Oracle Based Dynamic NFTs

by

DYNAMICON

Carlo Seppi

Jonas Ruchti

Kai Bajka

In Cooperation with

Chainlink Labs

BLOCKCHAIN Challenge

Chainlink

University of Basel

Center for Innovative Finance

Executive Summary

One of blockchain technology's major challenges is integrating real-world data onto the blockchain in order to use it. Oracles offer a service to do exactly that, and Chainlink Labs' decentralized oracle system is one of the most secure ways to obtain such data. This paper aims to demonstrate how three of the main Chainlink Labs services, the decentralized oracle network, Keepers, and the verifiable random function (VRF), can be used with dynamic NFTs and what additional value such can offer. First, some fundamental concepts are explained, and then the ecosystem surrounding NFTs as sports collectables is presented. Subsequently, the implementation of dynamic trading cards is presented to give a comprehensible case for NFTs using real-world data in the traditional NFT setting collectables. We will then extend our storyline to describe how these dynamic NFTs can be used for more business-oriented cases, where we especially focus on parametric insurance applications, to demonstrate how dynamic NFTs can be utilized outside of their usual collectable habitat.

Table of Contents

Introduction

The use case created by Chainlink Labs in collaboration with the University of Basel for this year's Blockchain Challenge focuses on the next generation of contractual agreements. The case explores whether it is possible to use blockchain technology, more specifically smart contracts and dynamic non–fungible tokens, as contractual agreements. Further, through NFTs, the economic value of the contractual agreement should be made tradable or usable in the CeFi or DeFi.

In short, the research question is: How can a contractual agreement be packed as a dynamic NFT and used or traded in the CeFi or DeFi?

Chainlink Labs

Chainlink Labs is the leading provider of secure and reliable open-source blockchain oracle solutions. It enhances smart contracts by connecting them to a wide range of off-chain data sources and computations, such as asset prices, web APIs, IoT devices, payment systems, and more. Chainlink Labs is dedicated to developing and integrating Chainlink's services as the standard decentralized oracle framework used by smart contracts across any blockchain. By aggregating different data sources for the same information, the provided data is automatically compared, reducing the risk of adding false information onto the blockchains. Data providers are listed at Chainlink Labs and must have a monetary stake, which can be slashed if false data is provided. The whole oracle system is therefore called a decentralized oracle network.

A tamper-proof random number on the blockchain is a complex problem. Block hashes are on-chain random solutions, but validators and miners can exploit these. Centralized off-chain solutions require trust in a third party, which may be able to manipulate the results to their advantage. Chainlink VRF provides a trustless, verifiable random function invulnerable to manipulation. A random number can be requested from a Chainlink node, which creates such. It provides cryptographic proof of how the number was created and publishes it on-chain before the random number itself is published a few blocks later. This ensures a tamper-proof, verifiable, and fair random number.

The contract itself cannot initiate smart contract functions. The initiation has to come from outside, either from an externally owned account or a contract account. With the help of the Keepers service, which uses off-chain computations to monitor data-feeds continually, the

smart contract conditions can be monitored, and its functions triggered by a keeper. This can be used to update an NFT's metadata with the help of an oracle.

Non-fungible Token

In 2021 alone, the NFT sales amounted to nearly US$ 20 billion, which is roughly a 570-fold increase from 2020. However, what is an NFT?

NFTs are blockchain-based, programmable deeds of ownership to a digital asset. The Latter is unique and non-interchangeable with another digital asset or token. NFTs can be considered a "certificate of authenticity" issued by the original creator on the blockchain. In comparison, a fungible token has the same characteristics as any other and thus can be replaced or interchanged. The most common illustration of a fungible asset is fiat money, i.e. a five Swiss Franc coin can be interchanged for another five Swiss Franc coins without losing its purpose or financial value. Therefore, what makes an NFT unique is that its record is stored as a unique identity (unite256) in the form of a hash-value, on any form of distributed ledger, making it irreplaceable. NFTs are created in accordance with a certain framework or standards deployed on the blockchain. Currently, the most adopted and used blockchain protocol is Ethereum under the ERC-721 standard. However, newer, more efficient protocols that allow for the asset to be semi-fungible such as the ERC-1155 standard are starting to emerge and gain market adoption. These standards are crucial for NFTs to remain unique, be traded, and owned. On the other hand, fungible tokens are developed on the ERC-20 standard, making them thus divisible. In the below overview, some major differences between the ERC-721 and -1155 token standards are outlined:

	ERC-721	**ERC-1155**
Ease of Use	Allows single operation for each transaction	Allows multiple operations in a single transaction
Properties	Each tokenID has only one owner	Each tokenID can have multiple owners
Cost of Minting	Expensive	Cheaper (than ERC-721)
Efficiency	Low	High
Storage Requirement	More	Less (than ERC-721)
Token Creation	Only one token in a single contract	Multiple tokens in a single contract

Table 1: Differences of the two NFT standards

The aforementioned digital deed gives its holders the exclusive ability to use, sell and transfer the asset's ownership rights, as devoted by the private key signature. These rights could pertain to resale, physical redemption, digital functions, financial benefits or other intangible rights. The NFT does not necessarily "contain" the asset but rather is a programmable record of ownership with an inbuilt pointer to the asset location. Most of the assets NFTs are a record of ownership off, are stored off-chain on either a private server or on ipfs [1]. Only a few NFTs have stored all the information, including the asset on-chain, and are therefore not dependent on any third party.

Applications of NFTs

NFTs are still in their infancy by no means have all use cases been developed yet. At the point of writing, some of the most prominent use cases are listed below.

- **Digital Art/Collectibles**: Digital scarcity that NFTs enable are a natural fit for digital art/collectibles whose value is dependent on a limited supply. Further, NFTs enable artists to sell their art directly, meaning they don't have to go to a middle man and can earn on secondary sales.

- **Music NFTs**: Similar to art and collectibles, music NFTs provide proof of ownership, allow artists to cut out the middleman and earn on secondary sale.

- **In-game assets**: Depending on the game, in-game assets such as skins, wearables, cars, and other gadgets, which usually are bought to gain an in-game comparative advantage, represent a sunk cost. This due to the fact that once bought they could not be resold. However, this is changing with NFTs. In-game assets can not only be owned but also traded; opening up a new economic sector.

- **Tickets & Certificates**: Tickets and certificates can be generated as NFTs to further eliminate counterfeits. The smart contract that runs on the blockchain can be programmed in a way that a minimum price goes towards the organizers and the surplus to the artists.

- **Virtual estates**: NFTs can further represent virtual or physical land. They are transforming the real estate industry: from fractionalizing physical properties for ownership and trading, to virtual land in the Metaverse.

Among all of the above mentioned use cases for NFTs, at the time of writing the collectible category is without a doubt the most common.

What is a dynamic NFT?

The next step in the evolution of NFTs are the dynamic NFTs (dNFTs). A dNFT's metadata is not fixed and can change. This can be used to adapt and change in response to external real-time events and data inputs from oracles such as the one from Chainlink Labs. Put simply, dynamic in this context refers to changes in the NFT's metadata triggered by a smart contract. This is done by including automatic changes within the NFT smart contract, which provide instructions as to when and how the metadata should change, e.g. from incoming information from oracles. Further, changes to the metadata are not the only dynamic elements of an NFT. For example, dNFTs can also be minted based on certain conditions, i.e., based on a predefined achievement by a soccer player such as amount of goal scored in one season.

Specific Use Cases of dNFTs

On one hand, the tokenID provides a permanent identifier for verifiable ownership. The metadata on the other hand is where the token's name is specified, traits are assigned, and file links are stored - housing the utility elements.

Below is a list - which by no means is exhaustive - of some potential dNFT use cases.

- **In-game character dNFT**: Collections such as the Bored Ape Yacht Club have a variety of traits, with some rarer than others. These traits are placed in the metadata of the NFT with a link to the IPFS where the corresponding image is stored off-chain. With dNFTs these traits can change and therefore be assigned to a different image. Later functionality is especially useful for in-game character progression. The in-game achievements and progressions are reflected in the updated metadata of the characters' NFTs which usually give the character further comparative advantages.

- **Tokenization of real-world assets**: Through the tokenization of real-world assets verifiable ownership is gained, however static NFTs do not reflect over time changes to the real-world asset. Take for example a tokenized house which over the years undergoes maintenance and market price fluctuations. Data which in static NFTs cannot be reflected, however through the use of dNFTs can be.

- **Digital, dynamic Collectibles**: Collectibles that are dynamic offer the collectors changes in the metadata based on real-world data inputs. For e.g. the famous soccer trading cards can through dNFTs represent the live stats of the world cup.

Legal and Regulatory

At the point of writing, it is not yet clear how to classify NFTs under current laws. In literature, the discussion under what legal construct NFTs shall be incorporated in, has raised controversials. There are voices that want to include NFTs under current property rights since they fulfill the main requirements of controllability and exclusivity. However, the majority voices in literature stand behind the fact that the numerus clausus of non-physical goods considered under property law is not to be amplified. Further discussions on the inclusion of NFTs under IP Law have also not reached a plausible conclusion.

The FINMA characterizes tokens, whether fungible or non-fungible in payment, utility, and security tokens, whereas some might be considered hybrid based on the original intention of the protocol and not on the possible usage. Therefore, a key point is to determine the nature of the NFT in question from a regulatory point of view. It is necessary to assess whether it is an alternative payment method (payment token), a mean of access/proof of ownership (utility token) or rather be considered as securities, i.e., typically having some sort of investment nature in the form of equity, bond or derivative, or are suitable for mass trading (security tokens). This distinction is essential to establish whether it is a regulated activity within the meaning of the financial market regulations and thus triggers compliance regulations such as FINMA licenses, anti-money laundering and sanctions regulations, among others.

Dynamic trading cards

NFTs as collectibles are used to showcase ownership of a digital asset, of which the owner is usually proud of. In order to keep this important characteristic of NFTs, we create digital trading cards of made-up football players of the fictional club *FC-Chainlink*. The collection of panini cards is well established in the football fanbase, especially surrounding the World Cup and European Cup. Our dNFT trading cards offer a digital collection and several perks compared to the physical cards. The dynamic part of the NFTs will be the players' Super League stats, including their played games and minutes and scorer points, which will be updated onto the NFTs. These stats will then be transferred into a point system. After the season there will be a raffle with various fan merch prizes and the more points a NFT has, the higher the chance of its owner to win is.

Market Analysis

Existing Approaches

NBA player LaMelo Ball has already published dynamic NFTs of himself. There are 8 different NFTs. They all have one specific stat that is upgraded, for example: points, blocks, assists, or a title like *rookie of the year*. They get updated through the whole career of LaMelo Ball. It is also issued on the Ethereum Blockchain and uses Chainlink's verified data feeds and VRF for verifiably random draws between the owners of the NFTs to select the winners of multiple prizes. [2]

In 2021 the Swiss football club YB created player's autographed cards as NFTs. Originally they sold them in bundles of three random NFTs for 98 CHF, but the NFTs of players bought after, in the winter transfer period, were sold individually for 33CHF. There are 50 tokens for each player. For their NFTs, YB is using the Flow Blockchain which is designed for NFTs and blockchain games.The digital contents the tokens refer to are not just pictures, but a 15 second video of a two sided trading card, with an abstracted player picture, signature and season number on one side, and a normal portrait, position, birthday, size, weight, jersey number and year since being a YB player on the other side [3]. The referenced video is not stored directly on the blockchain, but on the amazon cloud service *cloudfront*. YB has set up a point system for their players. They get points for dribblings, won tackles, assists, goals, played games and saves, with midfielders getting more points for goals and defenders getting more points for both goals and assists. There is a competition for NFT owners to collect the most points over all their NFTs, as the 10 people with most points win prizes from gift cards to season tickets. YB keeps track of the points and the owner ranking themself and displays it publicly.[4]

We also want to mention that Paniniamerica has created its own private ledger blockchain. They sell static NFT trading cards on their chain and they can also be auctioned off further. For all these transactions only USD can be used [5]. Using their own blockchain, allows Panini to hold the whole control over the infrastructure, rules out gas fees and offers basically unlimited scalability. But because the panini blockchain uses a private ledger, all it really offers are the pictures and the respective proof of ownership stored on their private servers with no ability to transfer them onto another platform. Therefore, all is dependent on Panini to continue their blockchain, which is a huge downfall in comparison to public ledgers. Hence, we think that Panini's solution of offering trading card NFTs on their own chain is not well suited for an NFT ecosystem.

Target Market

We aim at fans of football, especially those that have an interest in the collection of trading cards such as the world cup Panini cards. We speculate that this fascination for football player collectibles of the international cups can be transferred onto the club level and into the digital world. Further potential buyers can be general NFT collectors. But the fact that there is no specific limited amount of each NFT will lead to a small attraction of non-fan buyers, as the rarity is not known until the minting process is over.

Stakeholders

If this project is taken up by a football club, or even another sports club, the potential primary stakeholders are the football club issuing these NFTs and its fanbase. The club can gain an additional source of income, increase its contact points with the fanbase and hence strengthen its bond with them, increase its digital presence, and gain some initial strategic information about the whole blockchain technology. The fans on the other hand can collect trading cards of their club and even take part in a competition to win a raffle, while also for the first time being able to showcase their financial support digitally without sharing photos of their real-world merchandise. Further stakeholders are other clubs that are interested in participating in the NFT trend and wanting to join this project, as it would be technically possible to expand this product to more clubs, either in a separated but equal manner, or expanding the design, for example for a whole league.

Our Solution

We provide a complete on-chain solution, where even the NFT referenced images are saved on-chain. As of now, we think the best minting solution is to have an infinite number of mintable NFTs, but only in a limited timeframe, for example, the minting of new players stops, when the raffle starts. The price for the cards will for starters be given in ETH, specifically 0.001 ETH per card, which at the moment of writing would be around 3 CHF, and up to 300 cards can be bought at once. If a NFT trading card app is created for minting and trading the cards as well as displaying them, a minting price in CHF could be implemented relatively easily too with the Data Feeds of the current exchange rates provided by Chainlink Labs. This can be set at a similar price. We also provide the option that three of the same cards can be exchanged for an upgrade (more in detail later). As there are only 30 players, this seems like a fair price, although considering that the cards amount is theoretically unlimited and their upgradeability the optimal price could also be lower.

Minting Process

Inspired by the Panini trading cards, where the buyer does not know what cards they buy, we implemented a solution for a similar outcome. Since creating a fair random number on-chain is a very complex task, and to ensure that the user is not able to manipulate the system in such a way that they get the desired trading card, we use the VRF from Chainlink Labs. The users send an amount of ETH to buy the cards, e.g., 0.1 ETH for 100 trading cards, to the smart contract. This triggers the VRF. After a few blocks the VRF reveals the random number. This number is then used to airdrop the NFTs from truly random players to the users, as is visualized in Fig. 1.

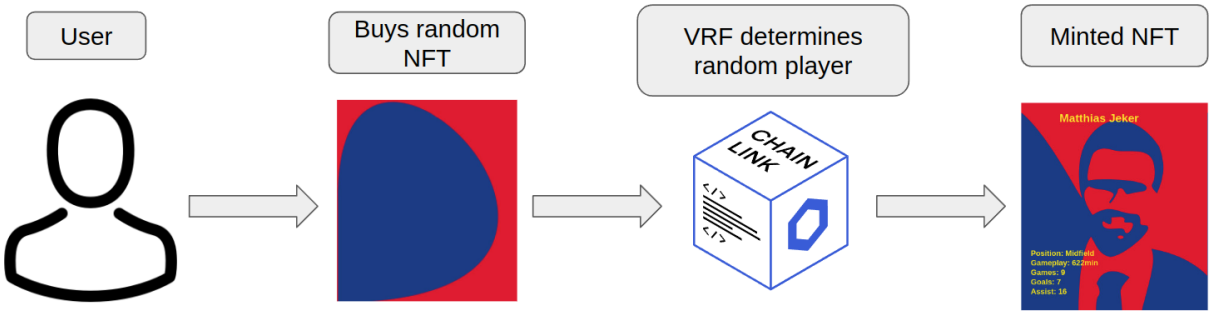

Figure 1: Demonstration of Minting Process

Updating Process

Football is an exciting sport and the seasonal performances vary between the players. The players' statistics can be looked up on certain sports websites. Since the player performance is dynamic during the season, e.g. the number of goals shot, or the number of minutes played changes over time, it is natural that the NFT of that player should adapt, as is shown in Fig. 2.

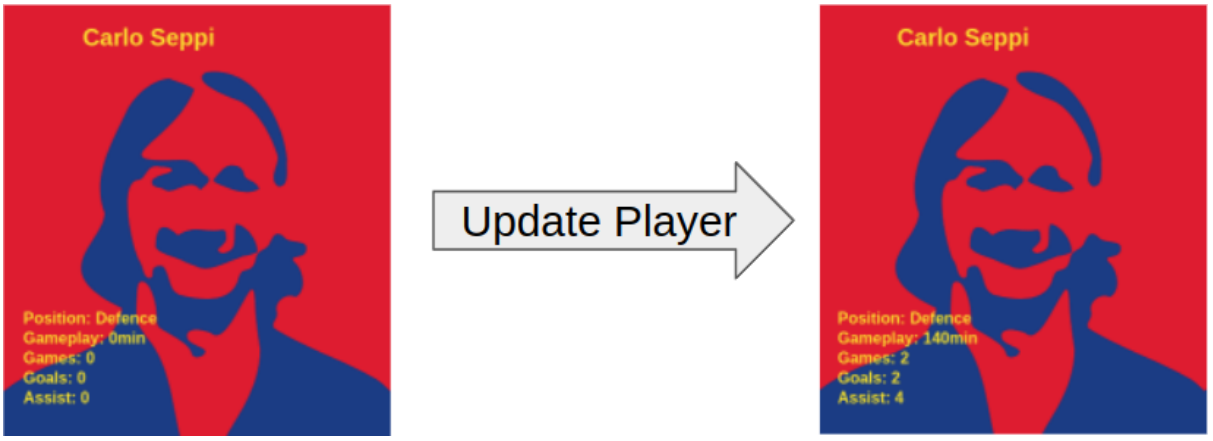

Figure 2: Updating the Player

The updates can be done by a person who is manually checking the statistics of the players and changes them manually, which is tedious work, and the person in charge could manipulate the statistics in their favor. We envision a fully automated process, where the trading cards are automatically updated. Luckily, Chainlink Labs provides a solution to that problem. With the help of the Keeper, we can implement that after a given time, the statistics of the players are updated automatically. In Fig. 3 we show how this can be achieved. The Keeper checks how much time has passed. In our implementation, if seven days have passed, it triggers the Oracle. The Oracle receives the updated statistics of the players, and updates the metadata of the NFTs. Chainlink Lab's decentralized oracle would be an ideal data provider, since it takes multiple sources into account and offers relatively high data integrity. However, due to the current limited availability of the decentralized oracle, Chainlink Labs provides an alternative. An API can be used to get the current statistics of the players. However, this can have a point of failure, since it will depend on the API provider, which could make a mistake or manipulate the statistics.

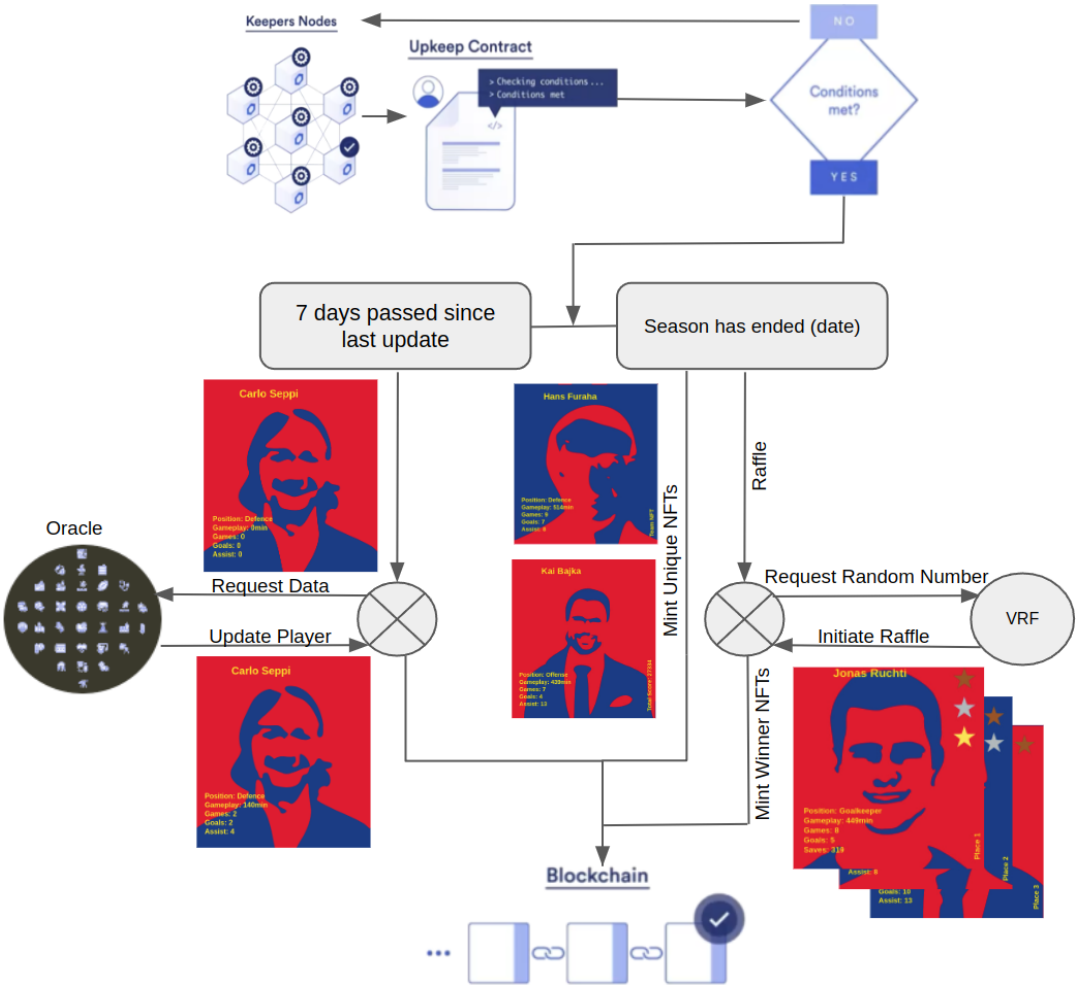

Figure 3: Update and Raffle process

Raffle system

At the end of the season, all the owners of the NFT trading cards will participate in a raffle. The winners of the raffle can win fan merchandise, like signed jerseys, boots, Keeper gloves, tickets, lounge seats, a specific ball which was used to score a goal and memorable events such as lunch with a player, or participating in a practice, as winnings. Every owner of such a NFT has a chance to win a prize, but the probabilities will differ. The more successful the player behind a NFT is during the season, the more points they collect, and therefore its owner's chances increase. There are several possible implementations of point systems:
Only the played minutes could be counted, as this stat is high if the player is having a good season and low if not. This would be the simplest solution and be relatively fair for comparing different positions, although there is a tendency that attacking players are substituted more often than others. In addition, it leaves out some very important factors and in some positions the competition may be weaker which would also distort the system.
Alternatively, position specific contests could be held, for attack, midfield, defense and goalkeeper. But this approach has several disadvantages. There are different amounts of players in each position, for example the goalie position usually has almost no competition and substitutes for the first choice usually only have a chance in case of injuries or suspensions. Further, if the system is only incorporated in one team and not in several or even the whole league the amount of players per position is too small for an exciting raffle. We decided to use a more sophisticated measure to allow for a comparison of different positions, currently we use the following formula to determine the points:

$$1+\text{game played}+\left\lfloor\frac{\text{minutes played per game}}{10}\right\rfloor+\lambda\,(\text{goals}+\text{assist})+\left\lfloor\frac{\text{saves}}{10}\right\rfloor,$$

where λ is 2 when the player is defense and 1 otherwise.

Such a self made formula aims at incorporating the most important stats, while not favoring any positions, but especially the second part is very hard to achieve. We analyzed former seasons of FC Basel with different formulas to try to achieve one that is as fair as possible while still being easy to understand and using natural numbers, as solidity does not allow for decimals. The basis for every player is 1, so each card has at least one point in the end, even if the player did not play throughout the whole season. In table 2 the most successful players in each position over the last three complete seasons are displayed in order to demonstrate the relative fairness between the positions in a condensed way. The statistics used for the point calculation are exclusively from the Swiss Super League, hence not counting any friendly, cup, or European tournament games. This makes it possible to

compare stats over the years and also opens up the possibility to extend the solution from a club base onto the whole league as the played games would be the same for every club and comparability is warranted. If a club applied our solution, it could potentially create a separate collection for each tournament it participates in.

	Season 18/19		Season 19/20		Season 20/21	
Position	**Player**	**Points**	**Player**	**Points**	**Player**	**Points**
Goalkeeper	Omlin	45	Omlin	52	Lindner	52
Defense	Widmer	56	Widmer	53	Petretta	48
Midfield	Zuffi	56	Frei	60	Kasami	56
Offense	Ajeti	62	Ademi	49	Cabral	63

Table 2: Most Successful players of seasons 18/19, 19/20, 20/21 [6]

Our NFTs will be based on the ERC-1155 standard, because we will create several identical NFTs of each player that will be fungible to one another. This allows updating all the same tokens of a player at the same time, which is much more efficient than updating each ERC-721 token individually. We also implemented the possibility to merge several identical NFTs to an improved version, with bronze, silver and gold stars, as shown in Fig. 4, which can only be implemented efficiently with the 1155 token standard. The conversion rate is displayed in table 3.

Figure 4: Original trading cards and the three upgrades

	Standart	Bronze	Silver	Gold

Standard Equivalent	1	3	9	27
Point Factor	1	4	15	50
Example Points	50	200	750	2’500

Table 3: Conversion rates

Again, all of this process should be automated. As shown in Fig. 3, the Keeper can trigger the raffle at the end of the season. This will do the following: users can not buy any new trading cards, nor can they upgrade them. In a second step all the owners will get a unique NFT, displaying the total score of all the owned cards. If they own the whole team, they will get an additional unique Team NFT, which is an incentive to collect all the players, just like with the physical trading cards. In a final step, the raffle triggers the VRF to get a fair random number. After some blocks the random number is revealed and used to choose the lucky winners. In theory, it can be implemented that each user (address) can win a maximum one prize but the users could easily spread their cards over multiple addresses increasing their winning chances.

User Interface Mockup

For the trading cards to have any utility for the collectors they must be conveniently displayable. Therefore we have set up an intuitive app mockup which allows for only some core functions. A collector has to set up an account and connect with a crypto wallet. They then have the options of minting new random players or buying them on a secondary market directly inside the app. If a user owns some cards, they are displayed inside their portfolio. They can sell their tokens through the secondary market as well, or start trading the cards just as with the traditional physical panini cards. The process of upgrading and updating the cards is similar to the initial purchase, as it has to be initiated with a transaction to the same address, but with an amount of zero ETH, which will cost some gas fees though.

Blockchain Decision

We will create our NFTs on the Ethereum blockchain. As we want to implement the verifiable random function (VRF) from our partner company Chainlink Labs we are limited to either Ethereum, the Binance Smart Chain (BSC) or Polygon. Our decision in favor of the Ethereum chain is based on several reasons. When we first started with the project in February the VRF was not available on Polygon, so we only had the opportunity to choose between Ethereum and BSC [7]. As of May 6th 2022 Ethereum is the largest blockchain, hosting 632 protocols which is about 250 more than the BSC has to offer, and a total value

locked of 143 billion USD being about ten times as large as that of BSC [8]. But more importantly the proof-of-staked-authority consensus mechanism is a large drawback for the BSC chain, as it only uses the 21 addresses with the most BNB tokens as validators which introduces some centralization to the blockchain which does not exist with Ethereum's Proof-of-work nor the coming Proof-of-Stake mechanism. Ethereum's downside is the high transaction fees, but with the coming merge, it is anticipated that the network will be able to handle transactions much faster and therefore gas fees will decrease sharply [9]. But because the BSC is originally a fork of Ethereum, a change of chains can be done relatively easily if it would be needed, for example to dodge the high fees.

Advantages

NFTs as collectibles have several advantages. They give fans the ability to openly show their support in the form of a collectible which the owner can be proud of. It makes it possible to sell the digital ownership of merchandise, therefore only a true supporter of the club who also is willing to spend money for it can showcase these digital merchandise. So the club can open a new channel for merchandise sales while fans get the opportunity to showcase their support in the digital world.

We compare our trading card tokens and raffle system to YB's as they have several similarities but simultaneously many strong differences. Both solutions are trading cards for football players of one club and both have implemented a point system for the players, coupled with a competition between the token owners for prizes. The advantages of our solution is that everything happens on-chain and is transparent as well as verifiable. The random minting process is verifiable thanks to Chainlink's VRF. The digital content is saved on-chain, ruling out that it can someday be taken down or something similar, which could happen to off-chain saved content like YB's videos. The significant seasonal statistics and raffle points are saved on-chain as well as the random function for the drawings of the raffle winners, making the whole process transparent and therefore almost impossible to be cheated. The fact that the competition is not won by the people collecting the most points, but holding a raffle makes it much more exciting in our opinion, but this point can be disputed. A further advantage is the dynamic aspect making our NFTs much more unique as they can differ strongly between seasons compared to YB's which only change the season number, if they keep their general design.

Technical Implementation

We provide the github code, and an instruction on how to buy the trading cards. As previously mentioned, we provide a completely on-chain solution of the NFTs. We use Matthias Nadler's code [10] as an infrastructure to save the NFTs on-chain. First, we needed to convert the images into an SVG and save the crucial information from the SVG on-chain.

In figure 5 we visualize how we convert an image (left) is converted into a SVG image (center). From the SVG we use the metadata how the image is created, namely, '*<path .. />*', and save these in the contract. We converted five images of us, and integrated them in a contract. These can be seen in the file *player.sol*. This ensures that the contracts do not exceed the limited space. An exemplary contract can be seen in Code 1. The first part adds the SVG information of the image. In addition, we add here, if the player has some upgrades or not. For each payer, a separate contract has to be deployed.

Figure 5: Processing of an Image to save on-chain.

```
contract Player0 {
  function getsvg(uint level) external pure
                  returns(string memory svg) {// svg image information
     svg = '<path style="fill:#0f3b86" d=".."/>';
…}
```

Code 1: Contract with svg information

The second step is building the SVG in such a way that it corresponds to the desired image for our NFT. Depending on the properties and the statistics of the player, the SVG should look different, e.g. having zero, one, two, or three stars on it, what is the number of goals the player has scored, and obviously which player is shown on the NFT.

To understand how the SVG is put together, we need to take a closer look at the file: *getPlayerSvg.sol*. In the file it has two contracts. The first one is *PlayerDetail* (Code 2), where the structure *playerDetail* stores the detailed information of the player. Here, the information like the name of the player, the number of goals they scored, etc. are stored.

```
contract PlayerDetail {
      struct playerDetail {... // Informations for the NFT }}
```

Code 2: Contract with name, position, and statistic of a player.

The contract *PlayerDetail* is inherited by the second contract *getPlayerSvg* (Code 3) and by the contract *createNFT* (Code 4) . Since both work with the same player detail and the two contracts will communicate with each other, they need to have the same structure to save the information of the players, but more on that later.

The contract *getPlayerSvg* returns an SVG compatible string. This string will be used as part of function *uri*, which will create the detailed information of the metadata for each NFT, e.g., the SVG image, the statistics of the player. To do so, the contract will need to communicate with the previous mentioned contracts. Therefore, we add each contract name, e.g. *Player0*, into this contract and the constructor adds the smart contract address to the corresponding contract. This enables the *getPlayerSvg* contract to access the *Player* contracts. That will be needed in the function *getSVG,* which returns an SVG-compatible string. How this string is constructed depends on what player is given, and what properties should be shown.

```
contract getPlayerSvg is PlayerDetail {
  Player0 player0;
  ...
  //  player.sol contracts, where the svg image of the players saved
  constructor(address _Player0, ...) {
      player0 = Player0(_Player0);
      ... }
  function getSVG(playerDetail calldata _details, ...)
                      external view returns (string memory svg) { ... }
}
```

Code 3: SVG image constructor for the NFTs

Note that the contracts we described are just the building blocks, so that we can get a string that is SVG-compatible and can be used as a on-chain reference for the *uri* function of the ERC1155 contract. Since all of informations are stored on-chain, we do not require IPFS or any other external server to access our images. With this infrastructure, we can in fact create

multiple different NFTs collection accessing these SVGs. If this is not be desired, then the rights to access the contract can be limited with an allow list of contracts (not implemented, but can be easily added). The token IDs from 0-19 represents a player with 0 to 3 stars. We like to have for all the situation an dynamic NFT (even when the season has ended). Since the statistic will not update anymore after the raffle, we added a small perk for all the unique NFTs (Place 1-3, Team NFT, and Score NFT): the image is dynamic. Depending on the current block.timestamp it will use a random image of one of the player, each time the metadata is updated.

The next contract we describe is *createNFT* from the file *createNFT.sol*. This contract inherits the contract *PlayerDetail* and *ERC1155.* We note that we had to do a slight modification on the ERC-1155 token standard from the openzeppelin libraries [11] and saved it as *ERC1155_edited.sol*. To know the token owners addresses at the start of the raffle, we need a mapping that keeps track, what addresses had at some point contact with the NFT(either through minting or transfering the NFT), namely _existenceOfAddress, and also a function which will return the list (Code 4).

```
contract ERC1155 is ... { ...
   address[] private _address; // all the people who own(ed) the NFT
   function _getAddresses() internal
                    view returns (address[] memory nft_address) {
        Nft_address = _address;} ... }
```

Code 4: Modified ERC1155 contract

Now we set up the ERC-1155 NFT with the contract *createNFT,* saved on the *createNFT.sol* file. We note that the name is shown on Opensea.org or looksrare.org, we have to add *string public name*. The details of the players are saved in an array which is added during the deployment of the contract. Again, we want to communicate with the *getPlayerSvg* contract to get the SVG for our NFT. The smart contract address is added in the deployment phase. We have two different modifiers which have control over the contract. *MintingProcess()* allows to define who will be able to interact with the contract, e.g., who can interact with the function *mintPlayer, burnPlayer,* and *getAddresses*. At the beginning it is the owner of the contract, but we will replace it with two different addresses of smart contracts (*Interface* and *mintingProcess*, which will be explained in further detail later). This contract uses the internal functions *_mint(...), _burn(...)*, and our added function *_getAddresses()* from the *ERC1155* contract. In most ERC-1155 NFTs, the ipfs or the html link is given to the location of the json file containing the detailed description of the name of the nft, image location, attributes, etc.

We do all of that on-chain, and therefore had to modify the function *uri(...)*. We note that to do this we need the *library Base64*. This can be also found in the *createNFT.sol* file.

There are other functions in the contract, like *updatePlayer* which the Oracle can use to update the statistics of the players. Also, some additional function to determine the raffle scores of the players and the total score of the NFT owners for the raffle.

```
contract createNFT is ERC1155, PlayerDetail {
  string public name = "FC-Chainlink"; // Name of Collection (Opensea)
  ...
  // gives a new index for each player details (format: structure)
  playerDetail[] _playerDetail;
  getPlayerSvg getPlayer; // comunicates with getPlayerSvg.sol
  address[2] contractOfMintingProcess; // allow list

  constructor(address _getPlayerSVG)  ERC1155("FC-Chainlink")  {
      getPlayer = getPlayerSvg(_getPlayerSVG);
      _addPlayer("player0","Goalkeeper",0,0,0,0,0,"Carlo Seppi"); ...}
  modifier mintingProcess() { ... }
  function changeMintingProcess(...) external onlyOwner() { ... }

  function mintPlayer(...) external mintingProcess(){...}
  function burnPlayer(...) external mintingProcess(){...}
  function getAddresses() external view  mintingProcess()
                   returns (address[] memory nft_addresses)  {...}

  function uri(uint tokenId) public view override
                       returns (string memory output) { ...
      // get details of player that should be shown
      playerDetail memory _details = _playerDetail[_tokenId];
      // get svg for the image
      string memory svg = getPlayer.getSVG(_details, ... );
      // create "json file" online
      string memory json = string(abi.encodePacked('{"name": "',...
            ' "description": "', ...
            ' "image": "'
            ' ,Base64.encode(bytes(svg)), ...
            ' "attributes": [...]'));
      output = string(abi.encodePacked('data:application/json;base64,',
                                       Base64.encode(bytes(json))));  }
...}
```

Code 5: Infrastructure for the ERC1155 NFTs

The contract *mintingProcess* in the file *mintingProcess.sol* uses the verifiable random generator VRF from Chainlink, so the contract inherits the contract *VRFConsumerBaseV2.* In this contract we have to add the settings for the VRF, which can be found here: https://chain.link/Chainlink-vrf.

In the constructor of the contract *mintingProcess* (Code 6), we add the address from the *createNFT* contract, so the contract *mintingProcess* can execute functions on the *createNFT* contract. We note that the *createNFT* contract has to add the contract address from the *mintingProcess* to the allow list by executing the function `changeMintingProcess(...)`. In addition, we need to give the subscriptionId from the VRF which can be registered here: https://vrf.chain.link/. To get a random card, we will use the VRF. We get a random number from the VRF after 3 blocks, and use this number to mint one or more players. We also decided to integrate a method, in which we can use an unfair random number, in case we do not want to use the VRF. The minting process for both are done in the function *buyPlayer.* If *vrf=true* is chosen, after 3 blocks the VRF node automatically executes the function *fulfillRandomWords(...)*. It is similar, when finding the winners of the raffel with the function. First it distributes all the unique NFTs, namely all the owner get an NFT with their total score on it, and if the owner has the whole team they get a team NFT. If *vrf=true,* a fair random number is chosen and after three blocks *fullfillRandomWords(...)* executes the raffle with the random number. If the raffel starts without a delay, an unfair random number is chosen , and it is possible to manipulate the raffle in such a way, that the attacker can has the desired outcome. Again, we have a modifier *interfaceAddress(),* which we will need to grant the contract *Interface* access to this contract. Fig. 4 visualizes how the VRF is used to get a random card.

```
contract mintingProcess is VRFConsumerBaseV2 { ...
  constructor(uint64 subscriptionId, address _createNFT_address)
                                VRFConsumerBaseV2(vrfCoordinator) {...}
  function buyPlayer(... , bool vrf) external  interfaceAddress() {
      if (vrf == true){//use vrf: run fulfillRandomWords after 3 Blocks
            s_requestId = COORDINATOR.requestRandomWords(...);}
      if (vrf==false) { // simulate vrf
            s_randomWords = block.timestamp;
            _buyPlayer();}  ... }
  function drawWinnerOfRaffel(...) { ...
      // Every owner gets unique NFT with the total score
```

```
      // When the owner has the whole team: get unique Team NFT
      if (vrf == true) {//use vrf: run fulfillRandomWords after 3 Blocks
            s_requestId = COORDINATOR.requestRandomWords(...);}
      else { s_randomWords = block.timestamp;  // simulate vrf
             _drawWinnerOfRaffel();} }
  function fulfillRandomWords(...) internal override { …
      if (numberOfPlayer[mintingCounter]==0) { // Raffel
            _drawWinnerOfRaffel();}
      else { _buyPlayer(); }   }
  modifier interfaceAddress() {...}
  function changeInterfaceAddress(...) external onlyOwner() {...}
}
```

Code 6: Minting and Raffle contract using VRF

To buy or upgrade the NFTs and to automatically update them, we added the smart contract *Interface* in the file interface.sol, so that the *Interface contract can* communicate with the smart contract *mintingProcess* and *createNFT,* and the address is added in the constructor. We also have to give the contract *Interface* permission to interact with the contracts *mintingProcess* and *createNFT*. In addition, three bools are set, deciding if the vrf, keeper, and the oracle should be used, respectively. Currently only a simulation of the oracle works, because we deploy our contract on the Rinkeby Testnet and decentralized sports oracles are currently only on Kovan Testnet. During the deployment it can be set, after how many minutes the oracle should update the NFTs and after how many minutes after deployment the Raffle should start.

We decided to use *receive(),* to buy and upgrade the NFTs. This function allows a user to send ETH to the contract. This will trigger the minting process of the NFTs. Sending an empty transaction will upgrade all the players to the maximum possible status. We decided to implement it in such a way, so that the user can directly see the Tokens on opensea, and it is a very easy interaction with the smart contract. We also added a *withdraw* function for the owner of the contract to withdraw the funds from the contract. The contract *Interface* inherits *KeeperCompatibleInterface,* allowing for a registered smart contract address on https://keepers.chain.link/ to interact with the smart contract. The Keepers are off-chain nodes, which check some condition in the function *checkUpkeep(...),* to check if the conditions in Fig. 7 are fulfilled. If they are fulfilled, a function called *performUpkeep(...)* is executed.

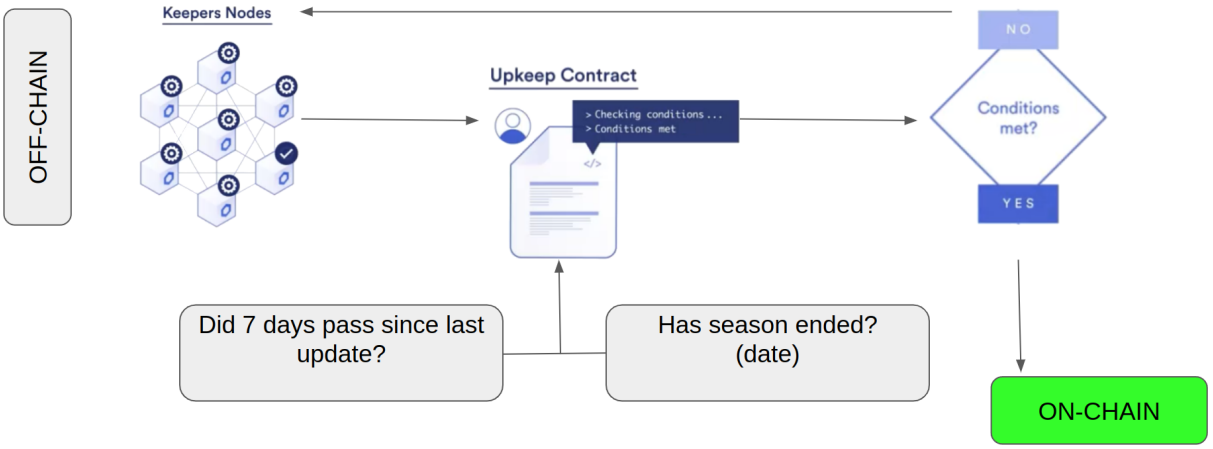

Figure 6: Keeper off-chain functions

If the *seven days since the last update* condition is met, *_updatePlayer()* gets executed which updates the player's statistics, which can be seen in figure 4.

In case the season has ended (or the given time to raffle has surpassed), the Keepers will execute *_createDistributionForRaffel(),* which will mint all the unique NFTs with the owners total score on them, and will draw a winner, as is visualized in figure 4. This is done when the contract *Interface* interacts with the contract *mintingProcess.* When the raffel begins, it will not be possible to buy any new players, nor is it possible to upgrade them.

```
contract Interface is KeeperCompatibleInterface {
      constructor(.... , bool _vrf, bool _keeper, bool _oracle,
                         uint _startRaffel, uint _checkOracle) {... }
      receive() external payable {
           if (msg.value>0) { _buyPlayer(msg.value, msg.sender);}
          else { _upgradeAllToMax(); }    }

      function withDraw(address _payout) external onlyOwner() { ... }

      // keeper check this condition
      function checkUpkeep(...) external view override
                                  returns (bool upkeepNeeded, ...) {
            upkeepNeeded  = _checkUpkeep();     }
      // will execute when upkeepNeeded=true (by keeper)
      function performUpkeep(bytes calldata performData )
                               external override {_performUpkeep(); }
      function _updatePlayer() internal { ...
        // Currently simulation: add random numbers to the players
        // here it can be added, so that it interacts with the Oracle }
      function _createDistributionForRaffel() internal {
         // distribution of total score NFT (unique) and chance to Win
```

```
        // Find the players who own the whole team
        ...
      // mint unique NFT and start raffle on contract mintingProcess
      _mintingProcess.drawWinnerOfRaffel(...);  }
```

Code 7: The user interacts with this contract, while the keeper automates everything.

We added throught the different contracts events. This events were added, so that the integration to web3 is easier. We note however, that currently we do not need a web3 interface to buy the players or upgrade them, because these function can be triggered by sending ETH to the smart contract *Interface*.
We created an app, how we envision to interact with the trading cards. This can be found here.

Limitations and Recommendations

When the contract is initiated all of the processes are done automatically. However, there are some limitations to our solution. Using the VRF, Oracle and the Keeper costs LINK. Therefore, a person still needs to ensure that all of the services have sufficient LINK, so at no point the services will fail. In our case, it will fail, if too many people mint and upgrade their NFTs, and therefore drain the funds we provided. To avoid this, a Keeper can check the balance of each service, and if it falls under a certain threshold it will top up the balance. This can be achieved by using Uniswap that exchanges the ETH on the smart contract to LINK and distributes it automatically.

One of the biggest advantages can also be a disadvantage. Namely, that the images are saved on chain. If there are copyrights violations, or at one point the person on the image wants it removed, it is not possible anymore. This can be solved, by using an allow list, defining who can access the contract. Another solution is to code the contracts in a more dynamic way, allowing it to be changed after the deployment. The drawback is that the owner of the NFT needs to trust the owner of the contract, that they will still be able to look at the image corresponding to the NFT owned later on.

The current provider of the sports data feeds is currently not an Oracle, but an API service provided by Chainlink and the distributor of the API, e.g. sportsdata.io [12]. The service was only on the Kovan testnet and not the Rinkeby testnet. Therefore we decided not to integrate a working data feed in our solution, but to use some randomness to update the players. Another solution could have been Chainlink Labs collaborating with the swiss football league (SFL) for an easy integration of the data feeds. Or otherwise, that we can get access to the

API feed from the SFL directly, and use the AnyAPI [13] service from Chainlink Labs, to extract the needed information.

Insurance extension

An additional focus task for our group was to think of an oracle-based dynamic NFT use case for classic business applications, to showcase how these NFTs can be used for much more than collectibles like our trading cards. We investigate the use of NFTs to issue parametric insurance in a transparent and decentralized way while also making the insurance and its counterpart tradable.

Parametric Insurance 1

A specific business application for the dNFT is weather insurance. For example, an insurance against hail in the month of August in Basel implemented with two opposing dNFT mintable for fixed prices, say for both 1 ETH. One NFT has the condition *hail = true* in its metadata and the other one has *hail = false*. The paid price is locked in a smart contract and paid out after the period has ended, to the owners of the NFT with the true condition. It is checked after specified periods, e.g. after the month of August by a decentralized weather oracle and the information is saved on-chain at the same time a Keeper activates a function of the NFTs to check whether their condition is true and if so leading to the payout, as visualized in figure 3.

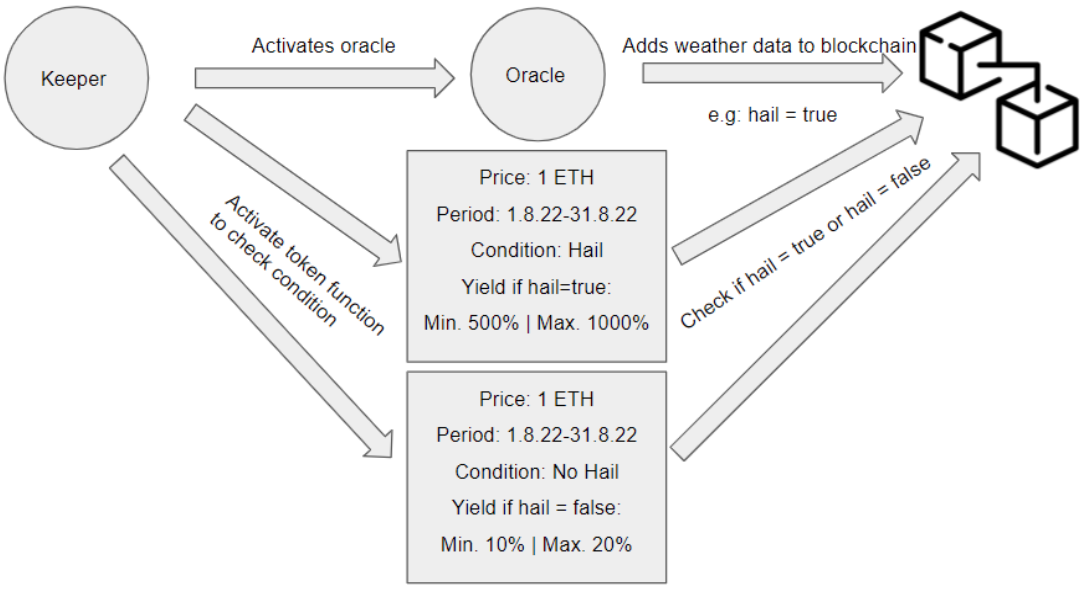

Figure 7: process of parametric insurance 1

There is an estimated probability for hail considering the same time period from previous years, for simplicity we assume a 10% probability for hail in Basel in August. Therefore, the no-hail NFT should be minted more often, leading to a distribution similar to the outcome probability as this resembles the expected payout, which is zero if the ratio is 1:9 and there are no transaction costs whatsoever. Therefore, the hail NFT owners have a much higher potential payout, but with a much smaller probability.

In order to enable people to insure against all possible outcomes this system can be implemented in such a way, that the party that wants to insure themself can specify the opposing conditions, for example more or equal to 200mm of precipitation in the month of August against less than 200mm (mean precipitation of August is 141mm) and then again these are mintable on a platform. This system could enable each party who looks for weather insurance to insure against a specific weather condition with a specific threshold they see as suitable for their need. For example, a farmer may know relatively exact at what amount of rain his harvest is endangered and therefore can mint NFTs to insure against too much or too little rain. They can also set a minimum and maximum return for each NFT, enabling a certainty for all potential investors. In our example in figure 4, when the first hail token is minted, there must be at least 5 no-hail buyers, so that the payout is realized in the end, otherwise a return of funds is initialized. But there is a maximum of ten no-hail holders per hail owner, as the yield if happening has to be at least 10%. When 8 no-hail tokens are minted the minimal yield if the condition is true of the hail token is doubled and therefore a second one can be minted, by the issuer themself or any other person wanting the exact same insurance.

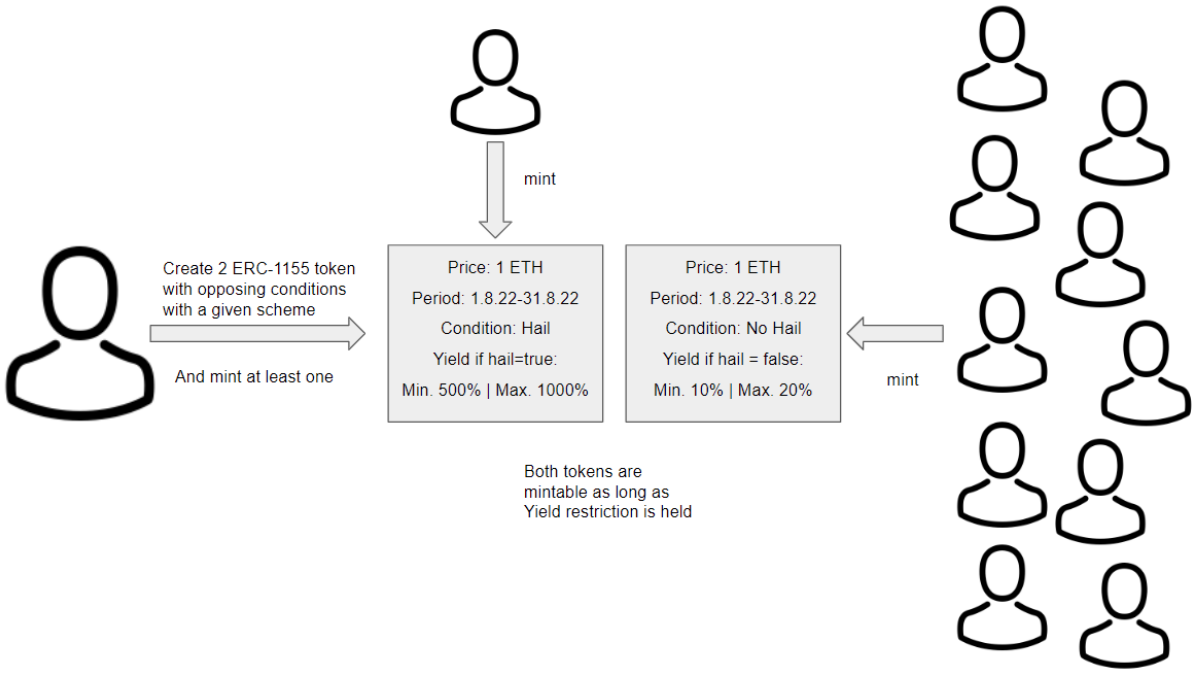

Figure 8: Minting process

If we assume zero transaction fees and a yield of comparable assets of zero and risk neutrality for investors, for every hail insurance NFT maximal nine counter NFTs can be minted, leading to a payout of net nine ETH for the farmer in case there is hail and his crop is at risk, compared to the cost of 1 ETH when his crop should be in good condition. Of course because there are transaction fees, and expected yields there will be minted less than 9 counter NFTs, which would be the cost of the insurance for the insured party. But we assume an almost perfect competition, as there should be no information asymmetries hence almost perfect knowledge and no discrimination thanks to the transparency of the public ledger, no hindrance to entre the market and therefore the number of investors should be sufficient. The different insurance NFTs would not be homogenous, as their conditions differ, but otherwise they are similar enough.

A clearly defined countable measure, such as the two mentioned examples, is needed in order to use this binary outcome as the payout condition, because the decentralized oracle network has to check the real-world data to update the condition on the NFTs. Besides the almost perfect competition, another advantage of non-fungible insurance tokens is that there is no counterparty risk, as the funds are locked in a smart contract. The tokens can also be traded after minting, which is an additional advantage. They would probably hold a similar value, only depending upon the yields in other assets, up to the point at which the weather can be predicted and from then on increase or decrease sharply based on the predictions and even the real weather.

Of course, there are also some disadvantages and problems associated with this insurance token system. The insurance is against an observable event such as weather and not the specific danger the insured party faces, such as crop loss, and therefore many insurances may be necessary to achieve the same protection as conventional insurance. Further, in this example, the longer the period considered for the condition, the fewer individual insurance tokens are needed, but with a longer period comes the possibility to have in a short period too much rain which is then equalized by no or little rain which could lead to destroyed crop but the insurance condition not being met. Also, as we described the system, if an insured event is happening in the beginning of a long period it takes a long time for the funds to be paid. But it could also be implemented that a Keeper initiates a check every so often and if the insured event happens or threshold is reached the payout is carried out.

As the funds used to mint the insurance tokens are locked, and therefore unproductive, in order to warrant the payout, this system is potentially better suited for short terms. Although

it could also be implemented that the locked funds are used to provide liquidity in a liquidity pool or staked or something similar for a specific period to increase the payout in the end.

For this insurance system we again envision the ERC-1155 standard to be the most suitable token standard, therefore, not being truly non-fungible, but fungible between the tokens with the same tokenID and the same condition in the metadata.

The described weather insurance process does not necessarily need dynamic NFTs, as all the core functions can be implemented with static tokens as well. But the dynamic aspect could add some interesting features, such as adding information about the payout directly onto the token. This could include if the NFT owner is eligible for a payout and how much it is.

This scheme could of course also be expanded into other parametric insurance areas with conditions observable by oracles, such as flight insurance, earthquake insurance or others. And also completely other business applications can be thought of that could profit from dynamic NFT products.

Parametric Insurance 2

A specific business application where we see a potential for the dNFT is parametric insurance, which means insurance for pre-defined sums upon arising of observable events. For an oracle-observable event, insurance tokens can be created (works with both ERC-721 and 1155). Such a parametric insurance could be made for the event of hail in specific periods over a specific region or piece of land. In such an instant the immutable data of the token contained the condition for a payout, the specific insured piece of land and the period. On the other hand the probability of the insured event, the paid price, and the owner address could be part of the mutable data. Such an on-chain parametric insurance could be used as an additional service by traditional insurance companies or it could even be implemented as a DAO, allowing for cost-reflecting insurance payouts as no returns would be necessary.

A party can insure a piece of land against a specific event such as hail. They mint a hail insurance token for their specific land. If they pay a certain amount to a specific address, their insurance token gets valid for the next month. The potential payout is the

paid price / the probability of event – insurance company fee or

paid price / the probability of event – small percentage to fill a coverage fund

When there are many people from all around the globe using this token, a pooling insurance is created as many parties facing differing weather conditions pay in and only a small number of parties receive payouts. Further, there needs to be some starter liquidity, which would be provided by an insurance agency or alternatively for the DAO structure a coverage fund should exist, which is filled at first perhaps by an investor and later with each insurance purchase. Such a fund could guarantee the payouts even if only a small number of users exist and the payouts exceed the payments in a specific period. To generate a fund in the beginning, an investor could be promised a small fee of each insurance purchase until his initial investment plus return is covered. After the payback, the fee could be used to fill the fund up to a certain limit and then be omitted if the fund is filled to a certain threshold. For better understanding some exemplary numbers will follow. Zero transaction costs and a full fund are assumed.

A party wants to insure its land against hail in the months of June, July, and August. The probability for hail being 10%, 10%, and 20% respectively. If they want a payout of 10 ETH in case of hail in each of these months, they will have to own a hail insurance token for their land and pay in the months of April, May, and June in order to get a valid insurance for the intended months. The payments would need to be 1ETH, 1ETH, and 2ETH to get a potential payout of 10ETH each month.

The dynamic part of the NFT would be data concerning the hail event probabilities for each month, provided by decentralized weather oracles and being calculated with the data for the specific period from several years before. Also, the condition for each month if the insurance has been bought and what the potential payout is. After each period the entitled parties get paid by a smart contract checking the weather data and the entitlement of the NFT.

If such a system is incorporated, there are costs for the decentralized oracle services, Keepers, transaction fees, and potentially protocol fees that accrue. But these costs have the potential to be considerably lower than traditional insurance costs, as the whole process can be automated and is relatively fraud secure.

At the same time there are several disadvantages and difficulties. The insurance is restricted to easily obtainable and verifiable data by oracles and therefore cannot be conditioned on damages but only such events. If it is implemented in the DAO context, there must be many parties from different areas insuring against the same event to diversify the risk.

The token itself would have no value, but only serve as a medium to show if the insurance is activated and how much is insured etc.

Conclusion

Dynamic NFTs open up new opportunities for the NFT collectibles, such as updating trading cards, implementing raffles based on points, etc. For business applications there still is the question why the NFT standards should be used, as anything these standards can offer, is also doable by smart contracts. In some instances a NFT solution could improve tradability, as the token could potentially be more easily sold compared to funds locked up in a smart contract.

References

[1] https://ipfs.io/

[2] https://opensea.io/collection/lamelo-ball-collectibles

[3]https://dkuxa1i6sgo8h.cloudfront.net/BSC-Young-Boys/BSCYB_NFT_21-22_Varga_Kevin.mp4

[4] https://www.bscyb.ch/nft

[5] https://www.paniniamerica.net/about-panini-blockchain

[6] https://www.sfl.ch/superleague/klubs/fc-basel-1893/

[7] https://blog.chain.link/vrf-v2-mainnet-launch/

[8] https://defillama.com/chains

[9] https://cryptomode.com/will-ethereum-2-0-lower-gas-fees/

[10] https://github.com/matnad/SVG-MVP

[11]https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC1155/ERC1155.sol

[12] https://sportsdata.io

[13] https://docs.chain.link/docs/make-a-http-get-request/