A Videogame Metaverse Based on the Polkadot Parachain Ecosystem

Seán Larkin
B.A.(Mod.) Computer Science and Business
Final Year Project 2022
Supervisor: Dr Donal O’Mahony

School of Computer Science and Statistics O’Reilly Institute, Trinity College,
Dublin 2, Ireland
Abstract

Blockchain technology was initially designed as a digital transaction medium, the first implementation of which was Bitcoin (Nakamoto, 2008). Since then, the range of functionalities that blockchain technology offers has only grown, as has the number of users of blockchain networks. This increase in functionality was sparked largely by the launch of the Ethereum blockchain in 2015 which built on the predominantly transactive capabilities of Bitcoin by enabling blockchain systems to run complex computer programs. Shortly after its launch, Ethereum’s original chief technical officer, Dr. Gavin Wood, was intrigued by a concept he believed would be an improvement on the Ethereum network, so he decided to leave Ethereum and pursue his vision for a blockchain that connected other blockchains. This would become the Polkadot network (Wood, 2016).

Polkadot was built to empower internet users to have ownership over their personal data, and it brings with it a set of brand-new possibilities for blockchain networks (Polkadot Lightpaper, 2022). The Polkadot network enables sovereign blockchains to connect and interact with one another seamlessly. Polkadot also massively reduces the economic cost of using a blockchain, making the network feasible for use by everyone at much greater scale than is possible with other blockchains.

The author believes that Polkadot’s purpose, as much as its technical performance, are what make the network ideal as a foundational pillar of the metaverse, and it is the goal of this project to examine the feasibility of using the Polkadot network and its features as an underpinning element of the metaverse. This is achieved by building a proof-of-concept system that uses Polkadot’s capabilities to allow users to exchange video-game based items for other items of value, be they in-game items or items in the real-world.
Acknowledgements

I first of all wish to wholeheartedly thank my supervisor, Dr. Donal O’Mahony for his incredible patience, persistent guidance and support as this project was undertaken. Without him the opportunity to research in this fascinating area would not have existed, and I would never have developed the tremendous passion for the field that I have over the past year.

I would also like to thank my peers who share a similar passion for blockchain and metaverse technology as I do, and who continued to encourage me in the pursuit of this work. Stephen, Ethan, Lola, Chris, and Adam without your contagious enthusiasm for the application of this technology I would not have learned nearly as much as I have done in the past year, or have had half as much fun doing it.

Finally, my incredible family, who were always around to support me through the inevitable moments of difficulty and frustration that comes with an undertaking such as this.

Thank you all sincerely.
# Table of Contents

**Chapter 1 | Introduction** ................................................................................................. 1

1.1 Motivation .................................................................................................................. 1

1.2 Approach .................................................................................................................... 2

1.3 Objectives .................................................................................................................. 2

1.4 Report Structure ....................................................................................................... 3

**Chapter 2 | Background** ............................................................................................. 4

2.1 Blockchain .................................................................................................................. 4

2.1.1 Outline .................................................................................................................. 4

2.1.2 Key Characteristics of a Blockchain ..................................................................... 5

2.1.3 Consensus Mechanisms ....................................................................................... 8

2.1.4 Value Exchange through Blockchains ................................................................. 13

2.2 Ethereum .................................................................................................................. 14

2.2.1 Outline .................................................................................................................. 14

2.2.2 Smart Contracts ..................................................................................................... 15

2.2.3 Token Types .......................................................................................................... 18

2.2.4 Transactions and Messages .................................................................................. 21

2.2.5 Ethereum Evolving ............................................................................................... 24

2.2.6 Decentralised Applications ................................................................................... 25

2.3 Non-fungible Tokens ............................................................................................... 27

2.3.1 Outline .................................................................................................................. 27

2.3.2 Key NFT Types ....................................................................................................... 28

2.3.3 Exchanges .............................................................................................................. 31

2.3.4 Royalties ............................................................................................................... 32

2.3.5 Uses of NFTs ......................................................................................................... 33

2.3.6 Issues with NFTs ................................................................................................. 35
### Chapter 4 | Technical Implementation

- **4.1 Overview** ........................................... 74
- **4.2 Technology Used** .................................. 75
  - 4.2.1 Runtime Environments and Programming Languages .................. 75
  - 4.2.2 Frameworks, Libraries and Services ................................ 76
  - 4.2.3 Videogames ........................................ 77
- **4.3 Minecraft Item Base** ................................ 78
- **4.4 Item Access Control** ................................ 91
- **4.5 Item Core and Item Core Interface** ............................ 98
- **4.6 Results of Proof-of-Concept Implementation** ...................... 104

### Chapter 5 | Evaluation

- **5.1 Proof-of-Concept Challenges** ................................ 115
- **5.2 Fulfilment of Requirements** ................................ 117
  - 5.2.1 Functional Requirements .................................. 117
  - 5.2.2 Non-functional Requirements ................................. 118
- **5.3 General Difficulties Encountered** ............................. 119

### Chapter 6 | Conclusion and Future Work

- **6.1 Conclusion** ........................................... 120
- **6.2 Future Work** .......................................... 121
Glossary

**Acala**: Decentralised finance platform on the Polkadot network

**Address**: Type in Solidity programming language

**API**: Application Programming Interface

**Array**: Data structure containing multiple items of the same type

**Avalanche Effect**: The breaking effect of editing one block’s data on the rest of the chain

**Bitcoin**: The first public blockchain, deployed in 2009

**Blockchain**: A linked-list style data structure that is cryptographically secure

**Canonical state**: True state

**Cipher**: Encrypted text

**Consensus**: Agreement

**Crypto-asset**: An asset held on a blockchain

**Cryptography**: The science of encrypting data

**Cryptocurrency**: A digital currency secured by blockchains and encryption algorithms

**Crypto-wallets**: Blockchain based wallets that hold cryptocurrency

**Data Provenance**: The history of how a piece data was created and used

**Decentralised Computing**: Computing that is distributed and uncontrolled by a single entity

**Decentralised Finance**: Financial systems operated through blockchain technologies

**Denial-of-service attack**: An attack that spams a network with requests, slowing it down

**DEV**: Moonbeam testnet’s native currency

**Dotsama**: An ecosystem comprising Polkadot and its sister network, Kusama

**EIP**: Ethereum Improvement Proposal

**ERC**: Ethereum Request for Comments

**Ether/ETH**: Ethereum’s cryptocurrency

**Ethereum**: The second blockchain to deploy after Bitcoin

**Ethereum Virtual Machine (EVM)**: The ability for Ethereum to run computer programs

**Etymology**: History of a word
**Fiat Currency:** Physical currencies like the US Dollar and the Euro

**Fungible:** Interchangeable

**GLMR:** Moonbeam mainnet’s native token

**Gwei:** Gigawei, 1 billion Wei

**Hash:** A string of letters and numbers generated from some data provided

**Heterogenous:** Diverse

**Homogenous:** Alike

**Immutable:** Unchangeable

**Key:** A unique identifier used to encrypt or decrypt data

**Kilt Protocol:** A Polkadot parachain supporting blockchain based identification

**Mainnet:** The live network of a blockchain

**Metaverse:** The digital universe, seen as an evolution of the internet

**Minting an NFT:** The creation of an NFT

**NBT Format:** Non-binary tag format

**NFT (Non-fungible token):** Cryptographically secure identifier of a unique digital asset

**Node:** A computer

**Nonce:** A number only used once

**On-chain:** On a blockchain

**Oracle:** An API that can securely connect blockchains to other internet addresses

**Permissionless Systems:** Systems that are open to everyone

**Polkadot:** A blockchain that connects other blockchains

**Polygon:** A popular blockchain network

**Private Key:** A key kept secret used to decrypt messages

**Promulgate:** To promote or make known

**Public Key:** An encryption key that is accessible by anyone

**Runtime Methods:** Functions invoked while a function is running

**SDK:** Software Development Kit

**Shard Chains:** A blockchain that is a component of a larger blockchain

**Smart Contract:** An Ethereum account type that can deploy computer code

**Solidity:** A programming language supported by Ethereum
Substrate: A programming language supported by Substrate

Testnet: A live blockchain network used for testing systems

Token burning: Destroying cryptocurrency tokens

Token minting: The creation of a crypto-token

Trustless/ Trust-free Systems: A system that does not require a third party intermediary, that act in a predictable manner, whose mechanisms are openly viewable

Uint256: An unsigned 256-bit integer

Web 3.0: The evolution of the internet involving decentralisation

Wei: The smallest denomination of ETH (1x10^{-18})

XCM: Cross-chain messaging standard on Polkadot

XCMP: Cross-chain messaging protocol on Polkadot

51% Attack: An attack on a blockchain network that involves controlling over half a blockchain network’s nodes.
Chapter 1 | Introduction

1.1 Motivation

The adoption of blockchain technology was growing rapidly in 2021. Cryptocurrencies and non-fungible tokens (NFTs) were garnering a huge amount of attention across social and mainstream media, promulgated largely by influencers such as Gary Vaynerchuk and Mark Cuban. The highly volatile nature of “crypto-assets” brought with it the potential for significant financial returns for investors (Leirvik, 2022) meaning blockchain technology’s use was largely focused on rapid wealth generation, with significant spikes in trading over the course of the COVID-19 pandemic (Guzmá, Pinto-Gutiérrez, and Trujillo, 2021). The author was broadly familiar with the concept of blockchain technology at this time, without truly understanding how it operated. Instead, the author’s fascination lay in the concept of a metaverse, which had been an interest of his from a young age, albeit without referring to the concept as a metaverse at that time. This fascination was further inspired by his viewing of Spielberg’s (2018) Ready Player One film. The plot of the film is centred around a metaverse in which the users partake in videogame-like competition to earn digital currency that they can then use to purchase unique items to equip onto their digital avatar, creating a digital item economy. The author had read that NFT technology could be used to represent unique digital items, and so wanted to explore how applicable this technology could be to creating such a digital item economy in an interoperable metaverse.

1.2 Approach

To begin with, this work will explain the technologies that will allow a metaverse-based digital item economy to function, in particular, blockchain and NFT technologies will be
discussed. A system architecture for a proof-of-concept implementation is then devised, while more complicated system architectures are also conceptualised. The proof-of-concept architecture is implemented, and its results displayed and evaluated before a conclusion is drawn and opportunities for future work in the area are considered.

1.3 Objectives

The overarching goal of this project was to use blockchain technology to examine the feasibility of a videogame-based metaverse implementation that would enable users to easily exchange unique in-game items across different in-game worlds, with verifiable proof of creation, ownership, and transaction history. Thus, adding real life-time value to videogame items.

To achieve this objective as set of subobjectives was assembled:

A system must be developed that enables:

- Real-world value to be attributed to in-game items.
- These items to be transferred from one owner to another.
- Newly acquired items to be used in a local game environment.
- An item’s state to change depending on how it is used in-game.

Furthermore, this work should:

- Evaluate the suitability of the architecture proposed to meet the system requirements.
- Discuss areas required for future development of the architecture.
1.4 Report Structure

Chapter 2 presents a review of the enabling technology for this work. The review begins by examining literature surrounding blockchain technology, which is the cornerstone of the work implemented in this report. Technology developments enabled by this underlying concept are then explored, before an analysis of the concept of a metaverse is performed.

Chapter 3 describes the design of the system presented herein. This involves the detailing of functional and non-functional requirements, and the outlining of a proof-of-concept system architecture designed to address these requirements. Limitations of the proof-of-concept architecture are touched upon before improved architectures are proposed.

Chapter 4 walks through the proof-of-concept implementation of the architecture discussed in the preceding chapter. The exact technology used in the implementation is described as is the rationale behind key design decisions. The functional results of the proof-of-concept are addressed at this stage alongside a prospective user journey.

Chapter 5 undertakes a thorough evaluation of the proof-of-concept implementation. This includes a description of challenges faced during implementation, and how they were overcome, as well as an evaluation of how the proof-of-concept addressed the requirements set out.

Chapter 6 brings the work to a conclusion. The author presents some personal reflections on the nature of the metaverse’s development going forward alongside a summary of the full report. The report concludes with the author’s remarks on opportunities for future work based on the learnings ascertained.
Chapter 2 | Background

2.1 Blockchain

2.1.1 Outline

In 2008, someone using the pseudonym Satoshi Nakamoto published a whitepaper outlining their vision for a “purely peer-to-peer version of electronic cash”. What Nakamoto had written about and provided the mathematics and logic behind was to become known as Bitcoin (Nakamoto, 2008). Bitcoin was built to provide a way to overcome the problem of electronic currencies being spent more than once, known as the double spending problem (Chohan, 2017).

Bitcoin overcame this issue by using a decentralised, peer-to-peer mechanism to keep account of and verify transactions, without the need for an intermediary such as a traditional financial institution. This mechanism is called a blockchain and today the terms blockchain and Bitcoin are almost synonymous with one another. Bitcoin is, however, not alone as the only public blockchain in operation, following its launch in 2009 there have been countless other blockchain networks developed and built upon, the most used of which currently is called Ethereum.

A blockchain is at its core a public database, that is managed across many computers on a network. They function as immutable, distributed ledgers of transactions that remove the need for a human intermediary to verify the validity of a transaction (Nofer et. al, 2017), thus allowing decentralised transactions to take place.
The term blockchain derives from the way in which they are structured. A “block” contains a batch of data, for example a list of transactions, as well as the cryptographic hash of the previous block in the “chain”, linking each block to its predecessor (Nie et. al, n.d.) in a linked-list like data structure. This attribute gives a blockchain its immutable state, as changing the data in any block changes its hash, which will therefore break the chain.

![Figure 2.1: Illustration of a Blockchain (Zheng et. Al, 2018)](image)

### 2.1.2 Key Characteristics of a Blockchain

**Cryptography and Keys**

Cryptography is an essential part of securing the state of a blockchain. There are two primary forms of cryptography that most blockchains make use of: Asymmetric-key cryptographic algorithms; and Hash functions (Singh, A., 2021).

There are a variety of different hash functions used across different blockchains to generate the hash of the data contained in each block. Many blockchains, for example Bitcoin, use derivatives of the SHA-256 algorithm that was originally used in HashCash (Back, 1997). Although, this is not the only hash algorithm in use, Ethereum for example uses a different 256-bit hash function, Keccak-256 for some of its operations (Vujicic, Jagodic and Randic, 2018).
By hashing the data contained within a block on the chain, and then including that block’s hash in the data to be hashed for the next block, tampering with the history of the chain is prevented as altering the data of one block, will change its hash and therefore cause an avalanche effect that breaks the links between blocks on the chain (Krithika and Rohini, 2020). It is through using these cryptographic hash functions that the history and state of a blockchain becomes immutable.

Asymmetric-key cryptography, also called public-key cryptography, uses an algorithm to generate a “private key” and its corresponding “public key”. The private key will generally be a randomly generated string of numbers, and then an irreversible algorithm is run to generate the public key that is associated with it (Singh, A., 2021).

Simply put, a user’s public key is openly viewable to the world and can be shared across the blockchain network. If a user, Aisling, wants to securely send another user, Barry, some data, that data is constructed using plain text initially, and it is then encrypted using Barry’s public key and sent. The only way this data can now be read is by decrypting it using Barry’s private key, which only he has access to (Fox, n.d.).

*Figure 2.2: Illustration of Asymmetric-key Cryptography*
Decentralised Systems

A system is referred to as distributed if its state is shared across multiple “nodes” or computers on a network (Waldo, Wyant, Wollrath, and Kendall, 1997). Blockchains are referred to as distributed networks for this reason, but more than just being distributed they are often labelled as decentralised networks.

It is relatively common for organisations to store data across multiple locations, and thus in a distributed manner. Many of these distributed systems, however, are controlled by a single centralised system and therefore that central system, such as a group of developers in a company, or account analysts in a financial institution, have ultimate power to determine what is the “true” state of the network and its data (Vatankhah Barenji et al., 2019).

Blockchain technology moves away from this central point of control to a decentralised point of truth. The state of the network is not controlled by a single entity, but rather by a cohort of independent nodes connected to the system, that communicate with each other to arrive at consensus on the state of the blockchain network (Xie et al., 2017). This decentralised nature of a blockchain, if implemented correctly, removes the need for individuals to trust third parties to handle or verify a transaction of any kind.

![Figure 2.3: Centralised, Distributed and Decentralised System Illustrated (Ahsan, 2018)](image)
2.1.3 Consensus Mechanisms

As mentioned above, the most common type of computer system used today is a centralised system, where the state of the system is decided and verified by a singular controlling entity (Hooda, 2021).

As decentralised systems, most public blockchains don’t have an individual controlling entity and so the nodes connected to the network must find a way to reach an agreement on what the true state of the system is. Blockchains use three primary methods to achieve this, which are called consensus mechanisms (Nair and Dorai, 2021). Proof-of-Work (PoW), Proof-of-Stake (PoS) and Proof-of-Authority (PoA) are the names given to these three methods.

Proof-of-Work Consensus

PoW consensus was the method implemented by both Bitcoin and Ethereum at their inception. Although, the method had found some practical implementation before its use in blockchain networks, being implemented in 1997 as a means of limiting denial-of-service attacks and email spam in a system known as HashCash (Back, 1997).

HashCash required a cryptographic puzzle to be solved prior to sending an email, thereby creating a small computational cost for sending a message, much like the cost of a stamp when sending a regular letter (Back, 2002). In short, HashCash operated by taking all the key data belonging to an email like date, recipient, the body of the email etcetera, and adding what is called a nonce (a random number that is only used once) to the end of it, before passing it to the hashing function. A “level of difficulty” was set for the function, which was to require the resultant hash to begin with a certain amount of zeros. The more zeros required
at the beginning of the hash, the greater the difficulty in solving the puzzle (Frankenfield and Anderson, 2021).

*Figure 2.4: Blockchain Hashing Visualised (Brownworth, n.d.)*

This same PoW method continues to be implemented by both Bitcoin and Ethereum which replace the ‘email data’ of the HashCash system with the transaction data contained in the latest block on the respective networks (Wackerow et al., 2022). Solving this puzzle to generate a new block for these chains became known as a process called “mining”, where each “miner” is a computer or “node” connected to and therefore maintaining the network while trying to solve the puzzle to generate the hash of the next block. Solving this puzzle earns the miner what is called a “block reward”, which is an economic reward in the form of the cryptocurrency native to its blockchain (Qian, Li, and Ma, 2021). The block reward system incites competition between nodes to solve the next puzzle, and then share the solution for the latest block with other nodes connected to the chain.
Blockchain networks try to keep the ‘block times’, the time it takes to generate a new block, constant. Constant block times can be achieved by increasing or decreasing the number of zeros required at the beginning of the next block’s hash (Hong, 2022), depending on how many nodes are running on the network. It follows logically that the greater the number of nodes trying to solve the puzzle, the faster the puzzle will be solved, thus decreasing the block times. When, therefore, there are a greater number of nodes connected to a blockchain, to maintain consistent block times, the network will heighten the difficulty of the puzzle by increasing the number of required zeros at the start of the hash. The opposite is true in the case that there are fewer nodes running on the blockchain.

When a new block is added by one node, all other honest nodes add it to their copies of the network’s state. In this way, PoW consensus removes the need to have a central decision maker to have network consensus (Frankenfield and Anderson, 2021). PoW also makes it incredibly difficult for mal-intentioned actors to break the system, as, if such an actor wished to tamper with some historical block data on the chain, they would need to re-hash all blocks after the one containing that data, which would be rejected by other nodes on the network unless the actor controlled 51% of the network, in which case this is called a 51% attack (Frankenfield, 2021).

PoW is not without flaw, however, as it necessitates a huge amount of energy to solve these problems consistently. For example, in 2021, the Bitcoin network alone used up more energy than the entire country of Sweden (Rushe, 2022).
Proof-of-Stake Consensus

The energy requirements of PoW consensus present an enormous problem both environmentally and for network scalability (Locke, 2022). So much so, that more recently developed blockchains, such as Polkadot, chose to implement a different consensus mechanism, PoS, Ethereum is also actively being updated to use a PoS consensus method for these reasons (Locke, 2022).

PoS block creation is generally not referred to as mining but rather as “minting” or “forging” (Wood, 2016). Nodes that wish to mint an upcoming block on the chain are called “validators”. Validator nodes lock-in, or “stake”, some of that network’s tokens and are then chosen, with a degree of randomness, to mint the next block. The larger the value of tokens at stake, the more likely that a validator is to be chosen to mint the next block on the chain. For example, a validator with one-thousand tokens staked is ten times more likely to be selected to mint the next block than a validator with one-hundred tokens staked, and thus is ten times more likely to receive a block reward. Should a validator be found to have acted maliciously, it will lose either some or all of its staked tokens depending on the severity of its misdemeanour in a process known as “burning”. As a result, as long as the value of the tokens staked is greater than that of the block reward, validators are incentivised to act honestly, ensuring the security of the chain.

Another type of actor in PoS consensus is called a “nominator”. The role of a nominator is much more simplistic than a validator, in that nominators’ sole function is to provide risk capital to validators in return for a proportional share of any block rewards earned. Multiple nominators can pool their resources together behind select validators, indicating heightened
trust of the validator while simultaneously making it more likely to be selected to mint a new block.

PoS based systems require significantly less energy to maintain the state of a blockchain than PoW systems, with some estimates stating that PoS uses as little as 1% of the energy that PoW consensus requires (Rushe, 2022). They also enable more individuals to benefit from block minting and block rewards by enabling nominators to stake relatively small amounts of value behind validator nodes, lowering the barrier to entry. PoS systems do still suffer a risk of a 51% attack, like PoW, but the risk factor is not significantly different from PoW consensus mechanisms (Frankenfield, 2021).

**Proof-of-Authority Consensus**

The PoA mechanism can be thought of as a method whereby a select number of nodes are assigned to act as “validators” for the network, which like in PoS, work to generate the next block to be added to the network (Valente, 2019).

The new block is then either added to the chain immediately without verification, or after a unanimous or majority vote by the other selected validator nodes. This significantly simplified block verification process, which removes the need for nodes to stake coins, means PoA based chains can process transactions rapidly with limited processing fees (Binance Academy, 2018).

PoA requires very little computing power as there is little to no competition between validators, and therefore far less energy is required to operate PoA in comparison to the
alternative methods, but it creates a very centralised system as only a select number of nodes ever control updates to the chain (Seth, 2021).

Hence it is favoured primarily by “private or consortium” blockchain networks (Valente, 2019) that require increased performance and its use cases in public blockchain networks are very limited both in number and scope.

2.1.4 Value Exchange through Blockchains

Blockchains provide a means of recording value exchanges without the need for a mediating third party (Skinner, 2016). The same way that value exchange in the Eurozone is not limited to people exchanging euros, value exchange on a blockchain network is not limited to people exchanging that network’s cryptocurrency tokens.

The public nature of blockchain protocols means that they can be used as a proof of record of any kind of digital value exchange for anything, ranging from a simple currency transaction to complicated contractual agreements such as leases, and theoretically any other exchange of value between parties that can be recorded digitally (Skinner, 2016).

Furthermore, when an asset is exchanged over a blockchain network, provenance of that asset’s data is ensured, meaning a blockchain tracks where an asset was created, where it was stored and where and when it was used (Wang, 2019). This in itself can influence the intrinsic value of an item. For example, a pair of football boots signed by Lionel Messi proves that those boots were once owned by someone of celebrity status, which positively influences prospective buyers’ purchase intentions (Newman, Diesendruck and Bloom, 2011).
2.2 Ethereum

2.2.1 Outline

The Ethereum blockchain network was launched on the 30\textsuperscript{th} of July 2015. Since then, its popularity has only grown, making it one of the most used blockchains in the world.

Ethereum has its own native cryptocurrency called Ether or ETH, but its key differentiator came in the form of smart contracts, which are effectively bundles of code that anyone can write, deploy and run on the Ethereum-Virtual-Machine (EVM) (Antonopoulos and Wood, 2019).

The EVM is technically one single entity that is maintained by thousands of connected, but distributed computers running an Ethereum client. At any given block, Ethereum has only one canonical state, and the EVM makes the rules for computing a new state from one block to the next. Ethereum’s use of smart contracts and the EVM are what make the blockchain more than just a distributed ledger, like Bitcoin, and transform it into a distributed state machine (Wackerow, Vianello and Ste, 2022).

Smart contracts enable an enormously diverse range of functionality on Ethereum, from standardised cryptocurrency, or more accurately “crypto-asset” token types, to unique transaction mechanisms and the ability to support decentralised applications or “dApps”.

In addition, Ethereum as a network is continuing its development. Currently, the blockchain uses a proof-of-work consensus mechanism to maintain its state across nodes, but this is due to be changed to a proof-of-stake consensus mechanism in the very near future which will enormously reduce the network’s energy consumption. Furthermore, Ethereum’s ability to scale and handle a greater number of transactions per second is being worked on and will be
deployed in the coming years to handle greater user traffic as it looks to lead the way for Web 3 technology into the future.

2.2.2 Smart Contracts

Ethereum’s use of smart contracts represented a significant innovation for blockchain technology, by enabling anyone to write a program that can be run on the EVM.

A smart contract is typically a collection of code and data (its functions and its state) (Douglas, 2022). They encode a set of conditions that when met, incur some action being taken (What are smart contracts on blockchain? | IBM, n.d.). They are immutable by default and cannot be deleted.

Smart contracts are also technically a type of account on the Ethereum network, and as such each contract is identifiable by a specific address. Given a smart contract’s nature as a computer program they are not controlled by a user, they simply run as programmed. Other Ethereum accounts, be they user accounts or other smart contracts, can interact with a deployed smart contract by invoking a transaction to that contract’s address (Douglas, 2022).
Ethereum smart contracts are also permissionless (Douglas, 2022), meaning there are no special requirements or privileges needed to write and deploy a smart contract. Any individual can deploy a contract to the Ethereum network, the only barriers to doing so are the ability to write in an Ethereum-based programming language, like Solidity, and “Gas fees” which are discussed in more detail in section 2.2.4, but are essentially the cost of making a change to the state of the blockchain.

A smart contract can in a sense be thought of as an open API, one can be invoked by any person or computer seeking to utilise its functionality or view any data stored within it. However, smart contracts cannot themselves make API calls to any source that is not on that contract’s blockchain.

The reason for this is to do with consensus and the deterministic nature of a blockchain network (Smith, 2022). To explain, if a smart contract is run from a particular node at a particular instant in time and, for example, tries to utilise Google’s API to reference the score of a sports game, when other nodes try to reach consensus on the state of the chain several seconds, minutes or even hours later, they too must run this smart contract calling Google’s
API. A problem now arises such that if the score of the game has changed in the interval time between those two API calls, the two nodes will not be able to agree on the chain’s state, as both will be returning different scores from the game through no fault of their own.

The solution to this issue comes through technology called “Oracles” which act as intermediaries between the on-chain and off-chain worlds (What Is an Oracle in Blockchain? Explained | Chainlink, 2021). Succinctly put, oracles can make API calls to off-chain sources, and then enter the off-chain data to the blockchain via a simple external transaction, meaning every node will read the same data value and can therefore achieve consensus.

Figure 2.6: Oracle Framework Visualisation (Mammadzada, et al., 2020)
2.2.3 Token Types

The Ethereum network consists of three primary types of token, referred to as token standards ERC-20, ERC-721 and ERC-1155, respectively. Each of these standards has different characteristics, abilities and use cases.

ERC-20

ERC-20 (Ethereum Request for Comments 20) was proposed very early in Ethereum’s development by Vogelsteller and Buterin. Its aim was to produce a standard for fungible tokens that would allow any Ethereum based tokens to be reused by different applications (Vogelsteller and Buterin, 2015).

ERC-20 tokens could be used to represent anything from financial assets like a share in a company, a custom cryptocurrency, or even the skills of a character in game (Wackerow, Scarset and Smith, 2021). Its nature as a fungible token standard, means that one ERC-20 token of a particular denomination can be exchanged for another ERC-20 token of the same denomination and neither party will be better or worse off for it.

Fiat currencies like the euro are fungible (Bhalla, n.d.), for instance, if Aisling gave Barry a one-euro coin and Barry gave Aisling a one-euro coin both parties will have exchanged the same amount of value and as such neither party is left at a benefit or a loss.
ERC-721 is another token standard that actually evolved from the ERC-20 standard, but unlike ERC-20, ERC-721 standard tokens are what are called “non-fungible tokens” or NFTs. A non-fungible token is used to identify something unique, which means that exchanging one NFT for another, does not necessarily equate to the same value (Entriken, Shirley, Evans and Sachs, 2018).

For instance, if Aisling owned Vincent van Gogh’s “Starry Night” painting and swapped it with Barry for Van Gogh’s “Sunflowers” painting both parties now have two entirely different items than they had prior, with their own individual characteristics and different economic but also emotional value.

All NFTs are distinct from one another and have a “tokenId” characteristic to enable this distinction. Each NFT’s tokenId is generated, and its details stored, at a specific smart
contract address, and as such, each smart contract address and tokenId pair must be globally unique (Smith et al., 2022).

Figure 2.8: ERC-721 Token Standard Functions and Events (ERC-721 OpenZeppelin Docs, 2022)

**ERC-1155**

Another critical evolution in Ethereum token types came in the form of the ERC-1155 standard. This built on the work already done through ERC-20 and ERC-721 standards, to provide a multi-token standard for a smart contract that can create and host both fungible and non-fungible tokens in the same place (Ashimine et al., 2022).

ERC-1155 removes the need for fungible tokens and NFTs to have two separate contracts, which both facilitates the development and deployment of a wide variety of tokens, as well as...
reducing the amount of redundant bytecode that implementing two separate contracts for each type leaves on the Ethereum chain (Radomski et al., 2018).

ERC-1155 standard tokens combine the methods in Figure 2.7 and 2.8 into one contract. It also alters the `balanceOf` function to take an address parameter, and a token Id parameter, returning the balance of tokens with that token Id at an address (Figure 2.9).

```
> gameItems.balanceOf(deployerAddress,3)
100000000
```

*Figure 2.9: ERC-1155 balanceOf Function Example (ERC-1155 OpenZeppelin Docs, 2022)*

### 2.2.4 Transactions and Messages

Considering the concept of a blockchain acting as a distributed ledger, a transaction event on a blockchain occurs anytime one account on that ledger transfers money to a different account, for example the transfer of 1 ETH from Aisling to Bob. This transaction changes the state of the ledger and hence the state of the blockchain.

Even on more functionally diverse blockchains like Ethereum, a transaction fundamentally represents this same concept of a change to the state of the chain, which requires that “miner” nodes execute a transaction and propagate the resultant state change to the rest of the network. Transactions on Ethereum can only be initiated by an externally owned account (EOA), that is an account owned and operated by a human being, rather than a smart contract (Wackerow et al., 2022). Transactions must be “signed” using the sender’s address and can
represent either a new contract being deployed to the network, or what’s referred to as a “message” being passed.

Messages are simply the data and the amount of Ether that is passed between two Ethereum addresses (Messages and Transactions on Ethereum, 2020), this can be between two EOAs or two smart contracts, or a mix of the two account types. In fact, all transactions on Ethereum function by creating and sending messages, with the sole exception of transactions that create smart contracts.

Figure 2.10: The four cases of message sending illustrated [CA meaning Contract Address] (Takenobu, 2022).

Messages are the key functionality that allow smart contracts to interact with one another, but only signed transactions have the power to change the state of the chain. This ensures that a single unit of on-chain activity is entirely recorded within a single transaction. Even if a transaction sends a message that invokes a contract which in turn invokes another smart
contract, because the inter-contract communication is just a series of message calls, the only update to the chain will come from the initial transaction call (Wood and Savers, 2022).

Updating the state of the Ethereum blockchain requires some computational energy to execute and therefore a fee is required to cover this cost (Richards et al., 2022). The unit of measurement of the computational effort needed to carry out operations on the Ethereum network is called “Gas”. Gas is essentially the fee required to conduct a successful transaction on the Ethereum blockchain (Antonopoulos and Wood, 2019).

Gas fees are paid for in ETH, specifically a denomination of ETH called “gwei”, where one gwei is equal to $10^{-9}$ ETH (1 gwei = 0.000000001 ETH). The gas fee for a transaction has become significantly more predictable than it once was, following the London Upgrade of the Ethereum network in August 2021 (Sigalos, 2021). Currently, gas fees are calculated using the following formula (Richards et al., 2022):

\[
gas \text{ units (limit)} \times (\text{base fee} + \text{tip})
\]

*Figure 2.11: How a message call updates the state of the chain (Takenobu, 2022)*
A user sets the maximum amount of gas units they are willing to use up to execute a transaction and if the execution is going to expend more than this limit the transaction is aborted (Antonopoulos and Wood, 2019). A base fee per unit gas is determined after each block is created, and users can arbitrarily set a tip whereby a higher tip means a greater reward for the node executing the transaction and therefore the higher the tip set, the greater priority is given to the execution of a particular transaction (Richards et al., 2022).

At the time of writing, the Ethereum network can only handle around 30 transactions per second (Kelly, Millman and Graves, 2022). Visa, in comparison handles around 1,700 transactions per second everyday (L, 2019). Ethereum does aim to improve its transactions per second capability as it transforms from “Ethereum 1.0” to “Ethereum 2.0” (Kelly, Millman and Graves, 2022).

2.2.5 Ethereum Evolving

The Ethereum blockchain is growing more and more popular, but as it evolves its old systems and methods grow outdated and are unable to adequately meet the demands and the vision of its creators and users.

For this reason Ethereum 2.0 or Eth2 has been proposed as an evolution of Ethereum that aims to radically improve the way the network operates, futureproofing it to become the backbone of Web 3. The upgrade to Ethereum was first proposed in January 2020 and is being rolled out in phases, the second phase of which, named “the merge” is expected to be deployed in Q2 of 2022, or certainly before the end of the year. The final major phase focuses on the implementation of shard chains, which to be brief splits the Ethereum network up from
a singular blockchain to function across 64 different blockchains (Kenny, Millman and Graves, 2022). It is expected that the Eth2 upgrade will be complete at some point in 2023 (Shard chains | ethereum.org, 2022).

The Eth2 upgrade will significantly impact two aspects of Ethereum in particular, its consensus architecture, and its scalability features. Eth2 will see Ethereum move from the extremely energy intensive proof-of-work consensus to a faster and more environmentally friendly proof-of-stake consensus architecture which may consume as little as 1% of the energy Ethereum uses today (Canales, 2022).

From a scalability perspective, using shard chains should mark a drastic increase in potential transaction throughput for Eth2 compared to Ethereum 1.0, by capably hosting over 100,000 transactions per second (Thomson, 2020). An increase in transactions per second will reduce the likelihood of enormous network congestion, and the hike in gas prices that comes with that, such as was seen with the CryptoKitties NFT game launch in 2017 (Hertig, 2017).

2.2.6 Decentralised Applications

Decentralised applications or “dApps” are applications that are built using a smart contract on a decentralised network and combine that with a frontend user-interface. DApps are an integral part of the Ethereum ecosystem and provide developers with a huge range of benefits. DApps inherently take advantage of the zero-downtime, data reliability and trustless computation behaviour that blockchains such as Ethereum provide (Dan et al., 2022).
Some examples of popular dApps are OpenSea, UniSwap, Axie Infinity and Pancake Swap. OpenSea is the largest NFT marketplace facilitating peer-to-peer transactions of Ethereum based NFTs. The dApp is the largest to run on the Ethereum blockchain by gas used (Wu, 2021). It saw its highest thirty-day trading volume of close to $5 billion in January 2022 (Sen Gupta, 2022).

![Figure 2.12: The OpenSea dApp combines a frontend user-interface with smart contracts on Ethereum](image)

Axie Infinity is a Pokémon style video game run partially on the Ethereum blockchain. When two players’ items battle one another, the rewards given to the winner have real-life value and can be traded for other fiat and crypto currencies at an exchange (Hayward and Graves, 2022).

UniSwap is the top decentralised exchange on the Ethereum network, allowing users to trade different cryptocurrencies. Since its inception it has seen close to $700 billion in trading volume (Brabus, 2021). PancakeSwap is also a decentralised exchange like UniSwap, it is run
PancakeSwap attracts over 11 times as many monthly users as UniSwap almost entirely due to the substantially lower fees it can offer to its users when trading cryptocurrency versus UniSwap’s high fees (Brabus, 2021). UniSwap’s expensive fees, like other Ethereum based dApps’, are driven by the high network congestion of Ethereum and therefore the cost of transaction is very high too.

Ethereum’s network congestion and high fees have led to some of the key development decisions taken within the later chapters of this paper.

2.3 Non-fungible Tokens

2.3.1 Outline

Blockchain technology has led to the rise of non-fungible tokens, or NFTs. NFTs can be used to represent items that are innately unique in the digital world (Entriken, Shirley, Evans and Sachs, 2018).

NFTs are created through a process called “minting”, which is the act of publishing a unique instance of a non-fungible token on the blockchain. This token can be linked to some metadata, that is to say, external data such as an image, piece of music, or any other digital data that the user wishes to associate this specific token with (Mudgil, 2021).

Originally implemented on the Ethereum network, NFT functionality has subsequently been built within numerous different blockchains, each offering their own benefits and NFT
standards that facilitate different functionality aiming to address specific use cases. NFTs can be bought and sold on different exchanges and enable their original creators to continue to generate revenue from them in perpetuity as they are traded between holders.

NFTs have several different use cases, ranging from uses in gaming and the metaverse through to being utilised to manage events. The most common use case for NFT technology at the time of writing, however, is artwork, with the most expensive NFT ever sold being a piece of art called “Everydays: The First 5000 Days”, the auction for which had a winning bid of $69.3 million (Graves, Phillips, and Hayward, 2022).

NFTs are still a new technology and thus they are imperfect and are constantly being developed, so there are naturally some issues with their use in certain scenarios.

2.3.2 Key NFT Types

As discussed in section 2.2.3 on Ethereum token types, there are two commonly used standards for smart contracts that instantiate NFTs, ERC-721 and ERC-1155.

To briefly recap, ERC-721 is a non-fungible token (NFT) standard that enables Ethereum tokens to represent unique items. ERC-1155 is a standard that allows for both fungible and non-fungible tokens to be created and managed from a single smart contract. There are, however, many other standards and variations for NFTs aside from the two mentioned, that have some unique characteristics and functionality. Furthermore, not all NFTs are created and operated on the Ethereum network, so to be comprehensive this section will also discuss an example of an innovative non-Ethereum based NFT standard.
ERC-998

ERC-998 is another Ethereum based NFT standard that is an extension of the ERC-721 standard. This standard enables ERC-721 tokens to own ERC-20 and even other ERC-721 tokens (Lockyer, Mudge and Schalm, 2018).

The jargon can become overwhelming, so to explain, ERC-998 is an NFT standard that enables Ethereum based NFTs to own other NFTs or Ethereum-based fungible tokens. This adds an extra level of complexity to NFTs as digital assets, enabling an NFT to be composed of other “crypto assets”. An ERC-998 token can represent an assortment of Ethereum digital assets, be they fungible or non-fungible tokens and this assortment can be traded in one single transaction (Jodet, 2018). Trading an ERC-998 token can be thought of like trading a portfolio of artwork all at once, rather than each painting within the portfolio individually.

![ERC-998 Token Illustration](image)

*Figure 2.13: ERC-998 Token Illustration (Jodet, 2018)*

RMRK 1.0, 2.0 and 3.0

RMRK NFTs are not based on any Ethereum standard and are not hosted on the Ethereum blockchain. They are hosted on a blockchain called Kusama, which is a sister blockchain to the Polkadot blockchain that is discussed in chapter 2.4.
RMRK NFT standards are comprised of five core building blocks, referred to as NFT “Legos” (Kim, 2021). The first of these building blocks is “on-chain emotes” which has been available since the RMRK 1.0 standard. On-chain emotes allow users to react to an NFT, similar to what one would expect when opening an emoji keyboard. RMRK 2.0 paved the way for Nested, Multi-Resource and Conditional-rendering NFT Legos (RMRK Docs, 2022).

Nested NFTs function like ERC-998 NFTs by allowing different assets to own and equip other NFTs. Multi-resource NFTs means that an NFT can link to more than one datatype, such as an audio book having a pdf resource, an audio file resource, and a cover JPEG resource, all of which auto equip depending on whether someone is reading the book on a Kindle, listening to the book through Audible or just browsing it in the store (RMRK Docs, 2022). Conditionally-rendered NFTs mean that once a condition is met, the state of the NFT can change, for example, a cartoon of a football player may change from having a generic pose to a happy pose if they scored a goal in their most recent match.

RMRK 3.0, due to be released in 2022, will see NFTs be able to function as decentralised autonomous organisations (DAOs). A DAO is a decentralised organisation that is governed democratically by anyone who has a vote, often represented as a token, in the organisation (Hackl, 2021). RMRK 3.0 will allow NFTs to create their own unique tokens which can be shared amongst stakeholders. If this NFT were, for example, a digital billboard in a metaverse, RMRK 3.0 token holders could then vote to allow this billboard to display a Coca-Cola advertisement (RMRK, 2021).
2.3.3 Exchanges

NFTs can be traded on NFT exchanges. There is a myriad of exchanges across various blockchain networks that all have their individual merits, niches, and requirements.

OpenSea

OpenSea, first mentioned in chapter 2.2.6, is a marketplace for Ethereum standard NFTs and is the largest NFT exchange in operation at the time of writing (OpenSea, 2022). It hosts all kinds of art, videos, collectibles and more, and it accepts over 150 token types as payment. OpenSea is home to some of the most popular NFT art collections such as the Bored-Ape-Yacht-Club. As an Ethereum based dApp it does subject its users to high gas fees.

Singular

RMRK NFTs are primarily traded on an exchange called Singular. Singular, like the RMRK NFT standard, is not based on Ethereum but rather on Kusama. Singular is the native exchange for RMRK tokens, as it is developed by the same team that develop the RMRK NFT standards (Singular, 2022). For that reason, it is one of the most popular NFT exchanges in the DotSama ecosystem. DotSama is the name referring to the Polkadot and Kusama sibling blockchains (Clarke, 2021).

Moonsama

Moonsama is an NFT exchange based on a blockchain network called Moonriver, which is a sister network to Moonbeam (chapter 2.5). Moonriver is referred to as a parachain of the Kusama network, parachains are discussed in more detail in chapter 2.4.3, but for now think of Moonriver as its own blockchain that also links to the Kusama blockchain. Moonriver hosts Ethereum-like smart contracts and is also on the Kusama network (Singh, M., 2021).
which means that Moonsama can host RMRK, ERC-721 and ERC-1155 standard NFTs. The type of NFTs on offer range from plots of land in a Minecraft server, to classic digital artwork, to unique items usable on the exchange’s native videogame, Exosama (Moonsama, 2022).

### 2.3.4 Royalties

NFTs have a unique feature that allows the original creator of the NFT to receive earnings in perpetuity as the item is traded. These creator fees, or royalties, can be set arbitrarily by the creator of the NFT as they are minting it. Different exchanges allow for different maximum royalty fees (Beck, 2021).

OpenSea, for instance, allows creators to set royalty fees up to 10% of the total sale price, meaning as their artwork, video, music, in-game item or whatever the NFT happens to represent is sold and resold, they will continue to receive 10% of the selling price (OpenSea Support, 2022).

Setting royalties is more of a balancing act than it may originally appear, however. While NFTs primary function is to allow people to own a unique item, many peoples’ primary motivation when purchasing an NFT is for it to act as a financial investment with the aim of generating a profit over a given time-period (Beck, 2021). Setting high royalty fees reduces the profit margin an investor will make on a sale, which can reduce the attractiveness of an NFT sale to begin with.
Using OpenSea as the example once more, if a user bought an NFT that had a royalty fee set to 10% with the aim of generating profit from the purchase, they would have to wait for the value of the NFT to increase by 14.3%, as OpenSea, like most NFT exchanges takes its own fee for hosting the sale of the NFT (2.5%).

\[
\text{Proportion of Sale Fee Received After Fees} = 1 - (a + b) = 87.5\%
\]

Assuming the item was purchase for a price of $1, then:

\[
\text{Sale Price Required to Break Even} = \frac{1}{c} = \$1.1429
\]

This represents a sale price increase of roughly 14.3%.

Figure 2.14: NFT Value Increase Required to Breakeven with 10% Royalty Fee on OpenSea

ERC-721 and ERC-1155 standard tokens do not natively support royalty fees, so an Ethereum Improvement Proposal (EIP) was created aiming to create a standardised extension onto these NFT standards that can allow for royalties to be added (Burks, Morgan, Malone, and Seibel, 2022).

### 2.3.5 Uses of NFTs

The list of potential use cases for NFT technology is growing day-by-day but already touches many established industries around the world. To avoid meandering through hypotheticals, this report will look at the most established current use cases of NFTs.
Artwork and Music

This remains the predominant use of NFT technology. The concept is quite simple, since every piece of art and music is innately unique, NFTs are the perfect way to represent this online (Fowler and Pirker, 2021). In addition, the aforementioned royalty features enable artists to continually earn revenue as their art is traded. For music in particular, this presents a very interesting use case where an artist could allow their music to be used in other media at either a set price per time used (Verma, 2022) or if used in a directly revenue generating piece of content, the music could be used for a percent of revenue per time used.

Gaming

NFTs are already seeing some use in gaming. Games like the partly Ethereum-based Axie Infinity or the DotSama based Exosama utilise NFTs to allow players to buy and trade unique in-game items that they or their character can equip and that will give them certain perks or idiosyncratic functionality (Hayward and Graves, 2022).

Metaverse

What exactly a metaverse is, is dealt with in chapter 2.6. For now, we will consider it as a digital world in which people and societies live, socialise, and communicate as they do in the real world. NFTs are seen as being a pivotal part of such a world, as they allow individuals to own unique items promoting a person’s ability to express themselves in a digital realm (Wilser, 2021).

Event Management and Ticketing

NFTs open-up a whole new range of possibilities for event managers too. The most recent SXSW festival was brought to life using NFT technology, from dedicated NFT galleries, to
using NFTs to take advantage of unique experiences, such as that offered to holders of NFTs in the “Doodle” collection, who could swipe their phone and their NFT would appear on the walls of the venue or even in the foam of their coffee (Thomas, 2022). Tickets to events could also be supplied as NFTs, which could unlock special features for attendees as was seen at SXSW or could simply act as a life-time memento of the event, like an old ticket stub from a Beatles concert (Wilser, 2021).

**NFTs as a Financial Asset**

The world of decentralised finance, which to simplify, is blockchain based financial systems, allows for NFTs to function as a financial asset. NFTs can be put up as collateral for loans, can be used effectively like a security when trading or can simply be bought and sold as a retail investor as the asset’s value goes up or down (Yousaf and Yarovaya, 2022).

**2.3.6 Issues with NFTs**

The NFT industry is still very much in its development phase, so there is constant innovation and differentiation occurring (Whittington et al., 2020). This does mean NFTs are still imperfect, and improvements are being made constantly. Chapters 3 through 5 of this paper focus on the use of NFTs in the gaming and metaverse context, and in that field there are four key issues currently.

There are two challenges with usability, specifically the rate at which NFTs are issued and the cost of minting and trading them due to gas fees on Ethereum. Ethereum’s low transaction per second volume means using NFTs at scale, at least on Ethereum is somewhat limited as they take significant time to mint (Wang, Li, Wang, and Chen, 2021).
The other two major hurdles to the use of NFTs in a videogame metaverse are NFT interoperability or the lack thereof, and the issue regarding updating NFTs (Wang, Li, Wang, and Chen, 2021). NFTs are currently limited to function and communicate with other NFTs on their same network. Which means if Aisling owns an NFT on Ethereum, and Barry owns an NFT on another popular blockchain like Solana, these two NFTs cannot interoperate. Furthermore, if Aisling uses her NFT in a videogame, and the state of the NFT changes, for example her NFT represents a car that can be used in a videogame, and she has given that car a new engine in the game enabling it to go faster, updating that NFTs state on the blockchain is very difficult because NFTs are immutable by default.

This paper will aim to address these issues in Chapters 3 through 5.

2.4 Polkadot

2.4.1 Outline

Polkadot was proposed by the ex CTO of Ethereum Dr. Gavin Wood in 2016 while he was working on solutions for “sharding” the Ethereum network. His vision surrounds a “consensual internet”, a decentralised internet where companies like Facebook and Google do not have ultimate control over a user’s private data. It was from this vision that he coined the term “Web 3.0” (Polkadot Network | Gavin Wood, 2022).

Polkadot is a blockchain that is built to be limitlessly scalable into the future. It achieves this by enabling other blockchains built for specific purposes to work together securely and
efficiently at scale, which is why Polkadot is referred to as a “heterogenous multi-chain framework” (Wood, 2016).

At a macro-conceptual level, the Polkadot network, or more accurately the Polkadot network protocol (Polkadot) is a public blockchain network that can be thought of as a ‘blockchain of blockchains’. Blockchains connected to the core Polkadot chain, known as the “relay-chain”, are referred to as parachains (Polkadot: Are you ready to start building?, 2020). Each parachain, though connected to the Polkadot relay chain, is its own sovereign blockchain, optimised for its own specific purposes. Parachains are often referred to as shards of the overall Polkadot ecosystem, and they are equipped to be able to communicate rapidly and securely between each other using a cross-chain messaging protocol native to Polkadot.

“Blockchain bridges” have existed for some time as a means of allowing blockchains to connect, often through a third party and they generally act as a means of transferring cryptocurrency from one chain to another. These bridges have significant issues such as slow processing speeds and poor security due to centralisation (Whiteboard Crypto, 2021). Polkadot’s relay-chain ensures its parachains do not encounter any of these issues when communicating between each other. Parachains can also act as entirely trustless and decentralised bridges connecting the Polkadot relay-chain to non-relay-chain based blockchains, like Bitcoin or Ethereum (Drwięga, 2021).

Polkadot natively implements a variation of proof-of-stake consensus, which means its energy consumption is significantly lower proportionately to Bitcoin and Ethereum. Using its native currency, the Dot, Polkadot supports decentralised governance of its platform by
allowing Dot holders to be able to vote on referenda or to use their Dot to vote for certain parachains to be included as parachains on the network.

It is worth briefly mentioning Polkadot’s sister network Kusama. Kusama is a “canary” network for Polkadot, it is different from a “testnet” on Ethereum as all actions carried out on Kusama have redeemable value. Kusama and Polkadot share a great deal of their core code base, but the key difference is that Kusama was created to facilitate rapid testing and development of new concepts, so everything tends to take place in shorter, less stable time periods on Kusama than on Polkadot (What is the difference between Polkadot and Kusama? · Polkadot Wiki, 2022).

2.4.2 A Heterogenous Multi-Chain Framework

Polkadot describes itself as a scalable, heterogenous multi-chain framework (Wood, 2016). It enables a set of completely unique and independent blockchains, an arbitrarily constructed example of which could be a set of chains including Bitcoin, Ethereum and the Binance Smart Chain, to co-exist within one ecosystem, hence the term *heterogenous*.

Polkadot provides no additional functionality over what these set of chains offer as they co-exist independently and unbeknownst to each other, except for two critical features. By existing within the Polkadot network, this set of chains can take advantage of Polkadot’s pooled security, and its trust free interchain transactability. These are the features that make Polkadot scalable (Wood, 2016).
The pooled security feature enables all chains connected to Polkadot to process their blocks in parallel. This is explained in more detail in chapter 2.4.4, but in short this means that computational operations on Polkadot can run rapidly on chains optimised to handle very specific actions. This is seen as a solution to the transaction bottlenecks (Polkadot: Are you ready to start building?, 2020) issue that occurs on blockchains that aim to support a “one-size fits all” blockchain network, like Ethereum.

Polkadot also enables the blockchains connected to it to communicate seamlessly and securely between each other. This overcomes the issue of a blockchain being a “walled-garden”. Most public blockchains are inherently sealed off from other blockchains and the result is that there is limited ability for these different networks to quickly and securely share data amongst each other (Wood, 2020). Chapter 2.4.3 details exactly how this interchain transactability, also called cross-chain messaging, is enabled.
Interchain transactability brings with it significant opportunities. The most prominent of which is that it enables dApp developers to theoretically build a dApp that requires some specific functionalities of a blockchain, but all of these key functionalities are carried out on unique blockchains optimised to do their one specific task. Each chain can carry its task out in parallel to the others and share key information between them as the operation is executed (What is Polkadot?, 2020).

Developers are empowered by interchain transactability to be able to build much more complex dApps, that run more efficiently and have a greater scope of utility for their users. Furthermore, Polkadot’s application level transactive system, means that applications on Polkadot decide what code gets executed and when (Bloomberg, 2021), rather than miners and/or smart contracts on Ethereum, which greatly reduces the economic barrier to use that Ethereum has (Allison, 2021) to a functional metaverse.

### 2.4.3 The Relay Chain, Parachains & XCMP

When discussing the Polkadot blockchain, the concept being referenced is really Polkadot’s central chain, that all the other chains wishing to join the ecosystem must connect to. This central chain is called the Polkadot “relay-chain” and is the cornerstone of the entire ecosystem (Annison, 2021).

The relay-chain is the piece of technology that enables Polkadot’s pooled security and seamless cross-chain communication to occur (Polkadot: Behind the Code, 2020). If the relay-chain was to exist in isolation without any other chains connected to it, however, it would be almost useless as the relay-chain itself does not support smart contract
functionality. Instead, the relay-chain is highly optimised to connect to other blockchains and support those chains in executing state changes. Polkadot’s relay-chain enables the blockchains connected to it to execute these state-changes in parallel (Bertocchi, 2021), consequently, blockchains that connect to the relay-chain are called “parachains”.

A major benefit of Polkadot’s parachain model is that the network is optimised to handle a much larger transaction per second volume than Bitcoin or Ethereum is able to (Sinha, 2021). It is in fact expected that with a future introduction of “nested relay-chain” systems on Polkadot, which is the concept of other relay-chains being connected as parachains to the current singular relay-chain (Polkadot Lightpaper, 2022), that the Polkadot network could quite quickly scale to manage over 1 million transactions per second throughput (Sinha, 2021).

As on Bitcoin and Ethereum, a state-change on the Polkadot network is referred to as a transaction and like on Ethereum, it is necessary for Polkadot to charge some fee for
executing a transaction in large part due to the risk of denial-of-service attacks without an economic consequence for clogging up the network (Transaction Fees · Polkadot Wiki, 2022). Polkadot does not use a gas-fee system like Ethereum, instead Polkadot charges for use of its network using a weight-fee model.

The weight-fee transaction model is comprised of a “weight fee”, a “length fee” and a “tip”. A transaction’s weight fee consists of a base weight, which covers the computational complexity (and therefore energy overheads required) of the transaction as well as the time it takes to execute, a weight fee also contains a call weight which covers the cost of any system calls required by the transaction. The length fee covers the size of the transaction in bytes, and the tip fee functions as it does on Ethereum by incentivising nodes to include a given transaction in the latest block (Transaction Fees · Polkadot Wiki, 2022).

By handling transaction fees at the relay-chain level, the weight-fee model removes the explicit need for parachains themselves to implement a “gas fee” system for handling transactions (Smart Contracts · Polkadot Wiki, 2022). In the event that a parachain does require or does choose to implement a “gas fee” system to handle its own network’s functions, since the Polkadot network’s “validator nodes” (chapter 2.4.4) do not compete with each other on price, transaction fees can remain substantially lower than those seen on the Ethereum blockchain (Rübe, 2021).

Polkadot enables its parachains to communicate between each other using a cross chain messaging format called XCM. XCM itself cannot send messages, but it provides a standard for how message transfers should be performed (Cross-Consensus Message Format (XCM) · Polkadot Wiki, 2022), to ensure that messages are guaranteed to be delivered in a trustless
manner and in a well-defined order (XCMP overview, 2022). XCMP (as opposed to XCM) is the protocol that enables parachains on the same relay-chain to communicate. Although XCMP is such a cutting-edge concept that it has not yet been fully developed, the conceptual architecture has some fundamental features that are unlikely to change (Cross-Consensus Message Format (XCM) · Polkadot Wiki, 2022).

XCMP ensures that cross-chain messages will not be passed to the relay-chain, but directly to the intended parachain and that a constraint will be placed on the byte size of the message. Through XCMP, parachains retain the ability to block incoming messages from other parachains. Furthermore, parachains’ native block-creating nodes called “collators” (explained in chapter 2.4.4) create what is effectively a queue of incoming and outgoing XCM messages safeguarding order (Cross-Consensus Message Format (XCM) · Polkadot Wiki, 2022).

*Figure 2.17: Illustration of the XCM tech stack (Cross-Consensus Message Format (XCM) · Polkadot Wiki, 2022)*
2.4.4 Polkadot Proof-of-Stake

The Polkadot network’s unique relay-chain based architecture implements a “nominated proof-of-stake” (NPoS) consensus mechanism (Jackson, 2021), which is an adapted version of the more general proof of stake consensus architecture described in chapter 2.1.3.

Key actors in the NPoS architecture go slightly beyond generic proof-of-stake (PoS) consensus systems. Validators remain the most fundamental actor involved as is the case in other PoS implementations. The role of the validator on the Polkadot network is two-fold. A validator node is responsible for production of new blocks on the relay-chain and is also responsible for validating new blocks produced by parachains. Validators do not compile the data for a new block on a parachain network, their role is simply to verify the legitimacy of the new parachain block and propagate the new parachain block out among the other validators, ensuring finality and consensus across the network. Validators are given a block reward for acting honestly (Wood, 2016).

Another key NPoS actor on the Polkadot network are “nominators”, which perform the same function as described in chapter 2.1.3. Nominators can support up to 16 validators with a financial stake which makes a validator more likely to be selected to forge a new block in return for a percent of the block reward given to the validator (Staking Facilites, 2020). Should a validator be found to act dishonestly, both its and its nominators will lose some or all of their stake.
Transaction “collators” are a type of actor specific to Polkadot’s NPoS implementation. Collators are full-nodes that run on parachain networks, and whose role it is to gather-up and propose the latest parachain-specific transaction data to be included in the parachain’s next block. Collators then pass this proposed block on to validators to ratify its state. The incentivisation of collators can vary as they are parachain specific actors, however, in general it is expected they vie to compete for transaction fee collection (Wood, 2016).

NPoS works much like an election. Implementing a method called the Phragmén method, which was developed in the late 19th century to optimise an approach for the selection of a small set of people from a larger set of potential candidates (Brill, Freeman, Janson, and Lackner, 2017), NPoS seeks to ensure that slots are assigned to validators in proportion to their nominations which maintains proportional justified representation.

Through the Phragmén method, the higher the stake behind a validator the more likely it is to be included in the active validator set. A set of validators will then operate for an “era” of time on the relay-chain, which at the time of writing is 24 hours, before a new set of validators is rotated into the active set (Moreau, 2020).

It is then necessary to avoid an underrepresentation of “minority” validators, that is validators which have lower stakes than their peers, as this poses a security risk if block minting is left to a small set of validators with large financial staking. This scenario would dramatically reduce decentralisation of the network and make it easier for validators to perform malicious actions in an undetected manor. For this reason, the active set of validators see their stake balanced using a method called maximin support (Cevallos and Stewart, 2021), which in-short temporarily raises the stakes of the least-backed nodes while they are in the active set.
Once a balanced and proportionally representative active set of validators is created, it is split into small sub-groups that serve specific parachains (Wood, 2016).

### 2.4.5 Decentralised Governance & Parachain Auctions

Polkadot’s native cryptocurrency is called the Dot. The Dot was designed to act not solely as a financial instrument, but as a tool used to indicate an individual’s stake in the Polkadot ecosystem. Dot is used to carry out a range of functions on the Polkadot network, all of which are aimed at ensuring true network decentralisation and security (Burdges et al., 2020).

The Polkadot protocol is described as a “metaprotocol” because fundamentally it is a very low layer of software functionality that sits beneath more advanced functionality that we would ordinarily call the “blockchain protocol” (Siller, 2021). Accounts, balances, transactions, parachains, governance, and much more fundamental aspects of a blockchain protocol are implemented at a higher level of the Polkadot framework than the underlying meta-protocol (Wood | A Walkthrough of Polkadot’s Governance, 2020).

This metaprotocol enables stakeholders of the network, those that own some Dot tokens, to vote on referenda that fundamentally decide how the Polkadot ecosystem will operate (Bassey, 2021). These votes can make quite dramatic changes to the runtime methods that allow Polkadot to function, be it changing the structure of accounts, updating the way in which transactions are recorded or voting on how parachains should function (Wood | A Walkthrough of Polkadot’s Governance, 2020). Dot tokens are also used to vote on non-technical operations such as network marketing proposals.
In this way the Dot token allocates decision making power to Polkadot’s stakeholders, decentralising the governance of the ecosystem.

A further example of this is the requirement for blockchains that wish to become parachains on Polkadot’s relay-chain to win an auction to earn their slot. Prospective parachains must compete with one another to amass support in the form of “locked” Dot-tokens before the end of an auction. Once an auction period is finished, the parachain with the most Dot locked is given the ability to connect to the relay-chain.

When a Dot holder votes for a particular parachain to earn its place on the Polkadot network their Dot is “locked”, meaning it becomes illiquid, for a length of time dependent on whether the parachain wins the auction or not. If a parachain does not win an auction, the Dot used to vote for it is returned to the owner at the end of the auction period, if the parachain is successful, however, the Dot will be locked for the full duration of that parachain’s lease period which is 96 weeks (Binance, 2021).

**2.4.6 Forkless On-chain Upgrades**

The nature of Polkadot’s metaprotocol enables seamless upgrades of the Polkadot network and its parachains without the need to fork the blockchain. To implement some upgrades, other blockchains, like Ethereum, require a hard-fork update (Wo, 2021).

A hard-fork is a radical change to the protocol of a blockchain that results in two branches, an old branch on the non-forked blockchain, and a new branch on the forked-blockchain. When a fork occurs, the older version of the blockchain is still generally operational and accessible,
so the only way to get users to use the updated fork of the network instead is through word-of-mouth and/or other social conventions (Frankenfield and Rasure, 2021).

The problem with forking blockchains is that it fundamentally splits a community, meaning an update to a blockchain network is an incredibly arduous undertaking and therefore most blockchains avoid updating as much as is possible, slowing innovation (Polkadot Network, 2021).

Polkadot’s metaprotocol and its decentralised governance systems enable simplified updates for the logic of Polkadot’s own relay-chain and for all of its parachains. Updating a Polkadot based blockchain to fix bugs, add new features or implement more advanced technologies is all done using “forkless upgrades” (Wo, 2021). This process is simplified further if the parachain is built using Polkadot’s Substrate framework, which can be thought of as being akin to a software-development-kit (SDK) for blockchains.

Forkless upgrades on Polkadot make it is easier to maintain blockchains as state-of-the-art networks and removes the laborious efforts required to manage a community through the upgrade process. This ensures that Polkadot based chains can easily remain at the cutting edge of technological innovation into the future (Polkadot Network, 2021).
2.5 Moonbeam

2.5.1 Outline

Moonbeam was the first parachain to fully deploy onto the Polkadot network, on the 11\textsuperscript{th} of January 2022 (Moonbeam Completes Launch Process, 2022).

Moonbeam is an EVM compatible blockchain, which means that it natively supports smart contract functionality, adding a new layer of complexity and development opportunities to the Polkadot ecosystem. Moonbeam can be used to instantiate ERC-20, ERC-721 and ERC-1155 token standards.

The parachain also has its own token, the GLMR. Much like ETH on Ethereum, GLMR has sub-denominations called “Wei” and “Gwei”. Wei is the smallest unit of GLMR, while Gwei stands for Gigawei and is the standard unit used to calculate gas fees on the network (Moonbeam Network Overview, 2021).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{moonbeam-denominations.png}
\caption{GLMR Denominations (Moonbeam Network Overview, 2021)}
\end{figure}
Moonbeam makes use of cross-chain interoperability on Polkadot while also supporting the programmable features of the Ethereum network, which allows for a huge variety of complex dApps to be built using the blockchain.

### 2.5.2 Ethereum-on-ramp

Moonbeam acts as an on-ramp for existing Ethereum based smart-contracts and dApps to connect to the Polkadot ecosystem (Moonbeam, 2021). Moonbeam supports solidity-based smart contracts, Ethereum token standards and Ethereum ecosystem and development tools, such as wallets (Ethereum Integration, 2021).

Developers face very little changes if migrating some projects from Ethereum to Moonbeam, and in fact many major Ethereum projects have already established connections to the Moonbeam network. SushiSwap is an example of a very popular decentralised exchange that has deployed on Moonbeam in March 2022.

Moonbeam does have some differences that developers need to be aware of especially if migrating projects over from existing EVM compatible clients. One of which is that Moonbeam makes use of the on-chain governance provided through Polkadot’s Substrate framework which enables the seamless upgrading of the blockchain itself. Another significant difference is Moonbeam achieves consensus on new blocks using Polkadot’s NPoS consensus architecture and at the time of writing Ethereum still primarily uses proof-of-work consensus. This means that Ethereum based contracts that rely on some internal aspects of the proof-of-work consensus will not function correctly on Moonbeam (Ethereum Integration, 2021).
There is a project being built externally to Moonbeam that will connect an Ethereum bridge directly with the Polkadot relay-chain. Development timelines for this project are somewhat opaque, however, so in the meantime Moonbeam is working on building its own “integrated point-to-point” Ethereum bridge that can act as a direct link between itself and Ethereum (Cross Chain Integration Plans, 2021).

### 2.5.3 Transactions

A key feature of Moonbeam is its facilitation of the use of smart-contract integrations with other parachains on Polkadot (Cross Chain Integration Plans, 2021). Using XCMP, Moonbeam will be able to power cross chain communication with other parachains such as the decentralised finance platform Acala.

Moonbeam aims to continue to evolve its network to allow its smart contracts to incorporate as full a range of features provided by other parachains on Polkadot as is possible. To that end, the Moonbeam team created the XC-20 token standard. This builds on Ethereum’s ERC-20 token standard that is used on its namesake and many other EVM compatible blockchains in existence (Moonbeam Team, 2022). The XC-20 standard on Moonbeam offers an array of benefits, one of which is the removal of the need for insecure liquidity bridges between blockchains, as an XC-20 token will be able to be transferred and utilised seamlessly by all parachains on Polkadot using the XCM format. XC-20 tokens should lay the foundations for seamlessly supporting other, more complex, asset transfers across the Polkadot infrastructure.
2.5.4 Opportunities for Moonbeam in the Metaverse

The Moonbeam network as a parachain on Polkadot presents a myriad of opportunities for developers due to its low cost of transaction, high transaction per second capability and its ability to harness the power of Ethereum, and Polkadot and its relay-chains.

The average dollar cost of executing a transaction on Moonbeam was 500 times lower than the cost of doing so on Ethereum on April 5th, 2022 (Figure 2.19).

![Figure 2.19: Ratio of Moonbeam Gas Fees (USD) to Ethereum Gas Fees (USD) on April 5th, 2022.]

This value is volatile and can fluctuate, but the overarching point is Moonbeam is a far cheaper network to execute transactions on, which is especially important when executing trades of NFTs on platforms like OpenSea, which cost over $38 on April 14th, 2022.
Figure 2.20: Average Gas Cost of an OpenSea Sale (Ethereum Gas Tracker, 2022).

Using the 1/500 ratio from Figure 2.19 above we can estimate the cost of the same transaction on Moonbeam to be less than $0.08. This price difference is critical if blockchain technology is going to become the backbone of a metaverse. If someone was to purchase an NFT representing a piece of clothing to use on their metaverse avatar for a price of, for example, $50, paying an additional $38 on top of that almost doubles the overall cost to the user. Whereas an additional $0.08 is relatively negligible even for a purchase of an item only worth $1.

At the time of writing, Moonbeam has the ability to handle more than ten thousand transactions per second (Hamilton, 2022), which is expected to grow to hundreds of thousands of transactions per second as Moonbeam itself develops alongside Polkadot. This is a further advantage that Moonbeam has as a platform for a metaverse, as it will be able to handle significant transaction throughput immediately, and can make use of Polkadot’s forkless upgradability features to continually develop and optimise its software as its use cases become clearer over time.
On top of all these features, being a parachain on Polkadot means that users who held metaverse items on Moonbeam, could realise their value very easily for other types of value that Polkadot’s parachains promise to deliver, such as credits on Pinknode, a proposed node-as-a-service platform for Polkadot developers (About Pinknode, 2021), or intellectual property hosted on InvArch (InvArch, 2021). These are two examples of the simple metaverse-based trading that could be possible using Moonbeam as a backbone.

2.6 The Metaverse

2.6.1 Outline

On the 28th of October 2021 Mark Zuckerberg announced the rebrand of the technology giant Facebook to Meta. Zuckerberg’s decision was based on his belief that “the metaverse will be the successor to the mobile internet” (Milmo, 2021).

Figure 2.21: Metaverse Depiction in Ready Player One Film (Spielberg, 2018)
The etymology of the term “metaverse” predates Zuckerberg’s announcement by quite some time, first appearing in a dystopian sci-fi novel by Neal Stephenson, Snow Crash (1992). The film Ready Player One (Spielberg, 2018) then presented the most visually in-depth depiction of a metaverse in popular culture. Yet, since Zuckerberg’s announcement the term metaverse has become a major buzzword around Silicon Valley across numerous diverse technology streams, without anyone truly knowing what exactly the metaverse is, what it will become, or what technology will be integral to its existence.

Talking about the metaverse today is much like talking about the internet in the 1970s, we all know that it is coming, we just don’t know how it will work, what it will mean or what it will do (Ravenscraft, 2021). Most conceptions of the metaverse involve some form of virtual reality and a digital 3D universe in which people can socialise and express themselves, this is common in both Stephenson and Spielberg’s works, and it seems to be a broadly accepted feature of the forthcoming technology.

Beyond this, it is hard to have certainty when talking about metaverse-related technologies. Truthfully, it is unknown even whether the term metaverse should be referred to with the definite or indefinite articles, should we speak about “the metaverse” singular, or “a metaverse” of which numerous independent variations exist.

One thing that is likely is that the metaverse (or metaverses, plural), will require some form of economic transaction ability. This is where decentralised blockchain technology is powerful. Blockchain and NFT technology enables people to trade the digital ownership of any item, be it a digital currency, some form of clothing to use on their metaverse avatar, or a digital trophy, collectible, or representation of achievement to show off in their metaverse.
collection (Reddy Gadekallu et al., 2022). If the metaverse is to be singular and is to be global, blockchain technology can help to homogenise currency, so a singular currency can be used and understood by all people in the metaverse. Blockchains like Polkadot can support governance of the metaverse, empowering users to be able to vote democratically on what happens within it. Blockchains are also decentralised, transparent and immutable, helping to avoid a metaverse of centralised tyranny that is controlled by the few (Reddy Gadekallu et al., 2022), which ironically is the plot of Spielberg’s film.

2.6.2 An Interoperable Metaverse

A unified metaverse experience fundamentally relies on independent metaverse projects being able to interoperate, the same way that modern web-browsers can link to any website and any webpages can link to each other and potentially share data (Takyar, 2022).

Pioneers of the metaverse see it ultimately comprising an evolution of the internet, and for that reason openness and interoperability are going to be critical aspects. Many companies, such as Microsoft through its videogame Minecraft, and Roblox, have already developed virtual worlds of their own and are claiming to be pioneers of the metaverse (Gent, 2021).

While the verity of that statement is subjective, it is certain that there is little to no interoperability between these 3D worlds today. There are a number of technical hurdles to overcome before this is possible, many of which do not have known solutions even at a theoretical level. Issues such as standardising the representation of 3D files and materials, creating a standard for physics such that characters move and vehicles drive in a normalised
way, and some way of assimilating simulation methods are all significant technological obstacles to overcome (Gent, 2021).

2.6.3 The Gaming Metaverse

As mentioned, Minecraft, Roblox and other videogames like Fortnite all state that they are pioneering a metaverse. Ways in which they are doing this include supporting a high degree of customisation of players’ in-game characters by allowing players to purchase different skins to use on their avatars (Colagrossi and Boyle, 2021).

Roblox is going even further than just enabling personalisation, hosting live events on its 3D platform. In November 2020, the online videogame held a live Lil Nas X concert over two days that attracted over 30 million viewers, and at which the popstar and all attendees were present as their in-game avatars. Roblox also equips everyone with the technology to build 3D online multiplayer friendly games on its platform. Although many of them might be quite simplistic at this moment, the possibility of the function itself serves to illustrate the advantages that videogame companies seem to have in a metaverse environment (Kharif and Gillette, 2021).

The gaming metaverse, however, like the metaverse as a whole, will benefit from presenting a unified experience where items earned in one game can be used in other games or at the very least exchanged with items from other games, which is not currently possible.
2.6.4 Item Ownership and Value Attribution

Items in the metaverse must have some value attributed to them that is bi-directionally exchangeable with value in the real world.

Currently, when a gamer plays a video game, they invest their energy, time and focus into unlocking and developing new skills and custom items. However, when that game becomes uninteresting to the player, that time and energy investment loses all its value, as there is no means to swap that value for anything else outside that game, either within the metaverse or the real-world.

Attributing value to in-game items will empower metaverse users to have ultimate ownership of the items they use (Reddy Gadekallu et al., 2022), and will create a full and liquid metaverse economy. Currently, most popular videogames allow players to spend money to buy in-game items, but few enable players to resell these items again or exchange them. Outside of blockchain based games, almost no popular videogames enable users to trade their in-game items back into real-world cash or other non-game-based value.

For videogame companies, allowing items to be earned, bought, exchanged with items in other games, and sold on for other real-world value would create lifetime value customers. It would make it more accessible for new users to start playing a new game or pick it back up at some point in the future. It also means players will be more inclined to start playing a new game or re-playing an old game as should they tire of it, their time and energy invested into playing will not result in a net economic loss.

Moonbeam’s smart contract architecture and accessibility to the full suite of blockchain functions that will be made possible on Polkadot puts the network in a unique position to be
able to attribute value to videogame items. Naturally, as a blockchain, assets held on Moonbeam will have some value that can be exchanged in and out of real-world value. Chapters 3 through 5 of this project will focus on how this can be done using the technology provided by Moonbeam.
Chapter 3 | System Design

3.1 Approach

To find a way to empower users to have total control and ownership over their metaverse-selves, the feasibility of a video-game based metaverse implementation that enables users to easily create and exchange in-game items across different worlds was examined. The author built a proof-of-concept system using blockchain technology to equip items with a verifiable proof of creation, ownership, and transaction history. The goal was to build a proof-of-concept that could add real-world value to in-game items, allow item transfers between owners, allow players to use the on-chain items in their local game environment, and enable an item’s state to change on-chain based on its use in-game.

This required research of a myriad of blockchain protocols, understanding their strengths and weaknesses for such an implementation, researching videogame architecture, learning how different games created and managed in-game items, learning how to develop smart contracts that could host a heterogenous array of items with different item formats and structures from different games as NFTs, and be able to manage them all on-chain within a singular meta-format. Furthermore, it was required to research and implement a method to update the NFTs on-chain should the item’s state change, and to do all of the above in a game-developer friendly method, such that this system could be tailored to suit any videogame with minimum expertise required on the developer’s part.
3.2 Alternative Approaches & Methods

Usage of Different Videogames

This implementation was built using the Minecraft videogame and its architecture as the author had familiarity with the game and was aware to an extent of the means in which its architecture and codebase could be accessed, read, and manipulated if required. In addition, Minecraft is a very popular videogame and there is a huge volume of documentation and tools available to assist developers in understanding its functions.

There are alternative videogames, in particular Roblox, that would be interesting to utilise to illustrate the proof-of-concept. Roblox is a 3D online game that has official development software, the Roblox Development Studio, designed to assist inexperienced game developers in the creation of Roblox based custom games. This would be particularly interesting to explore as Roblox describes itself as a metaverse platform (Gent, 2021), and new games can be built on the platform that have a slight degree of interoperability between them.

This work’s proof-of-concept could easily be adapted and/or built upon to incorporate Roblox based items, as the adaptability of this work to function easily with any game was a core pillar of the proof-of-concept architecture. The author did not choose to implement this project using Roblox based on the time required to build a whole new Roblox game, which would not serve the central purpose of the proof-of-concept, as well as it being less valuable in the long term than learning how to tailor the proof-of-concept to a pre-existing game.
Usage of Different Blockchains

This work was built on the Moonbeam blockchain due to its significantly lower gas fees and higher transaction throughput than Ethereum, and its integration with the Polkadot relay-chain ecosystem.

There are other chains and ecosystems that could at least be considered as alternatives to Moonbeam. Primarily Moonriver, which is a sister network for Moonbeam that is deployed on the Kusama blockchain, which itself is Polkadot’s sister network. Moonriver and Kusama were deployed before Polkadot, and that ecosystem is built to enable rapid innovation and experimentation so there is a much more developed ecosystem on these chains. At a certain point the author believed that he would have to develop the proof-of-concept on Moonriver as Moonbeam did not launch until January 2022 (Moonbeam, 2022).

Ethereum could also be considered as a viable option for this project given its enormously developed community. The major barriers to deploying on Ethereum were the high gas fees, limited transactions per second and an environmental consideration also, given Ethereum uses significantly higher amounts of energy than a proof-of-stake consensus blockchain. Other chains like Polygon could also have been considered for this project, as they overcome most of the aforementioned issues with Ethereum, however, their lack of presence on the Polkadot ecosystem and therefore limited ability to grow and expand potential use cases into the future was the factor that swayed the author away from developing the proof-of-concept on these chains.
Moonbeam is a solidity-based language which means that this work can be very easily tweaked to function on other solidity-based blockchains, which could be of particular interest when Ethereum 2.0 is fully developed and deployed.

3.3 Features Implemented

Minecraft Item Base
The *Minecraft Item Base* feature is a smart contract developed to act as an intermediary contract through which an item from the Minecraft videogame can be committed to the Moonbeam blockchain. This is the part of the proof-of-concept architecture that can be adapted to any other videogame, such that a “Roblox Item Base” or a “Fortnite Item Base” could exist for items created in those games. The Minecraft Item Base is discussed in chapter 4.3.

Item Access Control

*Item Access Control* is a smart contract that was developed to allow the author to manage any upgrades required to the smart contracts. It also allows the author to initially give rights to Minecraft developers to manage the Minecraft Item Base smart contract, which again could be adapted to any game. *Item Access Control* is discussed in chapter 4.4.

Item Core

The *Item Core* is a smart contract that turns an in-game item into an NFT. It acts as a focal point for the entire system, allowing any item type uploaded to its *Game Item Base* smart contract.
contract to be minted as a homogenous token type, regardless of the highly varied code and traits of different items in different games. Chapter 4.5 discusses the *Item Core* in detail.

**Item Core Interface**

The *Item Core Interface* is an interface, as opposed to a smart contract, that contains the function signatures of the *Item Core* smart contract. The *Item Core Interface* is what facilitates the system’s adaption to different code bases for different in-game items and is discussed alongside the *Item Core* in chapter 4.5.

### 3.4 System Requirements

#### 3.4.1 Functional Requirements

**Facilitate Minting of In-game Items as NFTs**

To meet the objective of attributing real-world value to in-game items, they must be able to be “minted” as NFTs. This inherently creates value as it ensures the items are verifiably unique or in scarce quantity. Furthermore, minting an item as an NFT ensures its history of creation and ownership are publicly viewable and verifiable on-chain.

**Support Standardised Tradability of Heterogenous In-game Items**

For an in-game item’s value to be extractable, it must be able to be bought, sold, and exchanged for other items of value, be they in-game items or not. While minting an item as an NFT natively requires it to be able to transfer ownership, if each item created was of completely different type and format, it makes it difficult for NFT exchanges to recognise and format the item before a sale.
Ensure NFT Items can be used In-game

It is important that once an in-game item is minted as an NFT, it is still possible to use the item within a game in the way that the game supports and does not obstruct the in-game experience of the player.

Enable NFT Items’ On-chain State to be Updated Based on In-game Use

Fundamentally, blockchains and much of the data they support are immutable. At the time of writing, there exists only two ways an NFT’s state can change on-chain (Khurana, 2022). One involves pre-writing conditions into the NFTs code and upon the meeting of those conditions, the NFT changes its appearance or abilities in a deterministic manner. The second method involves creating an NFT which is not upgradeable by default and coding a smart contract that can take that NFT, apply some change to it and mint the updated NFT as a new token, meaning there is now an “original” NFT and an “updated” NFT both in existence.

Neither of these methods are suitable for in-game items which can change in non-deterministic ways, for example, a custom paintjob on a car or random damage to some armour. Having an original and updated version of the same NFT as two separate NFTs also is not suitable as it detracts value from both items due to decreased scarcity. For this reason, a novel method of updating NFTs had to be developed to suit game-based use.
3.4.2 Non-functional Requirements

Security
The smart contracts must be secure, ensuring that certain elements of them can only be invoked by an authorised party. This is essential to stop users creating “overpowered” items that were never intended to be created by the game’s developers and that ruin the gameplay experience for other users.

Adaptability
The system must be easily adaptable to include the minting of any in-game item from any game that wishes to enable it. The optimal system architecture should require the minimum possible blockchain expertise for game developers to be able to allow items from their game be traded using this system.

Simplicity of Understanding
The system must produce in-game items that are easy for other smart-contract based exchanges to read and understand, so that the in-game items can be bought and sold across as many platforms as possible. The system must also ensure the minted in-game items are simple for the player to understand, as without this the purchase behaviour of players will be negatively impacted, as will players’ propensity to use a minted in-game item.

Accessibility
It is important that videogame players feel secure and empowered to be able to easily mint, trade and use their NFT items in-game. It is also important that players can clearly identify who the creators and previous owners of an item were, especially in the case of a creator or
previous owner being of celebrity status. The optimal system architecture details how this can be best achieved.

3.5 System Architecture

3.5.1 Overview

The proof-of-concept architecture implemented in Figure 3.1 is designed to illustrate as best as possible the functionality and feasibility of a video-game based metaverse built within the Polkadot ecosystem. However, as a proof-of-concept, this architecture is not optimal nor without usability issues which are considered.

A significant limitation of the architecture depicted in Figure 3.1 is the infancy of the Polkadot network itself, which is such a novel technology ecosystem that many of its core features have not been fully developed and implemented at the time of writing. Figure 3.2 depicts an improved system architecture, that can be implemented relatively simply in concurrence with the development of the Polkadot and Moonbeam blockchains.

An optimal system architecture is presented in Figure 3.3 that satisfies all requirements for a commercially active version of this system, but also requires more capital, knowledge, and human resources to implement than the author had at his disposal.
3.5.2 Proof of Concept Architecture

The proof-of-concept architecture was implemented using the Moonbeam network infrastructure. It consists of the *Minecraft Item Base* contract, *Item Core Interface*, and the *Item Core* contract. Another game item base contract, “*A.N. Other Item Base*”, which was not implemented, is included in Figure 3.1 to illustrate that multiple other game-item base contracts can exist independently and easily connect to the *Item Core* ecosystem through the *Item Core Interface*.

![Figure 3.1: Proof-of-Concept System Architecture](image)

This architecture has some issues, particularly around its security and ease of use. The data of each in-game item has to be manually extracted from a game when first minting an item, and manually copied back into the back end of the game when importing an on-chain item. Apart from the obvious arduousness of this process, it provides enormous security issues if let open...
to the public, as anyone has the power to tamper with an item’s details while the item transitions between the blockchain and the game. While a well-designed game item base contract should remove the risk of someone creating a game-breaking item, it means that an item’s state can be artificially edited without it being used in-game, which diminishes the value of a blockchain based system quite significantly.

The other issue with this architecture is that while items can be exchanged between owners, there is no public exchange directory on which to browse and trade various in-game items. This is down to the novelty of the Moonbeam blockchain, and it is expected that in the coming months NFT exchanges such as OpenSea will integrate with the network, which will resolve that issue. Similarly, in-game items cannot be sent between parachains on the Polkadot network as the XCMP protocol enabling this feature is not yet fully developed, but again this should resolve itself in the coming months.

### 3.5.3 Improved Architecture

The improved architecture in Figure 3.2 builds upon the proof-of-concept and features a game-based user interface and in-game functions that can communicate with a secure-server, which from there can interact with the *Minecraft Item Base*. It also includes the functionality that will be provided by developments in both the Polkadot and Moonbeam ecosystems regarding XCMP and third-party integrations.
The three game-based functions and GUI to accompany them should be extremely straightforward for an average videogame developer to implement, as should the communication between the game and the secure server. The server can then interact with its corresponding game item base using Moonbeam’s Covalent API, which is “RESTful”, and can be implemented in either Python or JavaScript (Getting Started with the Covalent API, 2022). This architecture will overcome the risk of illegitimate tampering with the items in transition, protecting their value. It will also remove the laboriousness from the process of minting in-game items as NFTs and using them in game, improving the experience for the users.

The author made an attempt at a boot-strapped implementation of this functionality himself using a local server. A third-party modification for the Minecraft videogame called WebDisplays was installed which provides both an in-game user interface and a back end for connecting to internet servers and webpages. The author was unsuccessful with the
bootstrapped implementation of this functionality, however, as the WebDisplays modification cannot run properly on macOS (Barbotin, 2019), the author’s native operating system.

The other aspects of this improved system architecture should come to fruition with time. While it is not guaranteed that the OpenSea exchange specifically will build an integration with Moonbeam, it is probable. Should that integration fail to materialise, another similar exchange will undoubtedly be built on the Moonbeam blockchain, simplifying the trading of NFT items for users. Polkadot’s XCMP protocol will also be implemented in the coming months which will allow the exchange of items between parachains.

3.5.4 Optimal Architecture

![Figure 3.3: Optimal System Architecture](image)
The optimal system architecture depicted in Figure 3.3 depicts a somewhat complex, but highly efficient and extremely user-friendly system design that takes advantage of the full range of possibility on the Polkadot ecosystem. It introduces a custom parachain that connects to the Polkadot relay-chain, and a custom wallet tailored to the custom chain.

**Advantages of the Optimal Architecture**

The custom parachain can be designed in such a way as to remove the need for smart contracts representing the *game item base*, and rather represent these with game specific “modules”. These modules would be very similar to smart contracts, except being built from Polkadot’s Substrate framework, they would be easily upgradeable by authorised accounts. The upgradability of *game item bases* is an essential facet of this architecture should it be deployed commercially, as videogames are constantly updated by adding new item types, or changing the possible attributes of existing item types. For example, a racing videogame may be updated to add a new type of car to the game, or to make an existing type of car faster in-game. Upgradability with smart contract architecture is notoriously cumbersome, and inefficient from a data management perspective as outdated contracts remain on-chain eternally.

The optimal architecture also makes use of features such as its own custom wallet, which should be built with the aim of facilitating the use of the custom parachain and facilitating the use of the system as a whole for non-tech savvy users. Other value-adding features of the optimal architecture include the native support for RMRK NFT standards that a Substrate based parachain will have, as well as the access to the Kilt Protocol when it deploys as a parachain on Polkadot. Kilt provides a means through which names can be associated with an on-chain address, meaning the proof-of-ownership of an item would not involve a list of
unrecognisable blockchain addresses of previous owners, but rather a blockchain based name (Kilt Protocol, 2022). This would make it much easier for users to verify whether someone of celebrity status was a previous owner of an item, adding value to it.

The optimal architecture can also use XCMP to be able to access the item core and continue to mint NFTs using the familiar ERC-1155 standard.

**Issues with the Optimal Architecture and their Solutions**

The most significant issue with the optimal architecture as depicted in *Figure 3.3* is security related. The removal of smart contracts until a later phase in the execution of minting, exporting, or importing an item increases the risk of a denial-of-service attack. This risk can be mitigated by building on Polkadot’s weight-based fee model to offer some form of proportional economic levy to the use of the custom parachain, disincentivising such an attack.

Another issue with the architecture in *Figure 3.3* is an inherent limit within the *Item Core* smart contract, which would still be used to mint ERC-1155 NFTs. This is an issue surrounding the NFTs’ token-IDs, which are created and stored in array. Array data structures in the Solidity language have an upper limit of $1.15 \times 10^{77}$ (or $2^{256}$) indexes, and while this limit is inconceivably large it is an upper limit that is based on the minimal possible computing power required elsewhere in the smart contract (Cuesta Cañada, 2019). This issue can be entirely circumvented by generating the token IDs on the custom parachain, and passing them to the *Item Core* via XCMP, because if an array of token IDs were to near its maximum length on the custom parachain, the chain could simply be upgraded to create an additional array of token IDs.
Chapter 4 | Technical Implementation

4.1 Overview

The technical implementation of this project encompassed a number of key conceptual advancements as well as the actual implementation of the code base itself.

It was clear from relatively early on during the implementation phase that understanding the nature of the code behind in-game items, and determining the best way to manage complex items on-chain, would be a deciding factor in the design of this system. This turned out to be a more significant obstacle than could have realistically been foreseen ahead of time, which is discussed in detail in chapter 5.3, but it forced the technical implementation to take a somewhat innovative approach to NFT creation and management at the time of writing.

The structure implemented involved:

- A Minecraft item base
- Item access control
- An item core and its corresponding interface

The results of this structure are then displayed alongside the user journey that would lead to those results occurring.
4.2 Technology Used

4.2.1 Runtime Environments and Programming Languages

Moonbeam

Moonbeam was the first parachain to be deployed on Polkadot (Moonbeam, 2022), and it brought with it the ability for smart contracts to be created and deployed within the Polkadot ecosystem. Moonbeam was chosen as the most appropriate network on which to deploy smart contracts that implement a videogame based metaverse based on its low gas fees and high potential for future interoperability and upgradability.

Ganache

Ganache is a personal blockchain for rapid solidity based smart contract and dApp development (Ganache | Overview – Truffle Suite, 2022). Ganache was used to test smart contract development and iterate quickly through different versions, allowing for quick alterations to be made without the need to deploy the contracts on a testnet.

JRE

The Java Runtime Environment consists of the Java Virtual Machine (JVM), Java platform core classes, and supporting Java libraries (What is Java?, 2022). The JRE was used to launch, run, and explore Minecraft’s file structures.

Solidity

Solidity is the language in which Moonbeam’s smart contracts are written. It is an object-oriented, high-level language designed specifically for the implementation of smart contracts (Solidity – Solidity 0.8.13 documentation, 2022).
The Java programming language is a general purpose, concurrent, strongly typed, class-based object-oriented programming language (Java Programming Language, 2022). It was used to experiment with Minecraft file structures, and artificially alter Minecraft in-game item information.

4.2.2 Frameworks, Libraries and Services

Truffle
Truffle is a development environment, testing framework and asset pipeline for blockchains using the EVM. It integrates with Ganache and offers built-in smart contract compilation, linking, deployment and binary management (Truffle | Overview – Truffle Suite, 2022). It was used to test the smart contracts in the Ganache environment.

Remix Integrated Development Environment
Remix IDE is an open-source web and desktop application. It provides a rich set of plugins with intuitive “GUIs” to help write and test solidity based smart contracts (Welcome to Remix’s Documentation, 2022). Remix IDE was used to develop and test smart contracts, and it was also used to help display some of the smart contracts’ outputs.

Moonscan
Moonscan is an instance of the Etherscan block explorer that is active on the Moonbase Alpha “testnet”, Moonriver “mainnet”, and Moonbeam “mainnet”. Moonscan provides users and developers with access to network statistics and granular insights into Moonbeam’s EVM
(Moonscan Block Explorer & APIs, 2022). It’s primary use in this project was to check the successful deployment and functionality of contracts on-chain.

**OpenZeppelin**

OpenZeppelin is a library for secure smart-contract development, with standardised contracts that have been thoroughly vetted by the community to ensure they are secure (OpenZeppelin Docs, 2022). *Item Access Control* was built by inheriting an OpenZeppelin contract called Ownable, and ERC-1155 NFT functionality was enabled in the *Item Core* by inheriting an OpenZeppelin smart contract called ERC-1155.

**Universal Minecraft Editor**

The Universal Minecraft Editor is a third-party tool which enables users to read and interpret Minecraft’s game files, which are otherwise encrypted in NBT format (Universal Minecraft Editor, 2022). The Universal Minecraft editor was used to gain insight into the Minecraft codebase and understand the data structures of items that are used.

4.2.3 Videogames

**Minecraft**

Minecraft is a videogame in which players can explore their own completely unique world, try to survive the night, and create anything they can imagine (Minecraft Official Site, 2022). The game has two editions, the “Java Edition”, a PC only version, written in the Java programming language and for which most modifications are built, and a “Bedrock Edition” which is cross-platform, and while it is still possible to modify this edition of the game, there is less pre-existing documentation, and it is harder to access the backend of this game.
Minecraft Java Edition was used as the game to centre the construction of the game item base part of the system around, as it was possible for the author to understand the structure and utility of its in-game items.

4.3 Minecraft Item Base

Outline

The code structure of a Minecraft in-game item was explored to understand its attributes and how best these items could be represented on-chain and interacted with there in a gas efficient manner.

The Minecraft Item Base is an example implementation that is representative of what could be any game-item base, showcasing the essential, solidity types and functions required to be implemented for the proof-of-concept architecture to work.

Minecraft Item Structure

Items in Minecraft Java edition are object oriented, meaning that for each item type that exists, there are a defined set of attributes it can have, and a deterministic set of mechanics it must comply with in-game. Items are stored in JSON format, with an item’s attributes denoted by a set of key-value pairs.
There are over 400 unique items in the Minecraft game, however, they all share the same broad structure, albeit with different attributes. To demonstrate the functionality of the system, an implementation that could handle items of type “sword” was devised. Within this code description, consider that all aspects that are “sword” related, including structs, functions, global constants and variables, could simply be duplicated to represent a pickaxe, chest plate or any other Minecraft item.

*Figure 4.1* illustrates the key-value pairs of a wooden sword at the moment of its creation in-game. The keys for all sword items are the same, the only thing that differentiates between swords at their inception is the item id, name and durability. The enchantment list of a sword has its values always initialised to false for all keys.
Minecraft Item Base Smart Contract

```
import './ItemAccessControl.sol';
import './ItemCoreInterface.sol';

contract MinecraftItemBase is ItemAccessControl {

```

Figure 4.2: Minecraft Item Base Dependencies and Inheritance

The Minecraft Item Base contract has two dependencies as shown in Figure 4.2. Minecraft Item Base is an extension of the Item Access Control contract which will be detailed further in chapter 4.4. This contract ensures that critical functions can only be utilised by users who have permission to do so.

Most aspects of the Minecraft Item Base, however, are accessible by the public as this maximises the potential user base of the system. Figure 4.3 illustrates the Sword struct that is in-built to the smart contract, and its corresponding functions will be detailed below. Any struct, function, constant or variable within the contract can be easily duplicated, with some minor tweaks required, to function for any other Minecraft Item type.

```
struct Sword{
    uint256 swordType; // identified by numeric ID
    uint256 maxDurability;
    uint256 currentDurability;

    //take enchantments in as a fixed-size uint256 array where index 0
    //corresponds to hasMending,
    //index 1 corresponds to hasCurseOfVanishing, etc.
```
A decision was made to implement the item structs using only uint256 (standing for a 256-bit unsigned integer) solidity value types as this type uses the minimum possible amount of gas, and all functional attributes of a Minecraft Item can be represented using an integer of some sort.

Initial approaches to the design were to utilise the strings provided from the item’s in-game JSON file, however, when attempts were made to execute functions using such structs, more complex items such as a sword expended too much gas and the transactions failed. After some research around solidity value types, a decision was taken to take a unit256 based approach to the implementation of an item struct within the Minecraft Item Base smart contract.
mapping(uint256 => bool) public swordInGameStatus;

mapping(uint256 => uint256) public swordIdToGlobalId;

mapping(uint256 => uint256) public globalIdToSwordId;

uint256 swordArray = 0; // index of sword array amongst other arrays

address public itemCoreAddress;

/*** ItemArrays Arcy ***/
Sword[] public swords;

/*
  LIST OF NUMERIC IDs
  wooden_sword 308
  stone_sword 312
  iron_sword 307
  diamond_sword 316
  golden_sword 322
  netherite_sword 604
*/

constructor(){
  swordTypeToMaxDurability[308] = 60;
  swordTypeToMaxDurability[312] = 132;
  swordTypeToMaxDurability[307] = 251;
  swordTypeToMaxDurability[316] = 1562;
  swordTypeToMaxDurability[322] = 33;
  swordTypeToMaxDurability[604] = 2032;
  itemCoreAddress = initItemCoreAddress();
}

Figure 4.4: Minecraft Item Base Global Constants, Variables, Item Arrays and Constructor

There are a number of mapping items that are variable throughout the Minecraft Item Base.

These mappings can be thought of as hash tables, whereby a key input will return its
corresponding value output. Mappings are gas efficient structures in solidity and as such they were chosen as the appropriate data structure to host the data as portrayed above. To briefly explain some of them, the \texttt{swordInGameStatus} mapping takes a sword Id as a key and returns a Boolean indicating whether the item is active in the game or not. If an item’s in-game status is true, it cannot be transferred between owners on chain.

This functionality is fundamental to allowing the in-game item NFTs be upgraded in an efficient manner. The pair of mappings entitled \texttt{swordIdToGlobalId} and \texttt{globalIdToSwordId} are used to flick between what is the globalId of a particular sword item, which is actually its NFT token Id, and its sword Id, which is the local Id of the item in the \textit{Minecraft Item Base}. The reason an item needs two Ids is detailed in chapter 4.5.

Other features in \textit{Figure 4.4} include a unit256 called \texttt{swordArray} that is set to equal 0, which acts as an identifier of the array of sword items should there be other item-types instantiated in the \textit{Minecraft Item Base} smart contract. For example, an array of pickaxe items could have its index set to 1, an array of chest plate items set to index 2, and so on. Again, the importance of this feature will be explained in chapter 4.5. There is an array of type swords to store the sword items as they are created, and the constructor initiates the maximum durability values that a sword of a particular type can have, as well as setting the \texttt{itemCoreAddress} which allows the \textit{Minecraft Item Base} to communicate with the \textit{Item Core} via the \textit{Item Core Interface}.

```solidity
function mintSword(
  uint256 _swordType,
  uint256 _currentDurability,
):
public returns (uint256)

// ensure the swordType provided corresponds to an accepted sword
require(
  isValidSwordId(_swordType)),
  "The item id number provided is not equal to an accepted sword"
);

// ensure current durability is between 0 and the swordType's max durability
require(
  (_currentDurability>=0) && (_currentDurability<=swordTypeToMaxDurability[_swordType]),
  "The sword provided does not have acceptable durability"
);

require(validateSwordEnchantments(_enchantmentsList),
  "The enchantments on this sword are not valid in the vanilla version of minecraft"
);

Sword memory _sword = Sword({
  swordType: _swordType,
  maxDurability: swordTypeToMaxDurability[_swordType],
  currentDurability: _currentDurability,
  hasMending: _enchantmentsList[0],
  hasCurseOfVanishing: _enchantmentsList[1],
  hasFireAspect: _enchantmentsList[2],
  hasKnockback: _enchantmentsList[3],
  hasLooting: _enchantmentsList[4],
  hasUnbreaking: _enchantmentsList[5],
  hasSharpness: _enchantmentsList[6],
  hasSmite: _enchantmentsList[7],
  hasBaneOfArthropods: _enchantmentsList[8]
});

swords.push(_sword);
As detailed in *Figure 4.5* the `mintSword` function takes 3 parameters, the sword type, its current durability, and an array of `uint256` values representing its enchantments, and returns a `uint256` value which represents the new item’s sword Id. The function initially makes three security checks to ensure that the item being minted is first of all of type sword, secondly that it has a durability that is in the acceptable range for its sword type and finally that the enchantments on the sword are valid enchantments. There are two internal functions used to do this that are illustrated in *Figure 4.6*.

Once it is verified that the item being passed is a legitimate Minecraft sword, it is initialised as a sword struct value, added to the array of swords, given its sword Id and its active in-
game status is set to true. It is then minted as an NFT through the Item Core and its respective mappings are updated.

```solidity
function validateSwordEnchantments(
    uint256[9] memory _enchantmentsList
) private pure returns (bool)
{
    //each comparison number is the maximum level of its corresponding enchantment
    bool validEnchantments = true;

    if(_enchantmentsList[0]<0 || (_enchantmentsList[0]>1)) {validEnchantments = false;}
    if(_enchantmentsList[1]<0 || (_enchantmentsList[1]>1)) {validEnchantments = false;}

    //ensure no sword can have smite, sharpness & bane of arthropods, as this is not in vanilla game
        validEnchantments = false;

    if(_enchantmentsList[7]<0 || (_enchantmentsList[7]>5) || ((_enchantmentsList[7]!=0) && ((_enchantmentsList[6]!=0) || (_enchantmentsList[8]!=0))))
        validEnchantments = false;

    if(_enchantmentsList[8]<0 || (_enchantmentsList[8]>5) || ((_enchantmentsList[8]!=0) && ((_enchantmentsList[6]!=0) || (_enchantmentsList[7]!=0))))
        validEnchantments = false;

    return validEnchantments;
}
```
```solidity
function isValidSwordId(uint256 _swordType) private pure returns (bool) {
    bool isValid = false;
    if (_swordType == 308) ||
       (_swordType == 312) ||
       (_swordType == 307) ||
       (_swordType == 316) ||
       (_swordType == 322) ||
       (_swordType == 604) {
        isValid = true;
    }
    return isValid;
}

Figure 4.6: validateSwordEnchantments and isValidSwordID Functions

function exportSword(uint256 _tokenId, uint256 _swordType,
    uint256 _currentDurability,
    uint256[9] memory _enchantmentsList)
    public {
    // verify ownership of that tokenId
    require(ItemCoreInterface(itemCoreAddress).balanceOf(msg.sender, _tokenId) > 0,
        "The account trying to export does not own this token" );
    
    // update its stats
    (address thisAddress, uint256 itemArray, uint256 itemArrayIndex) =
        ItemCoreInterface(itemCoreAddress).getItemInformation(_tokenId);
```
require(thisAddress == address(this), "Item does not correspond to this game-contract");

require(itemArray == swordArray, "The item's local array is not of type sword");

require(swordInGameStatus[itemArrayIndex], "This sword is not currently in game and thus can't be exported");

require(
  isValidSwordId(_swordType),
  "The item id number provided is not equal to an accepted sword"
);

//ensure current durability is between 0 and the swordType's max durability
require(
  (_currentDurability>=0) && (_currentDurability<=swordTypeToMaxDurability[_swordType]),
  "The sword provided does not have acceptable durability"
);

require(validateSwordEnchantments(_enchantmentsList),
  "The enchantments on this sword are not valid in the vanilla version of minecraft"
);

//update the stats of that specific sword item
swords[itemArrayIndex] = Sword(
  swordType:_swordType,
  maxDurability:swordTypeToMaxDurability[_swordType],
  currentDurability:_currentDurability,
  hasMending:_enchantmentsList[0],
  hasCurseOfVanishing:_enchantmentsList[1],
  hasFireAspect:_enchantmentsList[2],
  hasKnockback:_enchantmentsList[3],
  hasLooting:_enchantmentsList[4],
  hasUnbreaking:_enchantmentsList[5],
  hasSharpness:_enchantmentsList[6],
  hasSmite:_enchantmentsList[7],
  hasBaneOfArthropods:_enchantmentsList[8]
);
//set state to not active in game
swordInGameStatus[itemArrayIndex] = false;
ItemCoreInterface(itemCoreAddress).updateItemInGameStatus(
    address(this),
    msg.sender,
    swordIdToGlobalId[itemArrayIndex],
    swordInGameStatus[itemArrayIndex]
);
}

Figure 4.7: Export Sword Function

The exportSword function takes the same three parameters as the mintSword function plus one more, a uint256 representing a token Id. This is the NFT token Id of the item.

The exportSword function has the ability to update the on-chain data of the NFT item, ensuring that when an item is exported from a game onto the blockchain, its latest in-game data is represented on-chain, in this case if the sword had taken damage or had been enchanted, this would need to be represented within its on-chain NFT.

The function uses the Item Core Interface to ensure that the person trying to export this item from the game is the owner of the item and ensures that the item’s identifying data is accurate, which is explained in more detail in chapter 4.5.

Security checks are once again performed using the functions in Figure 4.6 to verify the validity of the sword to be exported, and once verified as valid the sword’s in game status is updated to false, indicating that it is not being used in-game. The Item Core Interface is then used to update the in-game status of the item at NFT level, allowing the item to be traded between owners.
function importSword(uint256 _tokenId) public {

    // verify ownership of item
    require(
        ItemCoreInterface(itemCoreAddress).balanceOf(msg.sender, _tokenId) > 0,
        "The account trying to import does not own this token"
    );

    (address thisAddress, uint256 itemArray, uint256 itemArrayIndex) =
    ItemCoreInterface(itemCoreAddress).getItemInformation(_tokenId);

    require(thisAddress == address(this), "Item does not correspond to this game-contract");

    require(itemArray == swordArray, "The item's local array is not of type sword");

    require(!swordInGameStatus[itemArrayIndex], "This sword is currently in-game and thus can't be imported");

    // change in-game status to "in-game"
    swordInGameStatus[itemArrayIndex] = true;
    ItemCoreInterface(itemCoreAddress).updateItemInGameStatus(
        address(this),
        msg.sender,
        swordIdToGlobalId[itemArrayIndex],
        swordInGameStatus[itemArrayIndex]
    );
}

Figure 4.8: Import Sword Function

The `importSword` function is quite straightforward, it takes only the NFT’s token Id as a parameter. The function first ensures the person trying to import the sword to their game is the owner of it. It then checks to ensure that item’s identifying data is accurate, both of these checks are done through the Item Core Interface. Assuming everything is in order, the sword’s status is set to in-game at both the Minecraft Item Base level and at the Item Core NFT level.
There are two other functions that are implemented within the *Minecraft Item Base* smart contract, displayed in *Figure 4.9*.

```solidity
function initItemCoreAddress() private pure returns (address) {
    return 0x3dC8Acb13Ee9EF93D83e0BD253819e83688B6843;
}

function updateItemCoreAddress(address _newItemCoreAddress) public onlyMinecraftExecutive{
    itemCoreAddress = _newItemCoreAddress;
}
```

*Figure 4.9: Initialise and Update Item Core Address Functions*

The *initItemCoreAddress* is an extremely simple private function which simply returns the latest address of the *Item Core* contract when the *Minecraft Item Base* contract is deployed. The *updateItemCoreAddress* requires an address parameter to be passed which should represent the updated address of the *Item Core*, should that be required. This function is limited to only be callable by a “Minecraft Executive” which is explained in chapter 4.4.

### 4.4 Item Access Control

*Item Access Control* is a smart contract that supports ownership and control of other smart contracts by certain privileged accounts. It is a relatively straightforward piece of code, which inherits some functionality from a contract in the OpenZeppelin library. This smart contract is, however, very important as it has the ability to limit the damage done by any mal-intentioned entity.
By inheriting this contract, other contracts within the system architecture can limit the use of functions, that if called incorrectly would break the entire system, to a small set of trusted accounts. It also allows for these accounts’ privileges to be removed at a certain point in the future and supports giving these privileges to previously unprivileged accounts. This is implemented in a way that only an account that is currently in a privileged position can give or remove privileges from other accounts.

**Item Access Control Smart Contract**

```solidity
import "@openzeppelin/contracts/access/Ownable.sol";

contract ItemAccessControl is Ownable{

```

![Figure 4.10: Item Access Control Dependencies and Inheritance](image)

The *Item Access Control* contract is an extension of the *Ownable* contract as provided by OpenZeppelin, which allows for an account that owns a contract to carry out certain privileged functions within it (OpenZeppelin Docs, 2022).

*Item Access Control* provides modifiers for other functions which when invoked ensure that those functions can only be executed by authorised accounts. It also provides some functions to manage the array of privileged accounts.

```solidity
address public ceoAddress;

//let these addresses correspond to official roles, eg index [0] = MinecraftCEO, [1] = MinecraftCTO etc.
address[5] public MinecraftExecutiveAddresses;
```
// @dev Keeps track of whether the contract is paused. When that is true, most actions are blocked
bool public paused = false;

bool public MinecraftPaused = false;

constructor()
{
    ceoAddress = msg.sender;
    MinecraftExecutiveAddresses[0] = msg.sender;
}

Figure 4.11: Item Access Control Global Variables and Constructor

*Item Access Control* contains a generic CEO address, which is always initialised to the person that initiates the contracts, as is the role of Minecraft CEO, whose privilege is only implemented in the *Minecraft Item Base* contract.

The *MinecraftExecutiveAddresses* array allows for up to 5 Minecraft executives to be given access to privileged operations (see *Figure 4.9*). The reason this is a fixed-size array is it limits the potential iterations through an array, which massively reduces gas consumption. It limits the risks of a potential attack as there are only a small number of addresses with these executive authorisations, it also reduces the memory usage required as the array is small, and it allows for every index within the array to correspond to an address with particular special privileges. For example, the Minecraft CEO address is stored at index 0 of the *MinecraftExecutiveAddresses* array.
Figure 4.12: Minecraft Executive Modifiers

Figure 4.12 depicts the two modifiers that control whether a function, such as that depicted in Figure 4.9, can only be executed by privileged accounts.

```solidity
modifier onlyMinecraftCEO()
{
    require(msg.sender == MinecraftExecutive[0]);
}

modifier onlyMinecraftExecutive()
{
    require(_newMinecraftExecutive != address(0));
    require(!isAlreadyMinecraftExecutive(_newMinecraftExecutive), "This account is already a minecraft executive");
}

function addNewMinecraftExecutive(address _newMinecraftExecutive)
public external onlyMinecraftExecutive()
{
    uint i = 1; // i = 1 to start after CEO address which is at i=0
    bool executiveSet = false;
    while((i < MinecraftExecutive.length) && (!executiveSet))
    {
        if(MinecraftExecutive[i] == address(0))
        {
            MinecraftExecutive[i] = _newMinecraftExecutive;
            executiveSet = true;
        }
        i++;
    }
    if((!executiveSet) && (i == MinecraftExecutive.length))
    {
        // Handle the case where none of the addresses were set as executives.
        // Maybe throw an error or logic to handle this.
    }
}
```
function removeMinecraftExecutive(
    address _minecraftExecToBeRemoved
)
    external onlyMinecraftExecutive{
        // ensure CEO is not getting removed
        require(_minecraftExecToBeRemoved != MinecraftExecutiveAddresses[0]);
        require(MinecraftExecutiveAddresses.length > 1);
        bool successfullyRemovedAddress = false;
        uint i = 1;
        while((i<MinecraftExecutiveAddresses.length) && (!successfullyRemovedAddress)){
            if(MinecraftExecutiveAddresses[i] == _minecraftExecToBeRemoved){
                MinecraftExecutiveAddresses[i] = address(0);
                successfullyRemovedAddress = true;
            }
            i++;
        }
    }

function setMinecraftCEO(
    address _newMinecraftCEO
)
    external onlyMinecraftCEO{
        require(_newMinecraftCEO != address(0));
        if(isAlreadyMinecraftExecutive(_newMinecraftCEO)){
            bool swapped = false;
            uint i = 1;// again starting at one to not parse the current CEO
            while((i<MinecraftExecutiveAddresses.length) && (!swapped)){
                if(MinecraftExecutiveAddresses[i] == _newMinecraftCEO){
                    MinecraftExecutiveAddresses[i] = MinecraftExecutiveAddresses[0];// keeps CEO as an executive
                    swapped = true;
                }
            }
        }
MinecraftExecutiveAddresses[0] = _newMinecraftCEO;

Figure 4.13: Privileged Account Altering Functions

The functions illustrated in Figure 4.13 detail functions which can add a new Minecraft executive address, remove an address from the list of Minecraft executives, and set a new Minecraft CEO, respectively.

The three functions are somewhat alike. The functions to add and remove a Minecraft executive address are only implementable by other Minecraft executives, meaning at least one executive must approve the addition of another executive account which adds some security. The account to set a new CEO can only be called by the existing CEO, which means that the CEO cannot be usurped by other executives, however, it also allows for minimal CEO accountability. This could be changed to incorporate some form of on-chain governance as provided by Moonbeam, that should a referendum take place to replace the CEO, a new CEO would automatically be assigned, without the need for approval of the existing CEO.

*Item Access Control* does have some other simple modifiers and small functions such as *isAlreadyMinecraftExecutive* which checks if an address is already contained within the array of Minecraft executives and returns a Boolean indicating whether it has or has not. The other modifiers and functions are similarly intuitive, and consequently the author will not overly analyse them. These modifiers and functions are included for viewing below in Figure 4.14.
modifier whenMinecraftNotPaused() {
    require(!MinecraftPaused);
    _;
}

modifier whenMinecraftPaused {
    require(MinecraftPaused);
    _;
}

function pauseMinecraft() external onlyMinecraftExecutive whenMinecraftNotPaused {
    MinecraftPaused = true;
}

function unpauseMinecraft() public onlyMinecraftCEO whenMinecraftPaused {
    // can’t unpause if contract was upgraded
    MinecraftPaused = false;
}

function isAlreadyMinecraftExecutive(address _newAddress) private view returns(bool) {
    require(_newAddress!=address(0));
    bool isExectuive = false;
    for(uint I = 0; i<MinecraftExecutiveAddresses.length; i++){
        if(MinecraftExecutiveAddresses[i]==_newAddress){
            isExectuive = true;
        }
    }
    return isExectuive;
}

Figure 4.14: Miscellaneous Other Item Access Control Modifiers and Functions
4.5 Item Core and Item Core Interface

Outline

The Item Core smart contract is the central hub around which the entire proof-of-concept architecture revolves. Through the Item Core Interface, it enables the minting of in-game items as ERC-1155 NFTs on the Moonbeam blockchain and implements novel functionality that allows each standardised token to represent a heterogenous array of in-game items from entirely different games which likely have entirely different characteristics.

It also enables innovative functionality that facilitates these items’ states to be updated, reflecting their most recent use in-game. It supports these functionalities while still maintaining the transferability that comes as expected with ERC-1155 tokens.

**Item Core Smart Contract**

```solidity
class ItemCore is ERC1155, ItemAccessControl {
    // Constructor
    ItemCore {
        // Delegate to Item Access Control
    }
    // 4.5.1 minting and id assignment
    function mintItem(uint256 _itemId, string memory _itemName, uint256 _quantity) public {
        // Implement ERC1155 mint function
    }
    // Other functions...
}
```

Figure 4.15: Item Core Dependencies and Inheritance

Figure 4.15 highlights the inheritance of both the ERC-1155 and Item Access Control contracts to the Item Core smart contract. The ERC-1155 contract is provided by OpenZeppelin and implements the methods required by the ERC-1155 token standard which supports the minting of both fungible and non-fungible tokens making it a highly gas-efficient contract (ERC-1155 – OpenZeppelin Docs, 2022).
The `itemIdentity` struct and its corresponding array called `items`, as shown in Figure 4.16, is what allows the system architecture to function in the unique way that it does. Each in-game item that is minted as an NFT is represented on-chain through an instance of an `itemIdentity` in the Item Core. Each item identity contains 4 key references, that allow the immediate identification of the in-game item it represents at any instant. The `gameContractAddress` points to the game item base smart contract, such as the Minecraft Item Base, the `gameItemArray` points to the item array within that contract, such as the swordArray in Figure 4.4, and the `gameItemArrayIndex` points to the local index of the in-game item within that array. The final element of the struct, `activeInGame` is a simple Boolean that allows the Item Core to keep track of whether an item is currently in-game or on-chain.
The array of type `itemIdentity`, called `items`, in *Figure 4.16* is implemented in the maximally efficient way evidenced in *Figure 4.18*. Due to the minimised computational effort required to host and run the functions alongside this array in the *Item Core* smart contract, the array limit will be close to its maximum capacity of \(1.15 \times 10^{77}\) (or \(2^{256}\)) indexes, ensuring that the proof-of-concept architecture is as optimised for as much longevity as possible.

```solidity
function mint(
   address _gameContract,
   address _userAddress,
   uint256 _gameItemArray,
   uint256 _gameItemArrayIndex,
   uint256 _amount,
   bool _activeInGame
)
public
returns (uint256)
{
  itemIdentity memory newItemIdentity = itemIdentity(
    gameContractAddress: _gameContract,
    gameItemArray: _gameItemArray,
    gameItemArrayIndex: _gameItemArrayIndex,
    activeInGame: _activeInGame
  );

  items.push(newItemIdentity);
  uint256 tokenId = items.length - 1;
  _mint(_userAddress, tokenId, _amount, "");
  return tokenId;
}

function transferItemOwnershipFrom(
   address _from,
   address _to,
```
public {
    require(!items[_tokenId].activeInGame,
        "This item is active in game, please export the item before transferring it");
    _safeTransferFrom(_from, _to, _tokenId, _amount, "0x00");
}

function getItemInformation(uint256 _tokenId) public view returns (address, uint256, uint256)
{
    return (items[_tokenId].gameContractAddress,
        items[_tokenId].gameItemArray,
        items[_tokenId].gameItemArrayIndex
    );
}

function updateItemInGameStatus(
    address _gameContract,
    address _userAddress,
    uint256 _tokenId,
    bool _newStatus
) public {
    require(balanceOf(_userAddress, _tokenId)>0);
    require(_gameContract == items[_tokenId].gameContractAddress);
    items[_tokenId].activeInGame = _newStatus;
}

**Figure 4.18: Item Core Functions**

*Figure 4.18* contains four functions which together guarantee the NFT operation of the proof-of-concept implementation.
The `mint` function takes in 6 parameters which include addresses of the game item base contract and the user who wishes to mint their in-game items, three uint256s representing the item’s array on the game base contract, the index of the item in that array, and the number of identical items the user is minting in this batch, as well as a Boolean to indicate whether the item is active in game or not. The function proceeds to create a token of the item that is minted on the chain, and returns its token Id.

The `transferItemOwnershipFrom` function requires parameters of the addresses of the sender and recipient, the token Id of the item being transferred, and the amount of those tokens being transferred. If the item is an NFT as opposed to a fungible token the amount should almost always be one. It ensures the item is not active in game and then calls the ERC-1155 `safeTransferFrom` method, sending the tokens to the recipient specified.

`getItemInformation` does not execute a transaction and therefore does not alter the state of the chain. This function simply takes a token Id and returns its corresponding item’s identifiers.

The final function, `updateItemInGameStatus`, ensures that the person attempting to update an item’s in-game status is the owner of the item, and that the item is being updated from the right game base contract. It then simply updates the item’s status to be active or not active in game. For this contract if an item is ‘not active in-game’ it means that it is currently on-chain and therefore can be transferred.

**Item Core Interface**
The *Item Core Interface* is not a smart contract. It is rather an interface to the *Item Core* smart contract (see *Figure 3.1*). When referenced by an external smart contract, interfaces provide a means for non-related smart contracts to invoke the functions of the interfaced smart contract, as long as the contract calling the functions has the correct address of the contract which implements those functions.

```solidity
interface ItemCoreInterface {

    function mint(
        address _gameContract,
        address _userAddress,
        uint256 _gameItemArray,
        uint256 _gameItemArrayIndex,
        uint256 _amount,
        bool _activeInGame
    ) external returns (uint256);

    function balanceOf(
        address _address, uint256 _tokenId
    ) external view returns (uint256);

    function getItemInformation(
        uint256 _tokenId
    ) external view returns (address, uint256, uint256);

    function updateItemInGameStatus(
        address _gameContract,
        address _userAddress,
        uint256 _tokenId,
        bool _newStatus
    ) external;
}
```

*Figure 4.19: Item Core Interface*
The *Item Core Interface* (*Figure 4.19*) contains representations of the *Item Core’s* functions depicted in *Figure 4.18*. The difference being the *Item Core Interface* contains no implementation of the functions, only their names, parameters and expected return types.

It is through this interface that the decentralisation of the proof-of-concept architecture is enabled. The interface allows myriad items with different idiosyncrasies to be implemented through the one contract. By allowing different games full autonomy over which of their items they allow on-chain, and how they wish to handle them, the proof-of-concept architecture provides a framework that simplifies the exchange of items for users and the management of items for game developers.

*Figure 4.7* showcases this, wherein the *exportSword* function makes calls to the *Item Core* by invoking the *Item Core Interface* at the address associated with the *Item Core* contract.

### 4.6 Results of Proof-of-Concept Implementation

**Outline**

The results of the proof-of-concept in this work will be presented in the form of user journeys. The journeys that will be presented will be that of two Minecraft players, Aisling and Barry, and they involve the key functions of minting, exporting, transferring ownership of, importing and exporting an updated version of a Minecraft wooden sword.
The author is aware that in reality, the proof-of-concept offers a sub-optimal user experience to Aisling and Barry, however, the purpose of this proof-of-concept is not to demonstrate an optimal user experience, like those described in the Improved and Optimal System Architecture (chapters 3.5.3 and 3.5.4, respectively), but rather is to prove that a game-based metaverse using blockchain is feasible, and that the Moonbeam ecosystem is a suitable environment for such a metaverse to be built upon.

Furthermore, the author wishes to point out that the smart contracts for the proof-of-concept were deployed and run on Moonbeam’s testnet, Moonbase Alpha, as there was no tangible benefit for the proof-of-concept implementation of deploying the contracts on the Moonbeam mainnet. The testnet’s native currency is DEV as opposed to GLMR, so to calculate the cost of transactions had they been executed on the mainnet, the transaction fee accrued by each transaction in DEV will be multiplied by the price of one GLMR in US dollars depicted in Figure 2.19.

Aisling is a world-famous Minecraft player, she has recently hit over 10 million YouTube subscribers on her channel, and she holds multiple world-records for beating the game in record time. She is beginning a new Minecraft series on her YouTube channel to say thank you to her fans for helping her reach the 10 million subscriber milestone, and within the first 10 minutes of her first video she has created her first wooden sword in-game (Figure 4.20). Aisling wishes to give this exact sword away to a fan using the proof-of-concept architecture described in this work.
Minting an Item

In the absence of a server and GUI as described in the improved system architecture chapter 3.5.3, Aisling copies the wooden sword item’s data out in the correct format and calls the \texttt{mintSword} function on the \textit{Minecraft Item Base} contract, passing the sword’s formatted data to the function as parameters to the contract’s Moonbeam address.

\textit{Figure 4.21: mintSword Function Call}

This transaction is executed, and costs Aisling a fee of 0.00096 DEV (\textit{Figure 4.22}) which equates to a cost of $0.0044 for Aisling.
Aisling’s sword now has a token Id, which happens to be 0 as it is the first item minted on this system (Figure 4.23), and her account’s balance of that token Id is 1, indicating that Aisling now owns the sword she just minted (Figure 4.24).

![Figure 4.23: New Sword Token Id](image1)

![Figure 4.24: Aisling’s Account Balance](image2)

**Exporting an Item**

Before Aisling can transfer the item to her number one fan, Barry, she must first export the item from her game, because, as shown in the bottom of Figure 4.23, the NFT’s status is active in game. Once more she formats the data correctly and passes it as parameters to the
Minecraft Item Base contract address on Moonbeam, this time calling the `exportSword` function, with her token Id of 0 (Figure 4.25).

![exportSword Function Call](image)

The execution of this exporting transaction is successful (Figure 4.26), costing Aisling 0.00026 DEV which can be equated to $0.0012 in transaction execution fees. The sword’s in-game status has been switched to false (Figure 4.27), meaning that the item is now active on-chain and can be transferred securely between owners.

![Successful Exporting of an Item on Moonbeam Testnet](image)
Transferring Ownership of an Item

Now that the sword has been exported, Aisling wishes to send it securely to Barry. She asks Barry for his Moonbeam address, which he provides, and she then calls the `transferItemOwnershipFrom` function at the `Item Core` contract address with the parameters filled in as they are in Figure 4.28. The “from” parameter, is Aisling’s address, while the “to” parameter is Barry’s address, the token Id is that of the sword that Aisling wants to send to Barry, which is 0, and the amount is 1, as the sword is an NFT and so it is one of a kind, only one exists.

Once again the transaction is successful (Figure 4.29), costing Aisling 0.00017 DEV, which equates to $0.0008. Aisling verifies that she no longer owns the sword by checking that her
account balance of tokens with token Id = 0 is empty, which it is (Figure 4.30). For peace of mind, Aisling asks Barry to ensure that his account balance of tokens with token Id = 0 is 1, and she is content when he lets her know that it is (Figure 4.31).

![Figure 4.29 Successful Transfer of Ownership of an In-game Item]

![Figure 4.30 Updated Token Balance of Original Account]

![Figure 4.31 Updated Token Balance of New Owners Account]
Importing an Item to a New World

Barry feels very lucky, as he knows that to be the owner of a Minecraft Sword previously created and used by Aisling is to be the owner of a very valuable asset. He knows he should refrain from using the item in-game, as that could depreciate its value, but his excitement gets the better of him and he imports the item into his Minecraft game using the `importSword` function of `Minecraft Item Base` contract, by simply inputting the item’s token Id (Figure 4.32). This updates the item’s in-game status to be active in-game (Figure 4.33), meaning it cannot be transferred on-chain at this time. The success of the transaction that allows the importing of the sword costs Barry 0.00033 DEV (Figure 4.34) or $0.0015.

![importSword Function Call](image1.png)

*Figure 4.32: importSword Function Call*

![Updated Item In-game Status after Import to Game](image2.png)

*Figure 4.33: Updated Item In-game Status after Import to Game*
Exporting an Item with Updated Statistics

Barry then uses this item in his local Minecraft game, to fight a spider monster that is attacking him. Using the sword item to fight the spider lowers its durability statistic (Figure 4.35).

Figure 4.34: Successful Import of NFT Item into Game

Figure 4.35: An Item’s Statistics Updating Through Use In-game
Barry does not want the sword to take any more damage, as he does not want the price to depreciate further, so he copies the sword’s in-game data and calls the `exportSword` function at the Minecraft Item Base contract address, passing the sword’s updated statistics along with its token Id as parameters (Figure 4.36).

![Figure 4.36: Calling the exportSword Function with Updated Item Statistics](image1)

The transaction is successful, and all is right once more in Barry’s life as he is now able to sell on this item, previously owned by the celebrity that is Aisling, for a good, albeit likely depreciated, price. The cost to him of re-exporting this in-game item was 0.00026 DEV (Figure 4.37), which is worth roughly $0.0012.

![Figure 4.37: Successful Export of Item with Updated Item Statistics](image2)
**Outcome**

The above user journey describes a simplistic narrative, yet it does highlight one significant use case of the proof-of-concept architecture, in that it is possible for videogame players to play games with the exact items their favourite video game players have used. This can be thought of much like being able to play football with a pair of football boots worn and signed by Lionel Messi in the Champions League final.

Furthermore, it should come across in the user journey that this system has been designed with a heavy focus on transaction fee optimisation. While the price of the GLMR token is volatile, and the transaction fees on the Moonbeam mainnet can be different at times to those on the testnet, it is unlikely that discrepancy between the transaction fees are so different as to radically alter the cost of executing these highly gas-efficient transactions (McLoughlin, 2022).

With that in mind, we should consider that all of the above transactions cost Aisling a cumulative $0.0064, or just over half a cent. Barry’s transactions fees were also so low as to be trivial, costing $0.0027, which is just over one quarter of a cent.

The low cost of the implementation of this proof-of-concept architecture clearly indicates that a video-game based metaverse on Moonbeam is absolutely feasible from an economic perspective.
Chapter 5 Chapter 5 | Evaluation

The evaluation of this work will discuss the challenges faced in the development of the proof-of-concept architecture. It will then proceed to evaluate the results detailed in chapter 4.6 against the system requirements set out in chapter 3.4. Finally, the evaluation section will conclude with a brief discussion about more macro-challenges faced throughout the process of this work.

5.1 Proof-of-Concept Challenges

In-game Item Code Handling on a Blockchain

A significant challenge faced very early on in the implementation of the proof-of-concept architecture was the challenge of taking the code that is used to represent an item in-game, and finding a way to represent this code on the blockchain in a gas-efficient manner. The author made multiple attempts at this before ultimately forming the method that is implemented in chapter 4.

An initial approach was focused on instantiating items on-chain using hexadecimal code. The hexadecimal code brought some success, but it ultimately was unviable for two main reasons. Since hexadecimal code contains digits 0 through 9 but also letters A through F, it could not (reliably) be represented using a simple uint256 type, requiring instead the slightly volatile “bytes” type. Bytes can be used to represent any data in solidity, but this functionality comes at a cost of significantly increased memory usage and therefore gas fees, to the point that more complex items became infeasible to implement through hexadecimal.
A second attempt was made to store the in-game items as two corresponding arrays of strings, one representing the items’ keys and the other their values from the JSON format. This again required the use of the bytes type to store strings, and the author learned very quickly that implementing strings in a smart contract should be avoided at all costs if gas optimisation is to be achieved. It was the learnings from the initial two attempts that drove the author in the direction of the uint256 based approach that is used in the final proof-of-concept implementation.

**Enabling Game Item NFTs to be Updated with Usage Statistics**

Significant challenges were encountered when attempting to implement the upgradability feature of the NFTs, as any other contracts or applications that allow for a similar method of NFT upgradability at the time of writing are unknown to the author. The author was familiar with some methods of NFT upgradability as described in chapter 3, but neither of these two approaches were appropriate for the proof-of-concept of this work. The author took some conceptual inspiration for the proof-of-concept’s decentralised token minting system from the Polkadot network’s architecture, where most of the “heavy-lifting” of the ecosystem takes place away from the central relay-chain, but the central relay-chain is the core piece of architecture that glues the whole network together. The relationship between the game item base contracts and the Item Core contract was thought of as being similar to that of the parachains and the relay-chain. Following some trial and error the author managed to establish such a system within the proof-of-concept work.

**Editing In-game Code**

The author attempted to expand the proof-of-concept architecture and implement some aspects of the improved architecture of chapter 3.5.3. The motivation behind this was largely
for demonstration purposes. Ultimately, this attempt fell through in part due to its lack of necessity for proof-of-concept functionality, but more so due to the inability of the WebDisplays Minecraft modification to function correctly on the author’s native operating system, macOS (Barbotin, 2019).

5.2 Fulfilment of Requirements

5.2.1 Functional Requirements

The functional requirements of the proof-of-concept system, as described in chapter 3.4.1, were all successfully implemented in this work.

Facilitate Minting of In-game Items as NFTs

In-game items were successfully minted as NFTs on the Moonbeam testnet, evidenced in Figure 4.22.

Support Standardised Tradability of Heterogenous In-game Items

Using the Minecraft Item Base or any other game item base contract to connect to the Item Core through its corresponding interface, enabled heterogenous in-game items from any videogame to be minted in a standardised NFT format. Once minted these items can be transferred between owners, implemented in Figure 4.29.

Ensure NFT Items can be used In-game
The Boolean logic that disables the tradability of an NFT if it is active in-game ensures that items can be traded and used in-game in a safe and secure manner. The success of which is highlighted in Figures 4.32 through 4.35.

**Enable NFT Items’ On-chain State to be Updated Based on In-game Use**

An innovative method of updating NFTs was devised and implemented in the proof-of-concept system, illustrated in *Figure 4.36 and Figure 4.37.*

### 5.2.2 Non-functional Requirements

**Security**

The proof-of-concept architecture is secure. It ensures that only authorised accounts can carry out critical tasks through the *Item Access Control* contract. It also ensures that no item can be created with statistics that could not be naturally affiliated with that specific item.

**Adaptability**

The proof-of-concept architecture is highly adaptable. The functions and data structures implemented in the *Minecraft Item Base* contract can be easily duplicated and only require minor changes to allow any other Minecraft item to be minted. The contract as a whole can be copied and used as a framework for other game item base contracts, whose exact functions can be easily tailored to that specific game’s items.

**Simplicity of Understanding**

The system creates standardised ERC-1155 NFTs and as such should be readable by any exchange or visualisation tool that supports that format.
Accessibility

The accessibility achieved is as good as is possible in the proof-of-concept work. Ideally, an optimal architecture would exist in an environment that would allow Moonbeam addresses to be associated with recognised names, simplifying the visualisation of creator and owner history for an item.

5.3 General Difficulties Encountered

Volume of Information to Consume

At the beginning of the college year, it would be accurate to say the author’s knowledge of blockchain technology was very limited. The author was vaguely familiar with the concept of a blockchain, in that it was the security mechanism that allowed cryptocurrencies to exist, and he had also seen the term NFT utilised frequently regarding digital art, without wholly understanding why that was.

The leaps required in his understanding of this technology to even begin to grasp the scope of this project, necessitated that the author take in vast swathes of information very quickly. Beginning by understanding the functionality of a blockchain, the author rapidly took onboard information about the innovation that was the Ethereum network and its solidity language, followed by the even further step forward that the Polkadot network was taking. This information onslaught was also occurring concurrently to Polkadot’s parachain slot auctions, which was the first time the author had come across Moonbeam. This all ensured it was critical that the author dedicated substantial amounts of time to understanding blockchain
systems and their individual characteristics before beginning to implement the proof-of-concept architecture.

**Hardware issues**

The author encountered some hardware issues in the project’s implementation phase. Namely that his then laptop was unable to properly run either Ganache or Minecraft. It certainly was not able to run them both simultaneously. This forced the purchase of a new physical machine, with comparably enormous processing power and the state-of-the-art Apple M1 Chip. Perhaps unsurprisingly, the new hardware drastically increased the speed at which the proof-of-concept could be developed and tested.

---

**Chapter 6 Chapter 6 | Conclusion and Future Work**

**6.1 Conclusion**

The reason the Polkadot blockchain was built was to help give individual people control, ownership, and sovereignty over their data, removing the need for society to trust gargantuan centralised technology companies to handle their data in a way that is “not evil” (Polkadot Lightpaper, 2022). The author strongly identifies with this vision. The goal of the author throughout this work was to elucidate the possibility that should the metaverse come into existence in the way that is expected, the people who use it will be able to own the things that they use within it.
Through the building and testing of a proof-of-concept system that enables individuals to buy, sell and trade items they have earned by playing any videogame, the author has shown that it is indeed possible for players to truly have ownership over the things that they use in a digital environment. This is pertinent given the integral role that videogames and videogame technology are likely to play within the metaverse (D’Anastasio, 2022).

This work also detailed how the proof-of-concept architecture can be expanded upon to take maximum advantage of the power of the Polkadot network in delivering a system that can enable any videogame player to easily realise the value of the items they have earned in-game.

The Polkadot network, like blockchain technology as a whole, continues to develop rapidly. It is the author’s hope and belief that as this technology grows and becomes more user friendly, more and more people will easily be able to use it to take control of the digital assets they create through their time spent in the digital sphere.

6.2 Future Work

The metaverse is an open-ended concept at the time of writing. Yet it is a concept that has the promise of empowering people to exist more freely in the digital world, to connect people around the world and to add more value to the time that people are spending online. It is critical therefore, that the metaverse is decentralised in nature. No single organisation, corporate or governmental, should have outright control over the metaverse and the people who use it. It must be a place that is controlled and governed democratically by its users.
Future work should consider the best ways to achieve this whether through blockchain technology or another means.

Future work specifically surrounding the architecture described in this work should focus on improving the user-experience of people who use the technology. This could be done initially by adopting some videogame development skills and integrating one or a number of games in with the proof-of-concept architecture, supporting players in using game-based user interfaces to manage their assets. A further step would be to attempt to build the system as its own parachain on Polkadot, enabling upgradability of the on-chain game item bases.

**Bibliography**


Canales, K., 2022. *Ethereum is about to get a huge facelift that could help fix crypto's climate problems — but spell the end of ether mining*. [online] Business Insider. Available at:


Huobi Global, 2021. *How to Vote in the Polkadot Parachain Slot Auction for Your Favorite Project?*. [online] Medium. Available at: <https://medium.com/huobi-global/how-to-vote-in-
the-polkadot-parachain-slot-auction-for-your-favorite-project-87acc5a728ce> [Accessed 10 April 2022].


Qian, H., Li, X. and Ma, J., 2021. LVRT: Low Variances of Solo Mining Reward & inter-block Time in Collaborative PoW. *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom).*


Solidity Docs. 2022. *Solidity — Solidity 0.8.13 documentation*. [online] Available at: <https://docs.soliditylang.org/en/v0.8.13/> [Accessed 10 April 2022].


