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 in detail what a lending and borrowing platform on blockchain is, one of the most famous use cases of decentralized finance (DeFi); 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][4].

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 [5] or **Twitter CEO Jack Dorsey** auctioning off his first-ever tweet for $2.9 million [6], 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 [7].

The Etherum blockchain has welcomed the phenomenon of NFTs and it is precisely on this blockchain that the most famous and valuable NFTs reside [8].

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. [9].

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 [10].

2.2.3 Lending and borrowing platform on blockchain

blockchain lending and borrowing platforms are the main driver of decentralized finance (**DeFi**) [11]. These platforms allow users to lend and borrow assets in a **decentralized manner**, without the need for a central intermediary. These platforms typically use smart contracts to facilitate lending and borrowing transactions, and they often use a variety of different assets as collateral.

Before describing the two largest lending and borrowing platforms in Ethereum (Aave and Compound), a few words are necessary to describe the concept of collateralization, loan-to-value, liquidation and liquidity pool [12]. Finally we mention the possible risks in using lending and borrowing platforms on blockchain.

Collateralization Collateralization is a fundamental concept in the financial industry. It simply refers to something you put up as a guarantee when borrowing money. For example, if you take out a bank loan to buy a house, the house will serve as collateral. If you fail to repay your loan, the bank will repossess your home [13].

It's the same in Defi. If you want to borrow some assets from the liquidity pool, you must provide the pool with some other assets as collateral. If you do not repay your loan, the protocol will not return your collateral to you. The collateral will be used to repay your debt to the liquidity pool.

Loan-to-value Loan to value (LTV) [14] is the ratio of the loan's value to the value of collateral. In a typical financial market, credit scores determine the risk involved in a loan. The lower the credit score, the higher the risk for lenders. Instead of credit scores, the crypto lending process offers asset-backed loans.

LTV determines the amount of cryptocurrency one would need as collateral before one could get a loan. The lender holds on to this collateral until the loan is fully paid back.

The main benefit of LTV in crypto lending is that it helps minimize the risk on the lender's part. The user also benefits from LTV in that they can access larger loans at lesser interest rates.

Liquidation In traditional finance, liquidation occurs when a company or group must sell some of its assets at a loss to cover a debt. DeFi liquidations are similar in that users take out debt from a protocol and provide crypto assets as collateral to back the debt. Thus, DeFi liquidation is the process by which a smart contract sells crypto assets to cover the debt [15].

Liquidity pool A liquidity pool is a crowdsourced pool of cryptocurrencies or tokens locked in a smart contract that is used to facilitate trades between the assets on a decentralized exchange (DEX). Instead of traditional markets of buyers and sellers, many decentralized finance (DeFi) platforms use automated market makers (AMMs), which allow digital assets to be traded in an automatic and permissionless manner through the use of liquidity pools [16].

Aave Aave is a decentralized non-custodial liquidity protocol where users can participate as depositors or borrowers [17]. Depositors provide liquidity to the market to earn a passive income, while borrowers are able to borrow in an overcollateralized (perpetually) or undercollateralized (one-block liquidity) fashion. As of the time of writing (Jan. 12, 2022), the total value locked (TVL) in Aave's smart contracts stands at $3.78 billion [18].

Compound Compound (COMP) is an Ethereum-based lending and borrowing protocol that algorithmically sets interest rates based on the activity in its liquidity pools [19]. As of the time of writing (Jan. 12, 2022), the total value locked (TVL) in Compound's smart contracts stands at $3.59 billion [20]. As a Compound user, you can lend and borrow some of the most popular cryptocurrencies instantly without having to go through a traditional financial intermediary. It's one of the largest and oldest lending and borrowing apps in the crypto world.

Compound is used extensively by DeFi developers, who programmatically integrate it into their DApps and use the protocol for dynamic borrowing and lending. Many yield aggregation protocols and other DeFi apps make use of Compound. The protocol is also widely used by the general crypto public, not just developers. Non-technical users can borrow funds from Compound by supplying a different crypto coin as collateral.

Risks [21]

- **Liquidation risk**: If the pledged collateral is a volatile crypto asset (such as ETH), and the value drops too far, then the ETH may get liquidated. This is a very undesirable result, as it means that the ETH is sold off after a price drop [22].

- **Crypto volatility**: Cryptocurrency is inherently volatile, and using crypto assets to pledge collateral for a loan can lose a user a significant amount of money. First, the funds are locked into the contracts and cannot be accessed until the loan is paid off. Second, the rules for required liquidations mean losing those funds when the value drops.

- **Liquidity risk**: Users that deposit crypto may not be able to withdraw funds if the liquidity drops too far. This means that they would need to wait until more crypto is deposited by other users to be able to withdraw funds.

2.2.4 NFTs as loan collateral

Nowadays NFTs can be used as collateral to secure a loan. The loan can work like any other DeFi loan, with the difference that the collateral is the NFT itself. The NFT, by its nature, is comparable to an illiquid asset, so for loans of this type, the lender must agree on the value of the collateral and decide on a loan-to-value in agreement with the borrower. This dynamic is the main aspect that allows the execution of the loan and any repayment or liquidation. As with DeFi loans, those with NFT collateral are also managed by smart contracts, which hold the NFT for the entire duration of the loan.

Advantages The advantage of NFT collateral loans is that of being able to obtain liquidity from an illiquid asset. In this way it is possible to benefit from the market value of the NFTs held, without the need to sell them. This last aspect is very advantageous if one thinks of the unicity of the NFTs. The nfts, being unique in fact, are not fungible and therefore selling an NFT to then buy back another one does not in any way guarantee that you are buying back exactly the same one that was sold.

Disadvantages The disadvantages of NFT collateral are many. The cause of all the disadvantages (according to our personal interpretation) is linked to the difficulty of transforming illiquid assets into liquid assets without incurring any repercussions or compromises. In fact, the blockchain is a closed system and the DeFi that is built on top of it inherits this property. This means that, more specifically in our case, if in a closed system such as DeFi an asset that is illiquid by nature such as an NFT is made illiquid, the system suffers an imbalance, favoring the borrower who wants to benefit of this action (make liquid your NFT). This imbalance in a closed system must necessarily be rebalanced by showing disadvantages for some agents of the system. The latter may be the lenders or the borrower himself in the event that his own action ultimately penalizes him. This explanation just made is an attempt to define the root of all the causes which then generate all the disadvantages that can be encountered by operating with DeFi loans with NFT collateral.

Some of the main possible disadvantages and risks are summarized below. However, this topic has not yet been analyzed in detail in the literature and there are many possible attacks that can undermine loans with NFT collateral. We have collected some articles to help the reader develop a personal idea on this topic [23][24][25][26].

- **Market Conditions**: Considering the NFT market is extremely volatile, the value of NFTs fluctuates by the second. With that, lenders risk being stuck with an overvalued NFT if the project plummets in value.

- **Risk of Default**: If you can't pay back the loan, you could lose your NFT. Considering many lenders lend less money than a particular NFT might be worth, you could end up losing money in the long run.

- **Improper Valuation**: An improper proper valuation of an asset means that you run the risk of losing money as a lender. Also, borrowers risk losing the ability to borrow more money if their valuation is undervalued.

We close the discussion by recalling that in order for a lending and borrowing platform to work well with NFT collateral, it is necessary to defend oneself against all possible negative scenarios that disadvantage the user of the platform itself.

2.2.5 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

2.2.6 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

- Arcade
- BendDAO
- DropsDAO
- JPEG'd
- LiquidNFTs
- nftfi
- NFTGo
- Pine
- reNFT

- X2Y2

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 The chocolate factory inside MeltyFi

In the MeltyFi protocol, each NFT is comparable to a **chocolate bar**. When the borrower applying for a loan, he decides how many squares of chocolate to divide his bar (his NFT) and what the price is for each square. In doing so, the chocolate bar (the NFT) is actually breaking down into small squares of chocolate (the **WonkaBars**), smaller than the bar itself. In this way, therefore, the borrower requests the loan and the illiquid and inseparable NFT is starting to lose these two properties (**it is melting**). Once this process is complete, lenders can decide whether and how many WonkaBars to buy. By buying a WonkaBar, the lender is financing the loan requested by the borrower.

In the event that the borrower repays the loan, the lender can melt the purchased WonkaBars to receive back the capital employed, plus the premium for having employed this capital (the **ChocoChips**, the fragments produced by the chocolate bar when dividing into chocolate squares).

In the event that the borrower does not repay the loan, all WonkaBar holders compete to win the NFT collateralized by the loan because the NFT is inseparable and only one WonkaBar holder can win it. Just like in the chocolate factory [27], at this point the WonkaBar holders open their WonkaBars with the hope of finding the only winning ticket (the **golden ticket**) capable of authorizing them to receive the collateral NFT as a prize. At this point, the winner can melt the WonkaBars he holds to receive the NFT as a prize, but also the ChocoChips for using his own capital in financing the loan. Those who are not winners can still melt their WonkaBars to receive only the ChocoChips.

3.3 Use case scenarios

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

3.3.1 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.

3.3.2 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.

3.4 Why using a blockchain

4 Software architecture

In this section, the software architecture of MeltyFi is shown in detail. All the tools used are mentioned, an overview of all the protocol components is given, and then they are explained individually. Finally, reflections are given regarding the design choices made.

The entire protocol is open-source and is searchable on GitHub [28]. It is also possible to interact with MeltyFiNFT via Etherscan[29] or Dapp[29]. Equally it is possible to interact with MeltyFiDAO via Etherscan[30] or Dapp[30]. MeltyFiNFT encapsulates the core of the MeltyFi protocol while MeltyFiDAO encapsulates the DAO management of the MeltyFi protocol.

4.1 Tools used

For the realization of the MeltyFi protocol we relied on the environment **Node.js** [31] in order to use the following npm packages:

- **@openzeppelin/contracts**: used for draw on all the already audited code that we would use when necessary [32]
- **@chainlink/contracts**: used for the realization of the whole part that makes use of oracles [33]
- **hardhat**: used as development environment [34]
- **@nomicfoundation/hardhat-toolbox**: used to deploy on goerli testnet and verify contract on etherscan [35]
- **hardhat-docgen**: used for automatic generation of code documentation [36]
- **dotenv**: used to load sensible datas in the code such as private keys and API keys [37]

The **Alchemy API** [38] was used to make deployment possible, and the **Etherscan API** [39] was used to make contract verification possible. **Remix** [40] was used as an editor while writing the contracts. The latter were compiled with **solidity compiler 0.8.17**. Overall, all the MeltyFi protocol is capable of running on the same or higher versions of **solidity 0.8.9**. This choice made avoids overflow problems and allows us to benefit from all the audit code that was intended to be used.

Finally, to allow MeltyFi to function properly, which requires interaction with oracles, the entire protocol was deployed on **Goerli testnet** [41]. **Goerli faucet** [42] and **Link faucet** [43] were used to enable this.

4.2 Protocol blueprint

The blueprint of the entire protocol is shown in Figure 1. As already mentioned, **MeltyFiNFT** is the contract that handles all the main lending and borrowing part of the protocol. MeltyFiNFT is also a contract that handles an ERC-1155 token, this token is none other than WonkaBar (the utility token). MeltyFiNFT is owner of **LogoCollection** (the meme token) to be able to minate at the user's request, it is owner of **ChocoChip** (the governance token) to be able to minate when due, and finally it is owner of **VRFv2DirectFundingConsumer** to make a request for a random word to be able to elect the winner of a completed lottery. **MeltyFiDAO** is the contract that manages the DAO of the protocol, so it needs a **TimelockController** and CochoChip as the governance token.

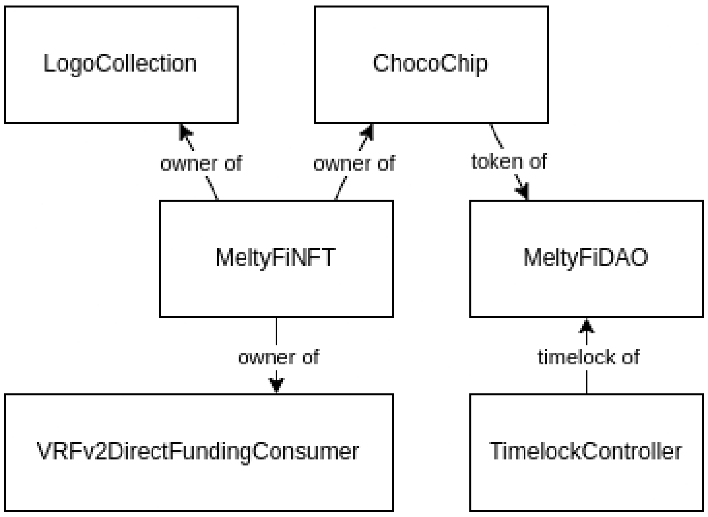


Figure 1: Protocol blueprint

4.3 LogoCollection

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.

Figure 2: LogoCollection

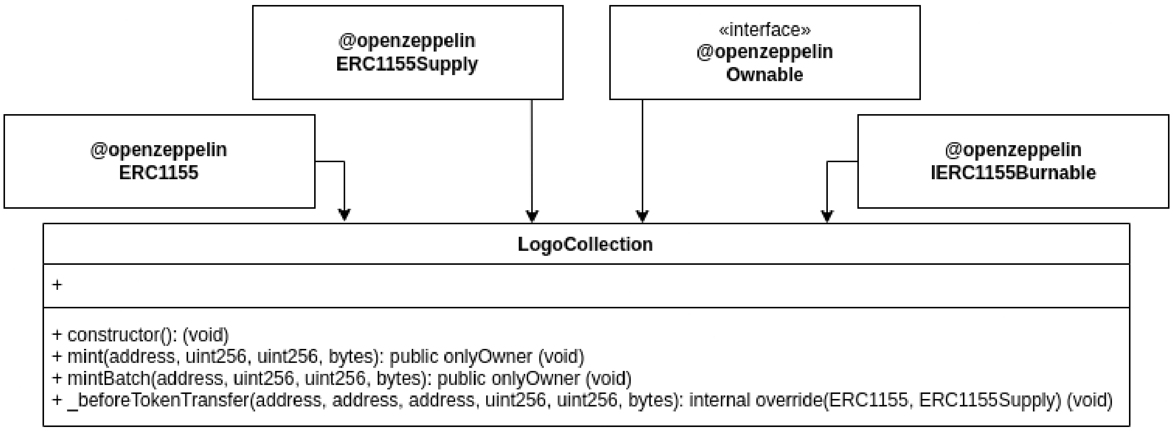

@openzeppelin
ERC1155Supply
«interface»
@openzeppelin
Ownable
@openzeppelin
ERC1155
@openzeppelin
IERC1155Burnable
LogoCollection
+
+ constructor(): (void)
+ mint(address, uint256, uint256, bytes): public onlyOwner (void)
+ mintBatch(address, uint256, uint256, bytes): public onlyOwner (void)
+ _beforeTokenTransfer(address, address, address, uint256, uint256, bytes): internal override(ERC1155, ERC1155Supply) (void)

Figure 3: LogoCollection.sol class diagram

4.5 TimelockController

4.6 MeltyFiDAO

4.7 VRFv2DirectFundingConsumer

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 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

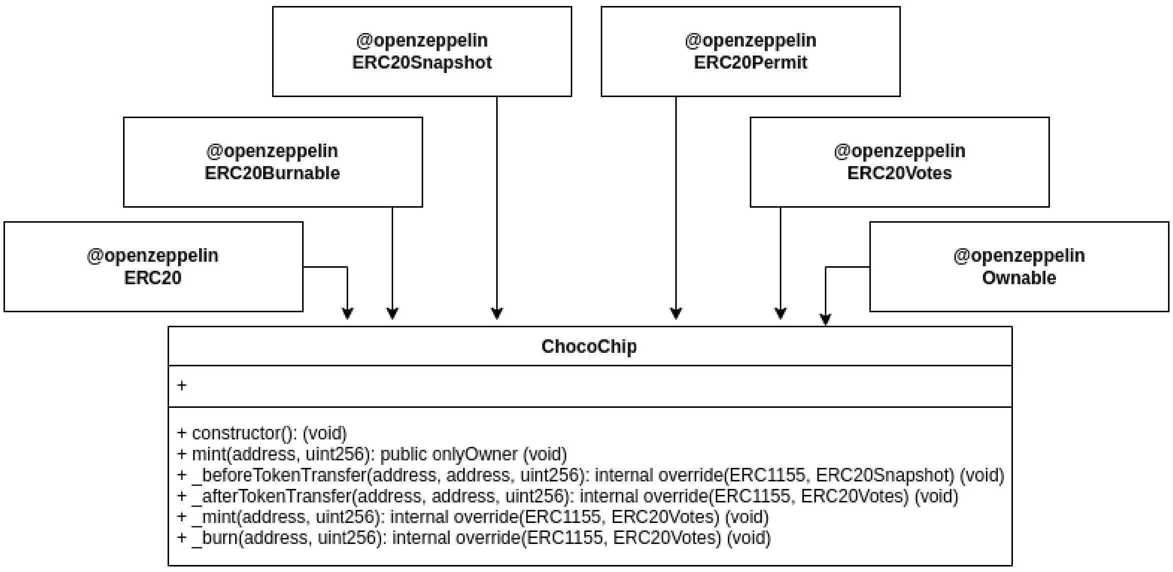

@openzeppelin
ERC20Snapshot
@openzeppelin
ERC20Permit
@openzeppelin
ERC20Burnable
@openzeppelin
ERC20Votes
@openzeppelin
ERC20
@openzeppelin
Ownable
ChocoChip
+
+ constructor(): (void)
+ mint(address, uint256): public onlyOwner (void)
+ _beforeTokenTransfer(address, address, uint256): internal override(ERC1155, 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 4: ChocoChip.sol class diagram

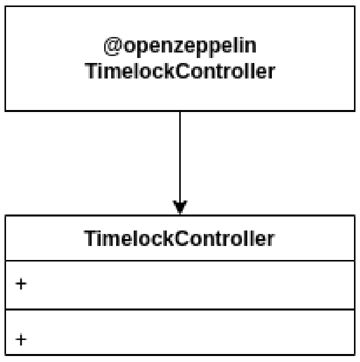


Figure 5: TimelockController.sol class diagram

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 WonkaBar

4.8.2 Lottery

```
/// Data type representing the possible states of a lottery
enum lotteryState {
    ACTIVE,
    CANCELLED,
    CONCLUDED,
    TRASHED
```

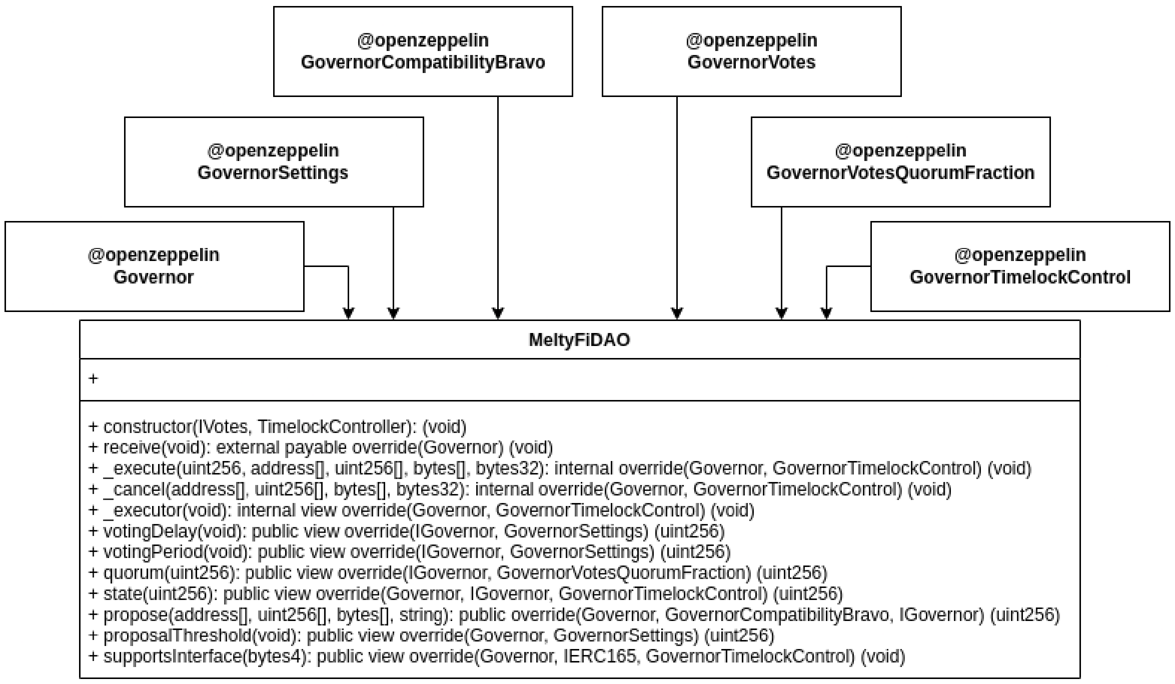

@openzeppelin GovernorCompatibilityBravo
@openzeppelin GovernorVotes
@openzeppelin GovernorSettings
@openzeppelin GovernorVotesQuorumFraction
@openzeppelin Governor
@openzeppelin GovernorTimelockControl
MeltyFiDAO
+
+ constructor(IVotes, TimelockController): (void)
+ receive(void): external payable override(Governor) (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 6: MeltyFiDAO.sol class diagram

```
}
/// Struct for storing the information of a lottery
struct Lottery {
    /// Expiration date of the lottery, in seconds
    uint256 expirationDate;
    /// ID of the lottery
    uint256 id;
    /// Owner of the lottery
    address owner;
    /// Prize NFT contract of the lottery
    IERC721 prizeContract;
    /// Prize NFT token ID of the lottery
    uint256 prizeTokenId;
    /// State of the lottery
    lotteryState state;
    /// Winner of the lottery
    address winner;
    /// Number of WonkaBars sold for the lottery
    uint256 wonkaBarsSold;
    /// Maximum supply of WonkaBars for the lottery
    uint256 wonkaBarsMaxSupply;
    /// Price of each WonkaBar for the lottery, in wei
    uint256 wonkaBarPrice;
```

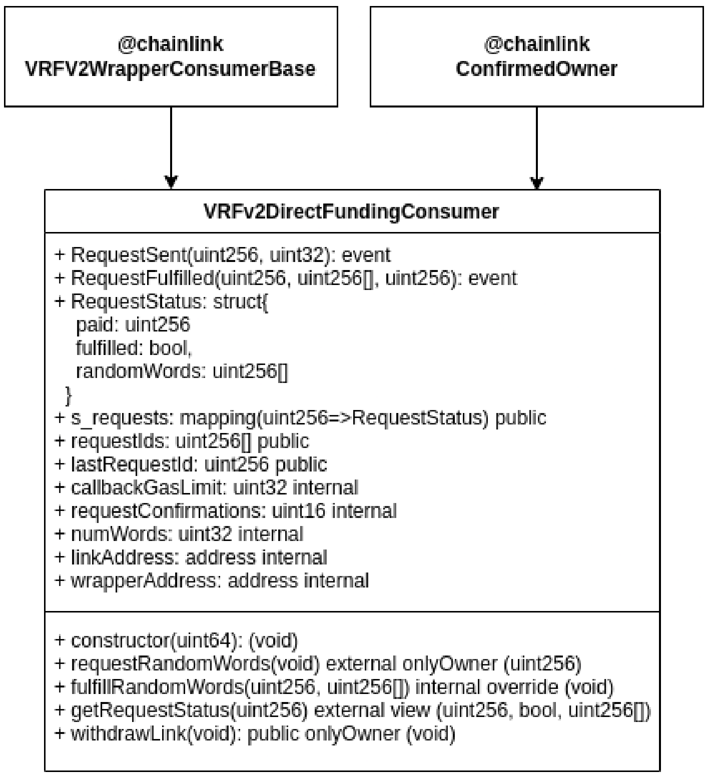

@chainlink
VRFV2WrapperConsumerBase
@chainlink
ConfirmedOwner
VRFv2DirectFundingConsumer
+ RequestSent(uint256, uint32): event
+ RequestFulfilled(uint256, uint256[], uint256): event
+ RequestStatus: struct{
paid: uint256
fulfilled: bool,
randomWords: uint256[]
}
+ s_requests: mapping(uint256=>RequestStatus) public
+ requestIds: uint256[] public
+ lastRequestId: uint256 public
+ callbackGasLimit: uint32 internal
+ requestConfirmations: uint16 internal
+ numWords: uint32 internal
+ linkAddress: address internal
+ wrapperAddress: address internal
+ constructor(uint64): (void)
+ requestRandomWords(void) external onlyOwner (uint256)
+ fulfillRandomWords(uint256, uint256[]) internal override (void)
+ getRequestStatus(uint256) external view (uint256, bool, uint256[])
+ withdrawLink(void): public onlyOwner (void)

Figure 7: VRFv2DirectFundingConsumer.sol class diagram

```
}
```

4.8.3 Data structs

```
/// maps a unique lottery ID to a "Lottery" object containing information about the lottery itself
mapping(uint256 => Lottery) internal _lotteryIdToLottery;
/// maps the address of a lottery owner to a set of lottery IDs that they own
mapping(address => EnumerableSet.UintSet) internal _lotteryOwnerToLotteryIds;
/// maps the address of a WonkaBar holder to a set of lottery IDs for which they have purchased a ticket
mapping(address => EnumerableSet.UintSet) internal _wonkaBarHolderToLotteryIds;
/// maps a lottery ID to a set of WonkaBar holder addresses that have purchased a ticket for that lottery
mapping(uint256 => EnumerableSet.AddressSet) internal _lotteryIdToWonkaBarHolders;
/// set that stores the IDs of all active lotteries
EnumerableSet.UintSet internal _activeLotteryIds;
```

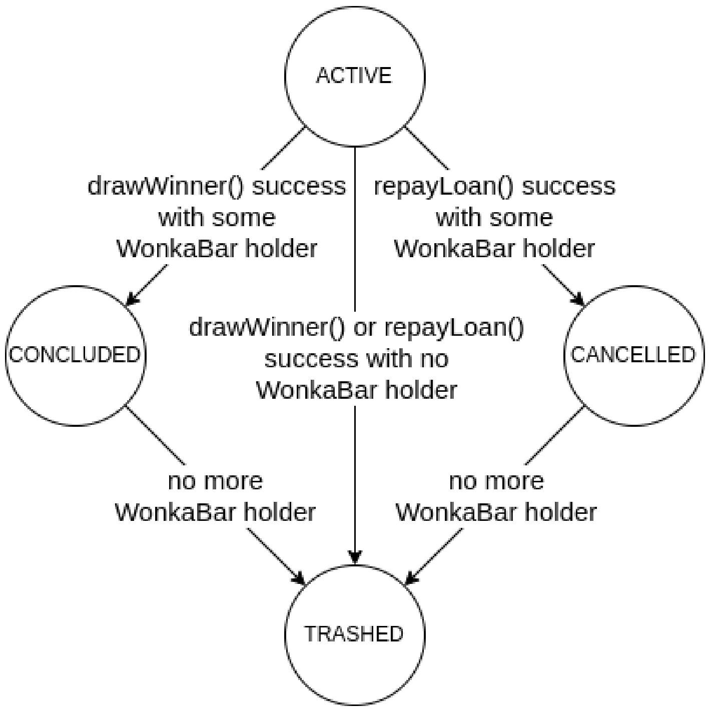

ACTIVE
drawWinner() success with some WonkaBar holder
repayLoan() success with some WonkaBar holder
CONCLUDED
drawWinner() or repayLoan() success with no WonkaBar holder
CANCELLED
no more WonkaBar holder
no more WonkaBar holder
TRASHED

Figure 8: Lottery state diagram

4.8.4 createLottery function

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

4.8.5 buyWonkaBars function

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
}
```

4.8.6 repayLoan function

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
}
```

4.8.7 drawWinner function

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
}
```

4.8.8 meltWonkaBars function

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:

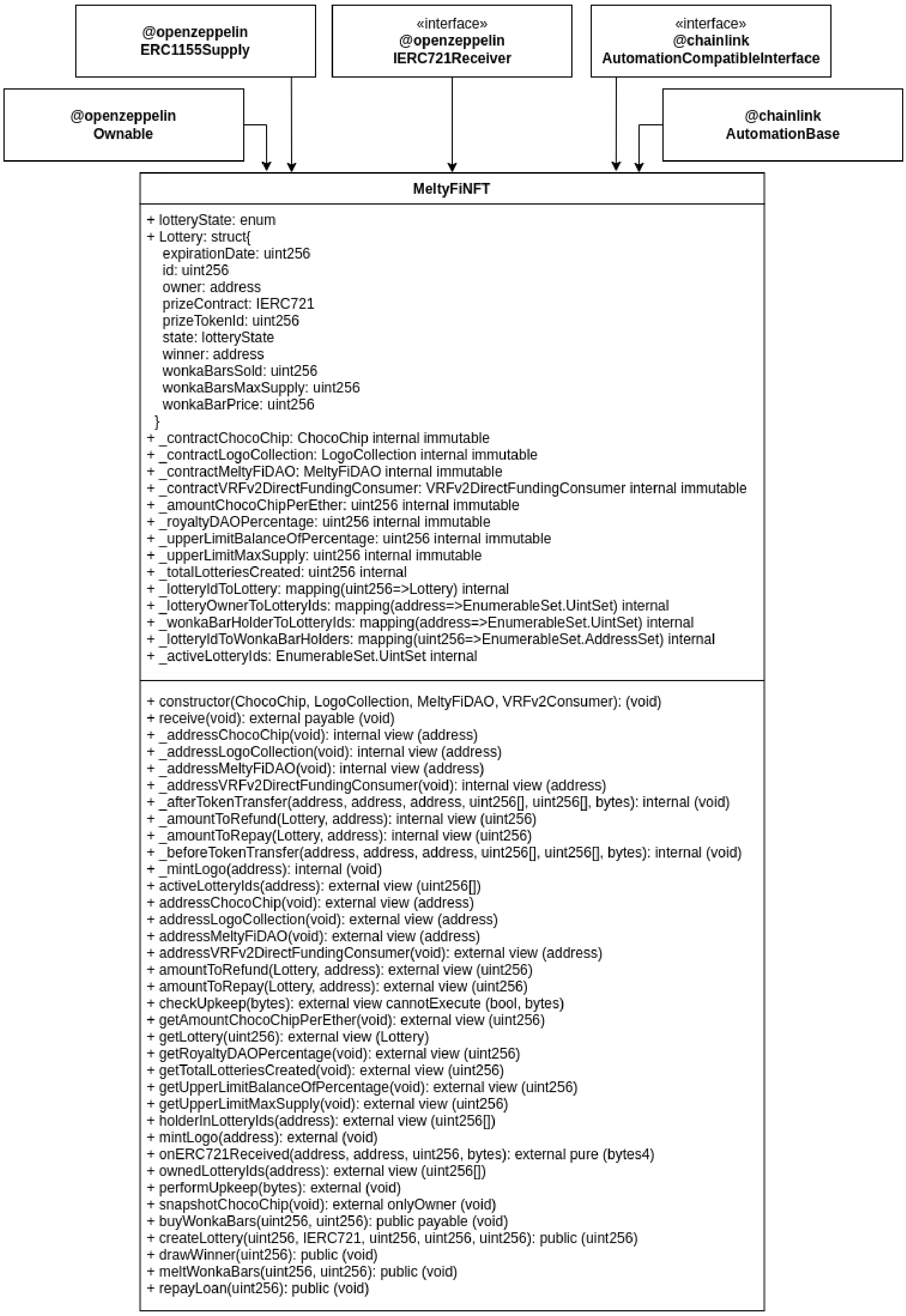
- Error if the user does not have enough WonkaBar balance to melt the given amount.
- Error if the lottery is trashed.
- Error if lottery is really active.
- Error if lottery is waiting to be concluded by the oracle.

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



@openzeppelin
ERC1155Supply
«interface»
@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
+ _contractVRFv2DirectFundingConsumer: VRFv2DirectFundingConsumer 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)
+ _addressVRFv2DirectFundingConsumer(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 (address)
+ addressLogoCollection(void): external view (address)
+ addressMeltyFiDAO(void): external view (address)
+ addressVRFv2DirectFundingConsumer(void): external view (address)
+ 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 9: MeltyFiNFT.sol class diagram

5 Implementation

5.1 Tools used

react-bootstrap, thirdweb, ethers, metamask, openseaapi, goerli(?)

5.2 MeltyFi.NFT DApp

buyWonkaBars: 0xa8017555c918384405794472eb8559144dc563cc427703b9cb5e310c4aab6222
createLottery: 0xe68f3f68b00dce4c299d0205dfbc72c302ae4d5e089b16d3c024755db70ffdf3

5.2.1 Home page and website fruition

The placement of the protocol description on the homepage serves as an effective introduction for users who may be unfamiliar with the technology. This approach allows them to quickly grasp the purpose and functionality of the protocol. As the protocol becomes more widely adopted, this information may be relocated to an "Info" section, accompanied by tutorials to enhance the user's understanding and utilization of the DApp.

As best practice we inserted a navbar that linked to the various pages, so that in each point of the DApp the user could easily move around the website. At the bottom we inserted a footer that contains useful addresses.

A fundamental thing is that thirdweb has a standard implementation of the login button that is really nice, and well integrated with metamask.

5.2.2 Lotteries page

The layout of this page features two tabs, "Browse Lotteries" and "Create Lottery." You can't access neither if you are not logged with metamask. In priciple to see active lotteries is needed just metamask, the point was that it's better to login with your metamask account at the beginning, rather than when buying, since when buying it could be that you wonna be fast. The "Browse Lotteries" tab allows users to view lotteries created by other users. We did an investigation on other lottery websites, and asked our self what could be the best experience for the user. This led us to the use of cards for each lottery, since they have an NFT as a prize, the actual image of the NFT is crucial, as users are highly influenced by what they see when the buy NFTs. Just below there are the information about the lotteries. The recurring principle for the frontend was to make the user experience as nice as possible, so many little details were added, for example the date is visualized based on the browser's language, too small price are not shown and you can see how many tickets are sold, so you know your chances of winning. The only button present on each card lets you buy the wonkabars for that lottery. We made a choice slightly different from normal in this case. When you want to buy a lottery ticket, you'll enter into a modal view, where the background is blurred and you can just see the information for you lottery. In this way you're truly focused on what you're doing. In this view, we considered important for the user a link that leads to the user contract, and another one that leads to the nft contract, in this way the user can conveniently decide if the owner and the contract are worth trusting. Also, now the wonkabar price is displayed entirely, no matter how small it is. There's a selector of how many tickets (i.e. wonkabars) it can buy and the final price is automatically computed, so the user doesn't have to insert the price. Again, the recurrent idea is always the same, the user is prevented from making mistakes, this also relieves the backend. The user can insert only positive numbers. The protocol forbids to buy more than 25 percent of the tickets, so the user can't insert a number of tickets that makes this property false. This requires some call to the state of the lottery, but saves the user (and the chain) from a wrong transaction that will be refused. In this way we eliminated many errors that could occur, but in the eventuality of other errors (e.g. the user can't afford the transaction) an error message will be displayed. When the user clicks buy, a nice metamask form will appear, and the modal view will close once the transaction is ended (unless there's an error). Another example of how the front-end could help the backend, is that the oracles could fail (probably due to insufficients fund) and don't close the lottery, so the frontend filters the active lotteries given by the backend adding also a filter for lottery with an expiry date that is greater than now.

The create lottery tab shows your nfts. There's no easy method to retrieve all the nfts using ethers, so we used opensea api, that for a given address gives you the list of all the active owners. Once again, in create lottery you're presented only with nfts of type ERC721, so that you can't put as collateral another asset (e.g. a fungible token). The card this time contains only the collection name and the id of the token, plus the button create lottery. As usual, when the user decides a button for which create a lottery, it enters into a model view. We decided to make the user insert the value for each ticket in wei, the reason for that is that it's more easy to choose between different values, especially for our purposes since we had little resources. We don't exclude to use eth in the future. The expiration date is inserted through a date picker, so that it is more easy for the user to select when he wants to end the lottery, rather than inserting the seconds. Also the total revenue is displayed, so that the user will know how much money he'll get if he sells all the tickets. Once the user creates the lottery, the frontend behaves a little bit differently. As always the reason is to avoid failing transactions. In order to create a lottery the user must approve the token for transfering to meltify address. If this transaction is executed the user can create a lottery. So the fronted first waits for the approve, and only if the original transaction is logged into the blockchain, the call for the lottery is created. If at any moment the transaction fails, an error is executed.

5.2.3 Profile page

The Profile page can only be accessed by a logged user (either with Metamask, repay loan quando nessuno ha comprato wonkabars repay loan quando qualucno ha comprato wonka bars melt wonka bar di una lotterya cancellata melt wonkabar di una lotteria conclusa di cui non sono vincitore melt wonkabar di una lotteria conclusa di cui sono vincitore

5.3 Deepening in the implementation

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

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

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

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

[7] Lennart Ante. The non-fungible token (nft) market and its relationship with bitcoin and ethereum. *FinTech*, 1(3):216–224, 2022.

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

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

[10] 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.

[11] What is decentralized finance (defi) and how does it work? https://www.investopedia.com/decentralized-finance-defi-5113835.

[12] Defi terms. https://medium.com/coinmonks/what-is-liquidation-in-defi-lending-and-borrowing-platforms-3326e0ba8d0.

[13] Collateral definition, types, & examples. https://www.investopedia.com/terms/c/collateral.asp.

[14] Loan-to-value (ltv). https://coinmarketcap.com/alexandria/glossary/loan-to-value-ltv.

[15] What is liquidation? https://www.investopedia.com/terms/l/liquidation.asp.

[16] What are liquidity pools? https://www.gemini.com/cryptopedia/what-is-a-liquidity-pool-crypto-market-liquidity.

[17] Aave. https://aave.com.

[18] Aave tvl. https://defillama.com/protocol/aave.

[19] Compound. https://compound.finance.

[20] Compound tvl. https://defillama.com/protocol/compound.

[21] Jon Frost Doerr, Anneke Kosse, Asad Khan, Ulf Lewrick, Benoît Mojon, Benedicte Nolens, and Tara Rice. Defi risks and the decentralisation illusion. *BIS Quarterly Review*, page 21, 2021.

[22] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 336–350, 2021.

[23] Monika Ghosh. Nft-backed loans gain traction, but what are the risks? https://forkast.news/nft-backed-loans-gain-traction-risks.

[24] Alex White-Gomez. Using nfts as collateral: A crash course with franklinisbored. https://www.one37pm.com/nft/using-nfts-as-collateral, October 2022.

[25] Danny Nelson. Bank run at nft lender benddao prompts attempt to avert another liquidity crisis. https://www.coindesk.com/business/2022/08/22/bank-run-at-nft-lender-benddao-prompts-attempt-to-avert-another-liquidity-crisi
August 2022.

[26] David Pan. Rising ether prices spark liquidity crisis at nft lender benddao. https://www.bloomberg.com/news/articles/2022-08-23/nft-lender-seeks-to-ease-liquidity-crisis-with-protocol-changes, August 2022.

[27] Charlie and the chocolate factory. https://www.warnerbros.com/movies/charlie-and-chocolate-factory, July 2005.

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

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

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

[31] Node.js. https://nodejs.org.

[32] Openzeppelin contracts. https://www.npmjs.com/package/@openzeppelin/contracts.

[33] Chainlink contracts. https://www.npmjs.com/package/@chainlink/contracts.

[34] Hardhat. https://www.npmjs.com/package/hardhat.

[35] Hardhat toolbox. https://www.npmjs.com/package/@nomicfoundation/hardhat-toolbox.

[36] Hardhat docgen. https://www.npmjs.com/package/hardhat-docgen.

[37] Dotenv. https://www.npmjs.com/package/dotenv.

[38] Alchemy api. https://docs.alchemy.com/reference/api-overview.

[39] Etherscan api. https://etherscan.io/apis.

[40] Remix: The native ide for web3 development. https://remix-project.org.

[41] Gerli testnet. https://goerli.net.

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

[43] Link faucet. https://faucets.chain.link.

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

[45] Benddao - web3 data liquidity. https://www.benddao.xyz.

[46] Drops - nft as collateral & instant nft loans. https://drops.co.

[47] Liquidnfts - borrow against your nfts or fund loans to earn. https://liquidnfts.com.

[48] Nftfi - borrow & lend on the leading nft liquidity protocol. https://www.nftfi.com.

[49] Arcade - decentralized nft loan marketplace. https://www.arcade.xyz.

[50] X2y2 marketplace. https://x2y2.io.

[51] Jpeg'd - bridging the gap between defi and nfts. https://jpegd.io.

[52] Pine - instant nft loans. https://pine.loans.

[53] renft – rental infrastructure for the metaverse. https://www.renft.io.

[54] Metamask: The crypto wallet for defi, web3 dapps and nfts. https://metamask.io.

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

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