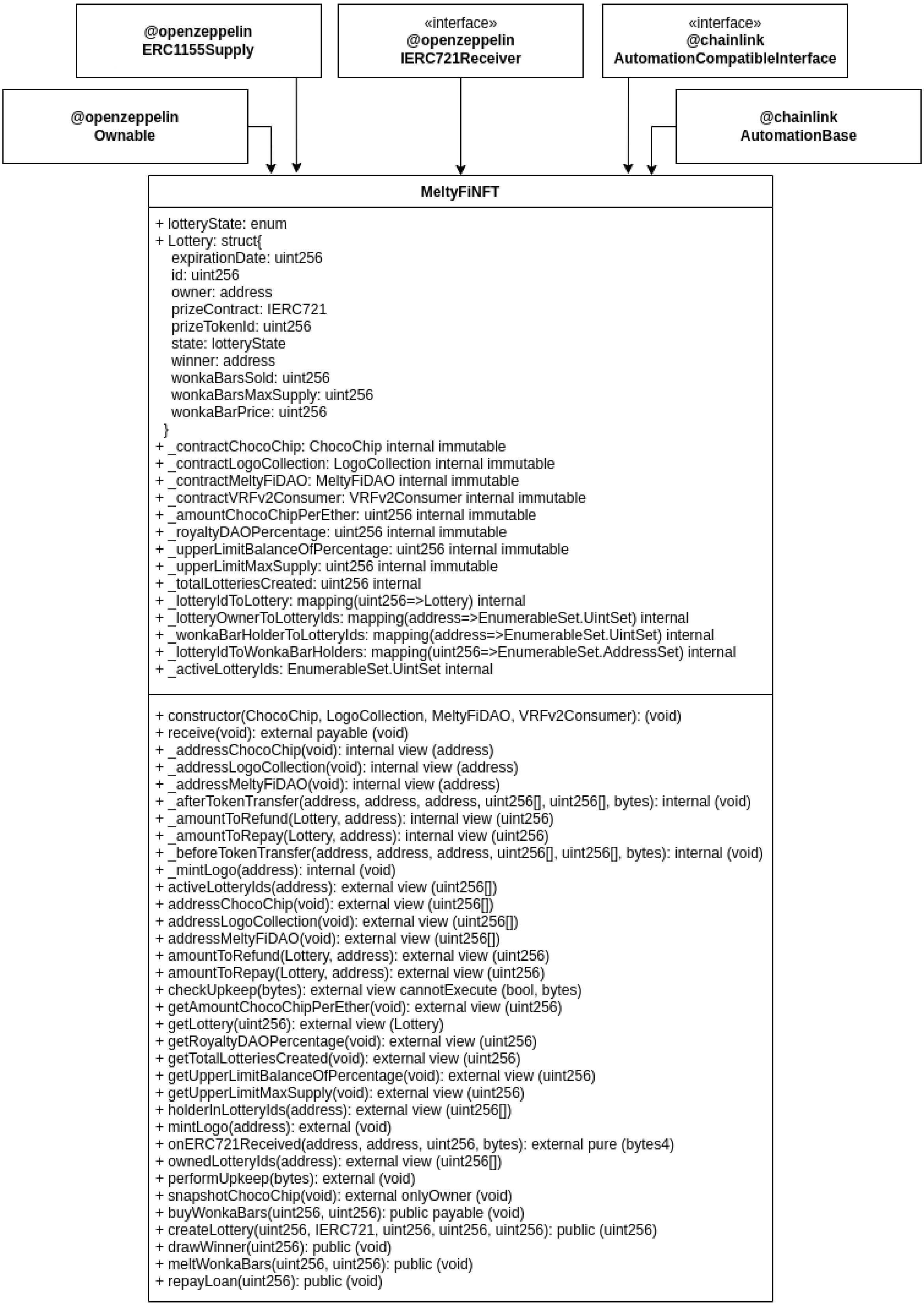

«interface»
@openzeppelin
ERC1155Supply
@openzeppelin
IERC721Receiver
«interface»
@chainlink
AutomationCompatibleInterface
@openzeppelin
Ownable
@chainlink
AutomationBase
MeltyFiNFT
+ lotteryState: enum
+ Lottery: struct{
expirationDate: uint256
id: uint256
owner: address
prizeContract: IERC721
prizeTokenId: uint256
state: lotteryState
winner: address
wonkaBarsSold: uint256
wonkaBarsMaxSupply: uint256
wonkaBarPrice: uint256
}
+ _contractChocoChip: ChocoChip internal immutable
+ _contractLogoCollection: LogoCollection internal immutable
+ _contractMeltyFiDAO: MeltyFiDAO internal immutable
+ _contractVRFv2Consumer: VRFv2Consumer internal immutable
+ _amountChocoChipPerEther: uint256 internal immutable
+ _royaltyDAOPercentage: uint256 internal immutable
+ _upperLimitBalanceOfPercentage: uint256 internal immutable
+ _upperLimitMaxSupply: uint256 internal immutable
+ _totalLotteriesCreated: uint256 internal
+ _lotteryIdToLottery: mapping(uint256=>Lottery) internal
+ _lotteryOwnerToLotteryIds: mapping(address=>EnumerableSet.UintSet) internal
+ _wonkaBarHolderToLotteryIds: mapping(address=>EnumerableSet.UintSet) internal
+ _lotteryIdToWonkaBarHolders: mapping(uint256=>EnumerableSet.AddressSet) internal
+ _activeLotteryIds: EnumerableSet.UintSet internal
+ constructor(ChocoChip, LogoCollection, MeltyFiDAO, VRFv2Consumer): (void)
+ receive(void): external payable (void)
+ _addressChocoChip(void): internal view (address)
+ _addressLogoCollection(void): internal view (address)
+ _addressMeltyFiDAO(void): internal view (address)
+ _afterTokenTransfer(address, address, address, uint256[], uint256[], bytes): internal (void)
+ _amountToRefund(Lottery, address): internal view (uint256)
+ _amountToRepay(Lottery, address): internal view (uint256)
+ _beforeTokenTransfer(address, address, address, uint256[], uint256[], bytes): internal (void)
+ _mintLogo(address): internal (void)
+ activeLotteryIds(address): external view (uint256[])
+ addressChocoChip(void): external view (uint256[])
+ addressLogoCollection(void): external view (uint256[])
+ addressMeltyFiDAO(void): external view (uint256[])
+ amountToRefund(Lottery, address): external view (uint256)
+ amountToRepay(Lottery, address): external view (uint256)
+ checkUpkeep(bytes): external view cannotExecute (bool, bytes)
+ getAmountChocoChipPerEther(void): external view (uint256)
+ getLottery(uint256): external view (Lottery)
+ getRoyaltyDAOPercentage(void): external view (uint256)
+ getTotalLotteriesCreated(void): external view (uint256)
+ getUpperLimitBalanceOfPercentage(void): external view (uint256)
+ getUpperLimitMaxSupply(void): external view (uint256)
+ holderInLotteryIds(address): external view (uint256[])
+ mintLogo(address): external (void)
+ onERC721Received(address, address, uint256, bytes): external pure (bytes4)
+ ownedLotteryIds(address): external view (uint256[])
+ performUpkeep(bytes): external (void)
+ snapshotChocoChip(void): external onlyOwner (void)
+ buyWonkaBars(uint256, uint256): public payable (void)
+ createLottery(uint256, IERC721, uint256, uint256, uint256): public (uint256)
+ drawWinner(uint256): public (void)
+ meltWonkaBars(uint256, uint256): public (void)
+ repayLoan(uint256): public (void)

Figure 7: MeltyFiNFT.sol class diagram

```
    /// transfer the prize to this contract
    /// create a new lottery
    /// update internal state
    /// return the ID of the new lottery
}
```

4.8.2 buyWonkaBars

This function allows a user to buy a specified amount of Wonka Bars for a lottery. The caller must send the correct amount of Ether along with the transaction. A percentage of the total spending will be transferred to the MeltyFiDAO contract and the rest will be transferred to the owner of the lottery. The caller's balance of Wonka Bars for the specified lottery will also be updated. The function raises errors in the following cases:

- Error if the lottery is not really active.
- Error if after this purchease the total supply of WonkaBars will exceed the maximum supply allowed.
- Error if the caller's balance of Wonka Bars for this lottery, after the purchase, will exceed the `_upperLimitBalanceOfPercentage`.
- Error if the value sent is not enough to cover the cost of the Wonka Bars.

The function parameters are the following:

- **lotteryId**: The ID of the lottery for which the Wonka Bars are being purchased.
- **amount**: The number of Wonka Bars to be purchased.

```
function buyWonkaBars(
    uint256 lotteryId,
    uint256 amount
) public payable
{
    /// retrieve the lottery with the given ID
    /// calculate the total spending for the Wonka Bars
    /// The lottery must be really active
    /// After this purchease the total supply of WonkaBars must not exceed the maximum supply allowed
    /// The caller's balance of Wonka Bars for this lottery, after the purchase, must not exceed the _upperLimitBalanceOfPercentage
    /// The caller must sent anough amount of Ether to cover the cost of the Wonka Bars
    /// transfer _royaltyDAOPercentage of the total spending to the MeltyFiDAO contract
    /// transfer the rest of the total spending to the owner of the lottery
    /// mint the Wonka Bars for the caller
    /// update the total number of Wonka Bars sold for the lottery
}
```

4.8.3 repayLoan

This function repays the loan for the given lottery ID. The caller of the function must be the owner of the lottery. The function raises errors in the following cases:

- Error if the caller is not the owner of the lottery.
- Error if the value sent is not enough to repay the loan.
- Error if the lottery is not more active.

The function parameters are the following:

- **lotteryId**: The id of the lottery to repay the loan for.

```
function repayLoan(
    uint256 lotteryId
) public payable
{
    /// retrieve the lottery with the given ID
    /// Calculate the total amount to be repaid
    /// The caller must be the owner of the lottery
    /// The caller must sent anough amount of Ether to repay the loan
    /// The lottery must be active
    /// Mint Choco Chips to the owner of the lottery
    /// Transfer the prize to the owner of the lottery
    /// Remove the lottery from the active lotteries
    /// set the expiration date to the current block timestamp
    /// If the total supply of WonkaBars is 0
        /// set the state to TRASHED
    /// else
        /// set the state to CANCELLED
}
```

4.8.4 drawWinner

This function draws the winner of a lottery. The function raises errors in the following cases:

- Error if the lottery state is not active.
- Error if the lottery expiration date is not passed.
- Error if the VRF request for random words is not fulfilled.

The function parameters are the following:

- **lotteryId**: The ID of the lottery.

```
function drawWinner(
    uint256 lotteryId
) public
{
    /// retrieve the lottery with the given ID
    /// The lottery state must be active
    // The lottery expiration date must be passed
    /// remove lottery from the active lotteries
    /// if there are no WonkaBar sold transfer prize to the owner, otherwise set lottery winner
        /// transfer prize to the owner if no tokens were sold
        /// set lottery state to trashed
    /// else
        /// The VRF request for random words must be fulfilled
        /// set lottery winner
        /// set lottery state to concluded
}
```

4.8.5 meltWonkaBars

This function allows a user to melt their WonkaBars of a specific lottery and receive a refund in return. The function raises errors in the following cases:

- Error if the user does not have enough WonkaBar balance to melt the given amount.
- Error if the lottery is trashed.
- Error if lottery is really active.
- Error if lottery is waiting to be concluded by the oracle.

The function parameters are the following:

- **lotteryId**: The ID of the lottery from which the WonkaBars will be melted.
- **amount**: The amount of WonkaBars to be melted.

```
function meltWonkaBars(
    uint256 lotteryId,
    uint256 amount
) public
{
    /// retrieve the lottery with the given ID
    /// calculate the total refound for the Wonka Bars
    /// the user must have enough WonkaBar balance to melt the given amount
    /// the lottery must not be trashed
    /// lottery must not be really active or waiting to be concluded by the oracle
    /// Burn the Wonka Bars for the caller
    /// Mint Choco Chips to the caller
    /// if lottery state is cancelled, also refound the caller
    /// if the caller is the winner and he does not already receive the price
        /// transfer prize to the caller (the winner)
    /// if all lottery's WonkaBars are melted, trash the lottery
}
```

4.9 Deepening of the design choices

5 Implementation

5.1 Tools used

5.2 MeltyFi.NFT DApp

5.2.1 Home page

5.2.2 Lotteries page

5.2.3 Profile page

5.3 MeltyFi.DAO DApp

6 Known issues and limitations

Let's now talk about some limitations of the protocol. For the borrower is hard to get a fast loan, since funds are to be retrieved from many lenders who buy wonka bars. moreover lenders are uncertain about the future of the lottery, because there are three different endings:

- first one, if the borrower repays the loan, lenders will be awarded with Choco Chips and refunded of their investment;
- second, if the borrower doesn't repay the loan and the wonka bar does not have the golden ticket, the lender will only be awarded with Choco Chips;
- but if the the wonka bar has the golden ticket (third case), the lender will be awarded with Choco Chips and the NFT of the borrower.

7 Conclusions

References

[1] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. https://ethereum.org/en/whitepaper.

[2] Coinmarketcap. https://coinmarketcap.com.

[3] Kristen E. Busch. Non-fungible tokens (nfts). Government Report R47189, CRS Reports, July 2022. https://crsreports.congress.gov/product/pdf/R/R47189.

[4] Christie's beeple (b. 1981). https://onlineonly.christies.com/s/first-open-beeple/beeple-b-1981-1/112924.

[5] Valuables auction site 'just setting up my twttr'. https://v.cent.co/tweet/20.

[6] Lennart Ante. The non-fungible token (nft) market and its relationship with bitcoin and ethereum. *MPDI*, 1(3):216–224, June 2022. https://www.mdpi.com/2674-1032/1/3/17.

[7] Nonfungible nft market history. https://nonfungible.com/market-tracker.

[8] Chainlink. What is an nft floor price? https://blog.chain.link/what-is-an-nft-floor-price, October 2022.

[9] Massimo La Morgia, Alessandro Mei, Alberto Maria Mongardini, and Eugenio Nerio Nemmi. Nft wash trading in the ethereum blockchain. *arXiv preprint arXiv:2212.01225*, 2022.

[10] VincenzoImp. Meltyfi. https://github.com/VincenzoImp/MeltyFi, October 2022.

[11] Nftgo. https://nftgo.io.

[12] Benddao. https://www.benddao.xyz.

[13] Dropsdao. https://drops.co.

[14] Liquidnfts. https://liquidnfts.com.

[15] nftfi. https://www.nftfi.com.

[16] Arcade. https://www.arcade.xyz.

[17] X2y2. https://x2y2.io.

[18] Jpeg'd. https://jpegd.io.

[19] Pine. https://pine.loans.

[20] renft. https://www.renft.io.

[21] Meltyfi.nft. https://meltyfi.nft.

[22] Meltyfi.dao. https://meltyfi.dao.

[23] Remix. https://remix-project.org.

[24] Metamask. https://metamask.io.

[25] Goerli faucet. https://goerlifaucet.com.

[26] Goerli opensea. https://testnets.opensea.io.

[27] Gerli etherscan. https://goerli.etherscan.io.

[28] @EridianAlpha. Maximal extractable value (mev). https://ethereum.org/en/developers/docs/mev, November 2022.

[29] Non-fungible tokens (nft). https://ethereum.org/en/nft.