Chapter 1

Problem Statement

1.1 The Problem

Video games are often large and highly popular pieces of software that are typically distributed for developers by a third party platform like Steam or Epic Games. Whilst these platforms provide benefits such as availability, and some social features they have some major downsides that include:

(a) taking a large cut of all revenue,
*Steam take a 30% cut [**?**, **?**]*
(b) being vulnerable to censorship from governments,
*The Chinese version of Steam is heavily censored [**?**]*
(c) the user's access to their games is linked to the platform.
If the platform shuts down, the user loses all their games

1.2 Goals

The goal of this project is to implement a large-scale distribution platform that will allow game developers to release and continuously update their games on a public network by directly interacting with their users. This is in the aim to provide greater profits to developer's, freedom from censorship, and better digital ownership for the user.

1.3 Scope

This project will look at how the Ethereum blockchain and smart contracts can be used to create a large-scale distribution platform for video games. This will be deployed to a 'testnet', where Ether has no value and applications can be tested in a live environment. Whilst the focus on this application is on video games, this should be a valid solution for software of all types.

The application will consist of a set of smart contracts, written in Solidity, with tests and a basic GUI to be written using TypeScript and React.js. Tools like Ganache and Truffle will also be useful for blockchain development.

Chapter 2

Background Research

2.1 BitTorrent

It is an unrealistic to expect that every game uploaded to the network will be downloaded by every user so only a subset of users will have the game installed and available to share. In this section and Section 3.2, I will look at how various peer-to-peer file-sharing networks allow users to discover and download content that is fragmented across the network.

BitTorrent [**?**, **?**] is the most popular P2P file-sharing platform, in which users will barter for chunks of files by downloading and uploading them in a tit-for-tat fashion, such that peers with a high upload rate will typically also have a high download rate. It is estimated that tens of millions of users use BitTorrent every day [**?**].

Download Protocol

For a user to download data from BitTorrent they would:

1. Find the corresponding .torrent file that contains metadata about the torrent.
2. The user will find peers, using a tracker identified in the .torrent, that are also interested in that content and will establish connections with them.
3. The user will download blocks[1], from peers, based upon the following priority:
   (a) **Strict Priority** Data is split into pieces and sub-pieces with the aim that once a given sub-piece is requested then all of the other sub-pieces in the same piece are requested.
   (b) **Rarest First** Aims to download the piece that the fewest peers have to increase supply.
   (c) **Random First Piece** When a peer has no pieces, it will try to get one as soon as possible to be able to contribute.
4. The node will continuously upload blocks it has while active.

Availability

One of the most significant issues facing BitTorrent is the availability of torrents, where *'38% of torrents become unavailable in the first month'* [**?**] and that *'the majority of users disconnect from the network within a few hours after the download has finished'* [**?**]. This paper [**?**] looks at how the use of multiple trackers for the same content and DHTs can be used to boost availability.

[1] nodes may reject downloads without the user providing data themselves in a tit-for-tat fashion

2.2 Ethereum

Ethereum is a Turing-complete, distributed, transaction-based blockchain that allows the deployment of decentralized applications through the use of smart contracts. Ether is the currency used on Ethereum and can be traded between accounts and is used to execute smart contract code on the network.

Smart Contracts

A smart contract is an executable piece of code, written in Solidity, that will automatically execute on every node in the Ethereum network when certain conditions are met. Smart contracts are enforced by the blockchain network and remove the need for intermediaries and reduce the potential of contractual disputes.

Gas is used to measure the computational effort of running a smart contract and must be paid, in Ether, before being processed and added to the blockchain. This helps prevent DoS attacks and provides economic incentives for users to behave in a way that benefits the whole network.

Example Use Cases

Some examples of applications that can be deployed to the Ethereum network are:

- Financial applications, such as decentralised exchanges and payment systems,
- supply chain management and tracking,
- voting and governance systems,
- unique digital asset systems, and
- data storage and sharing platforms.

Chapter 3

Literature Review

3.1 Blockchain-Based Cloud Storage

Blockchain technology can be leveraged for distributed cloud storage to provide both public and private storage. In table 3.1, I detail some examples of how blockchain has been used to create cloud storage platforms:

One gap found when researching these solutions was that few offered file versioning that would allow a user to view previous versions of uploaded data. File versioning is a particularly important to this project as users will likely all have varying versions of the same software.

Paper	**Description of Solution**
Blockchain Based Data Integrity Verification in P2P Cloud Storage [**?**]	This paper uses Merkle trees to help verify the integrity of data within a P2P blockchain cloud storage network. It also looks at how different structures of Merkle trees effect the performance of the system.
Deduplication with Blockchain for Secure Cloud Storage [**?**]	This paper describes a deduplication scheme that uses the blockchain to record storage information and distribute files to multiple servers. This is implemented as a set of smart contracts.
Block-secure: Blockchain based scheme for secure P2P cloud storage [**?**]	A distributed cloud system in which users divide their own data into encrypted chunks and upload those chunks randomly into the blockchain, P2P network.
Blockchain-Based Medical Records Secure Storage and Medical Service Framework [**?**]	Describes a secure and immutable storage scheme to manage personal medical records as well as a service framework to allow for the sharing of these records.
A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems [**?**]	This solution uses IPFS, Ethereum and ABE technology to provide distributed cloud storage with an access rights management system using secret keys distributed by the data owner.

Blockchain based Proxy Re-Encryption Scheme for Secure IoT Data Sharing [?]	An IoT distributed cloud system for encrypted IoT data that uses a proxy re-encryption scheme that allows the data to only be visible to the owner and any persons present in the smart contract.

Table 3.1: *Examples of blockchain cloud storage systems [?]*

3.2 P2P File Sharing

It is unreasonable to expect every node to have a copy of each game uploaded to the blockchain so data will be fragmented across the network. This project will use ideas from various P2P file-sharing networks to help connect nodes interested in the same content Table 3.2 shows some example p2p file-sharing networks.

The main issues involving these networks are:

1. **Trust** Nodes are typically anonymous and you can never fully trust that what you're downloading isn't malicious, and
2. **Payment** These platform don't allow users to pay for content and are generally large sources of piracy.

System	Description of Solution
IPFS [?]	IPFS is a content-addressable, block storage system which forms a Merkle DAG, a data structure that allows the construction of versioned file systems, blockchains and a Permanent Web.
BitTorrent [?]	BitTorrent is a p2p file-sharing system that has user bartering for chunks of data in a tit-for-tat fashion, which provides incentive for users to contribute to the network. More on BitTorrent can be found in Section 2.1
AFS [?, ?]	The Andrew File System was a prototype distributed system by IBM and Carnegie-Mellon University in the 1980s that allowed users to access their files from any computer in the network.
Napster [?]	Napster uses a cluster of centralized servers to maintain an index of every file currently available and which peers have access to it. A node will maintain a connection to this central server and will query it to find files; the server responds with a list of peers and their bandwidth and the node will form a connection with one or many of them and download the data.
Gnutella [?]	Gnutella nodes form an overlay network by sending *ping-pong* messages. When a node sends a *ping* message to their peers, each of them replies with a *pong* message and the *ping* is forwarded to their peers. To download a file, a node will flood a message to its neighbors, who will check if they have and return a message saying so; regardless, the node will continue to flood their request till they find a suitable node to download off of.

Table 3.2: *Various global distributed file systems.*

Chapter 4

Design

4.1 Stakeholders & Requirements

Stakeholders

Game Developers ***primary***

This group will use the application to release their games and its updates to their users, who they will reward for helping to distribute it.

Players ***primary***

This group will use this application to downloaded and update their games off of. They may also contribute to the distribution of the games to other players for an incentive provided by the developers.

Other Platforms ***secondary***

This group consists of platforms like Steam or Epic Games, which serve as the main competitor to this application. It is likely that as more developers choose this application, this group will see a loss in revenue.

Requirements

Tables 4.1 and 4.2 show the functional and non-functional requirements of this project organized using MoSCoW prioritisation.

Functional Requirements

ID	Description
	Must
F_M1	Store game metadata on a blockchain
F_M2	A node must data as constant-sized shards from its peers
F_M3	A node must be able to discover peers who have their desired game installed
F_M4	Games must be updatable through the blockchain
F_M5	A node must be able to upload games

F_M6	A node must be able to download games in their entirety from nodes in the network.
F_M7	A node must be able to verify the integrity of each block it downloads
F_M8	The application should run on the Ethereum network
F_M9	Users must be able to purchase games from developers over the network
F_M10	Users must be able to prove they have purchased a game
	Should
F_S1	Seeders should have a way to prove how much data they have seeded
F_S2	Seeders will only upload content to users who have a valid proof of purchase
F_S3	Allow for the distribution of DLC for games
	Could
F_C1	Allow users to request specific game versions
F_C2	Provide a simple GUI for interacting with the blockchain

Table 4.1: *These requirements define the functions of the application in terms of a behavioural specification*

Non-Functional Requirements

ID	Description
	Must
NF_M1	The application is decentralized and cannot be controlled by any one party
NF_M2	Any user must be able to join and contribute to the network
NF_M3	Game uploaders should be publicly identifiable
NF_M4	Metadata required to download the game should be immutable
	Should
NF_S1	This application must be scalable, such that many users can upload and download the same game at the same time.
NF_S2	Only the original uploader can upload an update to their game
NF_S3	Only the original uploader can upload a DLC for their game
	Could
NF_C1	The application could have an intuitive GUI

Table 4.2: *Requirements that specify the criteria used to judge the operation of this application*

4.2 Design Considerations

Architecture

Figure 4.1 shows the architecture of this application:

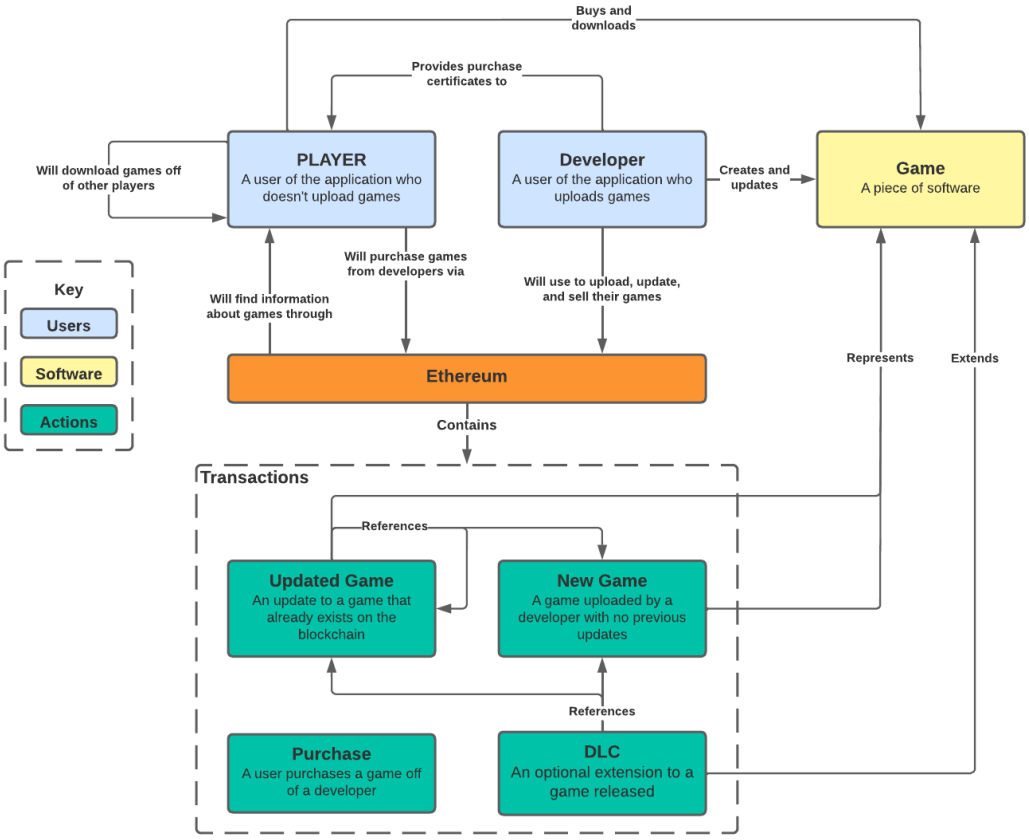


Figure 4.1: *Architecture of the application*

Type of Blockchain

To satisfy **NF_M1** and **NF_M2**, we will need to use a public blockchain, which will benefit our project by:

- being accessible to a larger user-base, which should boost availability and scalability (**NF_S1**),
- reducing the risk of censorship (**NF_M1**), and
- providing greater data integrity (**NF_M4**)

Ethereum is a public blockchain that allows developers to publish their own distributed applications to it. It comes with an extensive development toolchain so is an obvious choice for this project (**F_M8**).

Sequence Diagram

Figure 4.2, shows the main interactions between actors in the application.

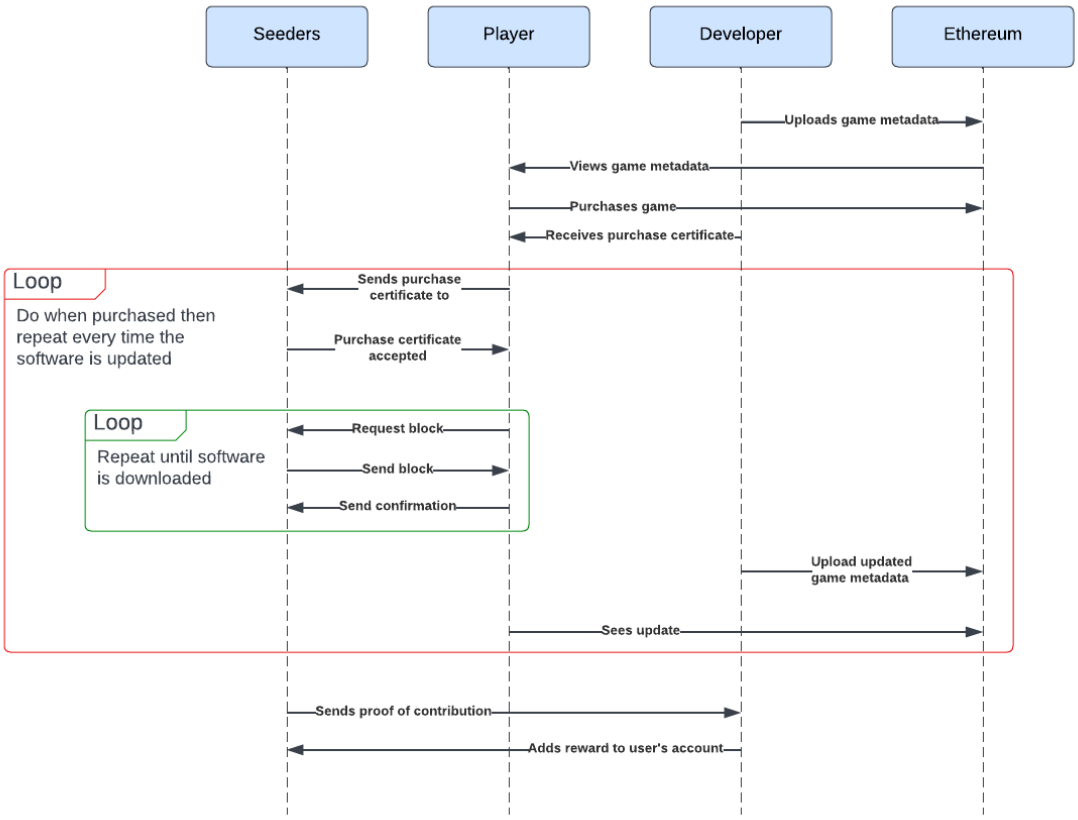


Figure 4.2: *A sequence diagram showing some of the main interactions within this application*

Uploading Content

For developers to upload their game (**F_M5**), they must provide a digital certificate to prove their identity (**NF_M3**) as well as the required metadata (**F_M1**) for identifying, downloading and verifying their game. The developer is then expected to allow users to purchase the game off them and seed the game to at least an initial group of users.

Purchasing Content

Users will purchase content from sellers using Ether (**F_M9**) and will be provided with a proof of purchase (**F_M10**) that is encrypted with the developer's private key. By using public key infrastructure, it gives proof of purchases authenticity that allows them to be validated by any node in the network.

The proof of purchase will include: the root hash of the title purchased, the seller's address, the buyer's address, a timestamp, and a unique seeder token to be used for proving distribution.

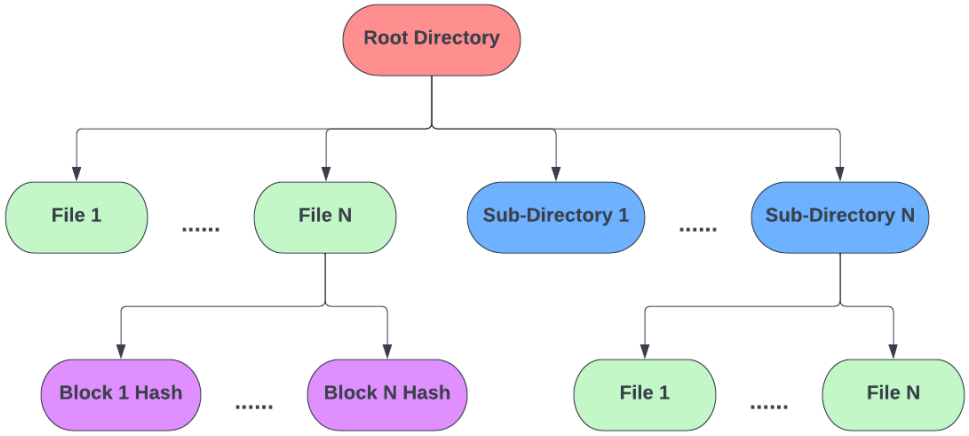


Figure 4.3: *Hash data about shards will be stored in a tree that mimics the folder structure of the data*

Downloading Content

Games will be content addressable, using their root hash stored on the blockchain. This will be used to help connect nodes who are interested in the same content (**F_M3**). Once a user connects to a node it will:

1. Send their proof of purchase (**F_S2**).
2. request individual shards from the node using the shard's hash (**F_M2**),
3. use the metadata from the blockchain to verify the shard's contents (**F_M7**),
4. send a confirmation message that proves the successful transfer of a block (**F_S1**),
5. and repeat this until the entirety of the game is installed (**F_M6**).

As the blockchain will only be used to store metadata about games uploaded to the network, not every node will have each game installed. Instead, this project will take ideas from networks like BitTorrent, where nodes will seek out peers who have the game installed using a tracker hosted by the uploader.

Updating Content

To satisfy **F_M4**, developers will perform the same steps as in **Uploading Content** but will also include the hash of a previous block that contains the older version of the game. This will include the restriction that only the original uploader can upload an update for their game (**NF_S2**).

It is likely that many shards will persist between versions so a node will only ever download the new or changed data. To satisfy **F_C1**, a node may optionally keep older shards that have been removed or changed.

Downloadable Content

Developers will typically offer paid or free DLC (**F_S3**) for their games that users will purchase separately. Each DLC will need:

1. **Dependency** A reference to the oldest version of the game they apply to, and
2. **Previous Version** A pointer to the previous version of the DLC.

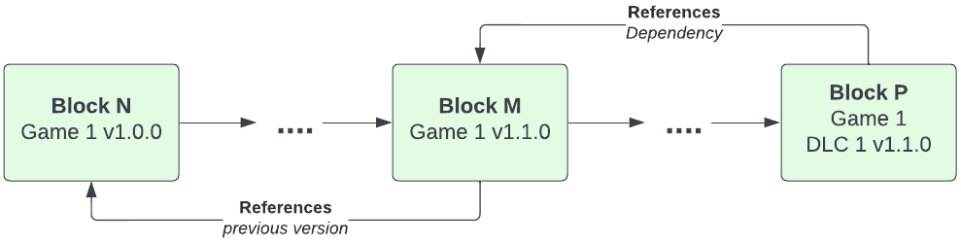


Figure 4.4: *How blocks relate to each other. An update will reference the previous version whilst a DLC will reference, which piece of software and version it is dependent on.*

Proving Contribution

A purchase certificate will include a confidential *seeder token*. When a user successful downloads a shard of data they will send a confirmation message, including their seeder token, that is encrypted with the game developer's public key. A user will prove their contribution (**F_S1**) by sending a collection of these messages to the developer, who can decrypt and validate the tokens.

4.3 Limitations

This project will not attempt to mimic any of the social features (friends, achievements, message boards, etc.) provided by platforms like Steam.

Section 2.1 highlights the issue of availability in p2p sharing networks and this platform will likely suffer from similar issues. These can be mitigated by having an active community or good incentives to help distribute the game.

Chapter 5

Project Management

5.1 Risk Assessment

Risk	Loss	Prob	Risk	Mitigation
Difficulty with blockchain development	2	3	6	I will seek advice from my supervisor about how to tackle certain problems and if necessary, what aspects of my project I should change.
Personal illness	3	2	6	Depending on the amount of lost time, I may ignore some of the SHOULD or COULD requirements.
Laptop damaged or lost	3	1	5	All work is stored using version control and periodic backups will be made and stored locally and in cloud storage. I have other devices that could be used to continue development.
The application is not finished	1	3	3	Using agile development will ensure that I will at least have a minimal working application. If I feel that I am running out of time, I will focus on testing and finalising the report.

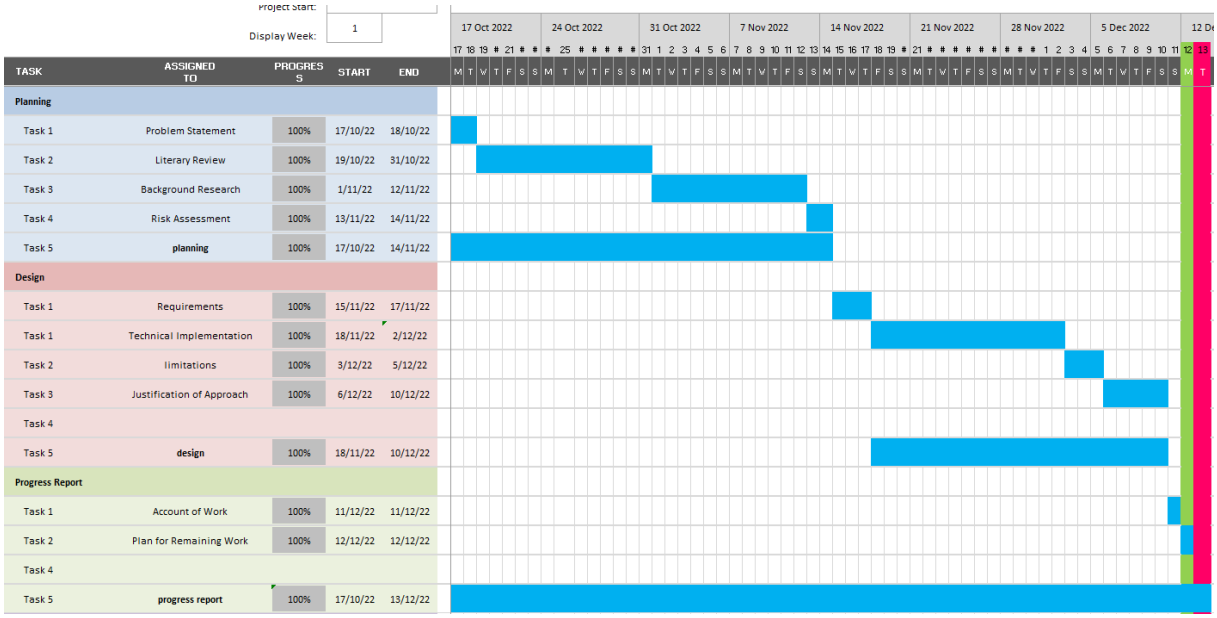
Table 5.1: *The risk assessment of this project*

5.2 Work to Date

My work has primarily been on research, looking at how blockchain has been used to build cloud storage systems as well as how various peer-to-peer function and perform. I have proposed a design for the application to be built on the EVM.

5.3 Plan of Future Work

Below is a high level plan for the work I have remaining.



TASK	ASSIGNED TO	PROGRESS	START	END
Planning				
Task 1	Problem Statement	100%	17/10/22	18/10/22
Task 2	Literary Review	100%	19/10/22	31/10/22
Task 3	Background Research	100%	1/11/22	12/11/22
Task 4	Risk Assessment	100%	13/11/22	14/11/22
Task 5	planning	100%	17/10/22	14/11/22
Design				
Task 1	Requirements	100%	15/11/22	17/11/22
Task 1	Technical Implementation	100%	18/11/22	2/12/22
Task 2	limitations	100%	3/12/22	5/12/22
Task 3	Justification of Approach	100%	6/12/22	10/12/22
Task 4				
Task 5	design	100%	18/11/22	10/12/22
Progress Report				
Task 1	Account of Work	100%	11/12/22	11/12/22
Task 2	Plan for Remaining Work	100%	12/12/22	12/12/22
Task 4				
Task 5	progress report	100%	17/10/22	13/12/22

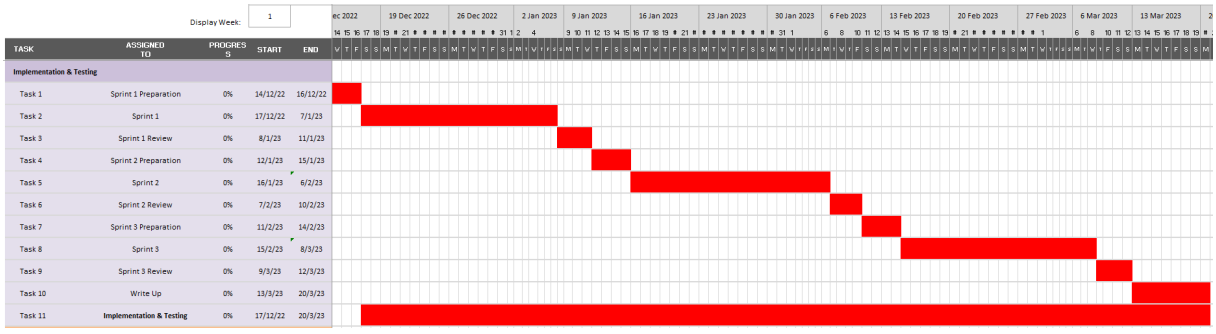
Figure 5.1: *A Gantt chart for my work up until the progress report.*

Implementation & Testing

In this phase I will use the agile development methodology to build my application. My sprints will all be structured into three phases:

1. **Preparation** Deciding on the set of requirements to complete and making any initial design decisions and diagrams,
2. **Implementation** Using test-driven development, I will work on requirements based on their prioritization, and
3. **Review** I will discuss the completed work in that sprint including design choices, what was completed, and any issues.

By using a preparation and review phase with each sprint, I can make detailed notes on my implementation as I go so when I come to write the full report I have a strong starting point.



TASK	ASSIGNED TO	PROGRESS	START	END
Implementation & Testing				
Task 1	Sprint 1 Preparation	0%	14/12/22	16/12/22
Task 2	Sprint 1	0%	17/12/22	7/1/23
Task 3	Sprint 1 Review	0%	8/1/23	11/1/23
Task 4	Sprint 2 Preparation	0%	12/1/23	15/1/23
Task 5	Sprint 2	0%	16/1/23	6/2/23
Task 6	Sprint 2 Review	0%	7/2/23	10/2/23
Task 7	Sprint 3 Preparation	0%	11/2/23	14/2/23
Task 8	Sprint 3	0%	15/2/23	8/3/23
Task 9	Sprint 3 Review	0%	9/3/23	12/3/23
Task 10	Write Up	0%	13/3/23	20/3/23
Task 11	Implementation & Testing	0%	17/12/22	20/3/23

Figure 5.2: *The plan for my three sprints*

Testing Strategy and Results

This phase will consists of several sections:

1. **Testing Strategy** The approach I used for writing tests throughout the implementation phase and how these were used to evaluate the completion of the requirements set out in Section 4.1.
2. **Test Results** A report on the results of my automated and manual testing.

Evaluation

This phase will have me evaluating how successful my project was, as a whole, by focusing on several key areas:

1. **Project Organisation** How successfully did I structure my time in this project?
2. **Outcome of the Application** How successful was my application in regards to a solution to the problem set out in Section 1.1 and in terms of the requirements set out in Section 4.1?
3. **Limitations and Future Improvements** What were the limitations of my project and what would I change about it if I had more time or were to start again?

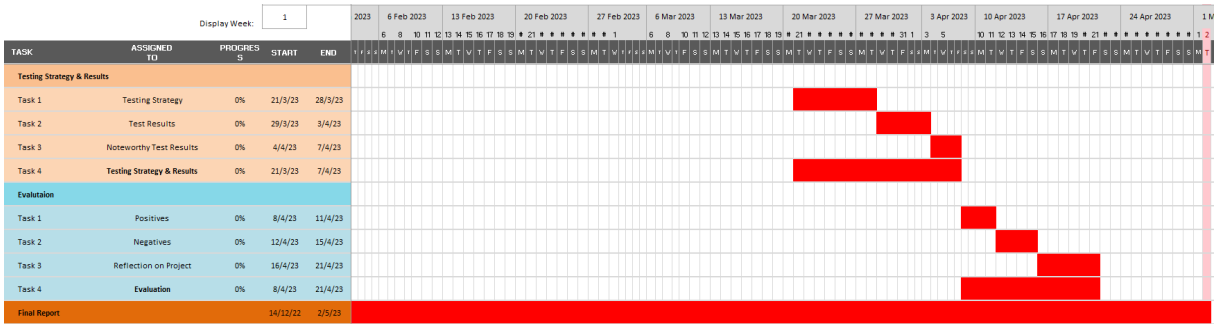


Figure 5.3: *The plan for my testing and evaluation phases*

Final Report

I will be writing the first draft of my report throughout the project, aiming to finalise sections at the end of each phase of this project. Any leftover time at the end will be focused on finishing the report.