HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

BoomBlock - Incorporating NFTs into Classical Bomb Game

NGUYEN PHU TRUONG
truong.np184319@sis.hust.edu.vn

Major: Information Technology

Supervisor: PhD Tran Vinh Duc _____

Signature

Department: Computer Science

School: Information and Communications Technology

HANOI, 07/2023

ACKNOWLEDGMENTS

In the first part of this graduation thesis, I want to express my sincere gratitude and appreciation to all those who have supported and guided me throughout this journey of completing my graduation thesis.

I extend my profound thanks to my mentor, **Dr. Tran Vinh Duc**, for his unwavering support, invaluable guidance, and deep knowledge of my research field. His encouragement and insightful feedback have played a vital role in shaping this work and motivating me to excel beyond my limits.

I cannot overlook my mother's presence in this expression of gratitude. She has been there through all the ups and downs of my life, teaching and guiding me from the very beginning. Her trust, encouragement, and unwavering support have been a tremendous source of motivation for me to persist through challenges and complete this academic endeavor. I sincerely thank her for her sacrifices and everlasting love and care.

Next, I cannot express enough gratitude to my beloved partner, **Nguyen Kim Chi**, for her unwavering emotional support and enthusiastic interest in my endeavors. She dedicated her time to studying with me, listened to my ideas and opinions, stood by me during difficult times, and shared my joys in progress. Her continuous support and encouragement have been a significant driving force that helped me overcome difficulties, face challenges, and complete this thesis.

Furthermore, I want to express my thanks to my friends, **Chu Hoang Lan, Chu Manh Hai, Do Ton Nhat Minh, Nguyen Thi Thu Thao, Nguyen Kim Long, Nguyen Tuan Dung**, who have not only supported and assisted me in this thesis but also accompanied me throughout my student journey, contributing to the person I am today.

Finally, I thank my family, friends, and teachers for their support and encouragement throughout my academic journey. Their motivation, genuine hearts, and motivating words have kept me strong and confident throughout completing this work. Thank you, everyone!

ABSTRACT

In this graduation thesis, an NFT-based game has been developed, drawing inspiration from the classic Bomb IT game. The primary focus of this project is to address the growing interest in NFTs and blockchain technology within the gaming industry, recognizing the untapped potential of integrating NFTs into popular games.

A decentralized gaming platform was created as the foundation for this innovative approach, allowing players to have ownership, traceability, and utilization of NFTs as in-game assets. Smart contracts were implemented on the blockchain to ensure transparency and security in NFT ownership and transactions. Consequently, players can now collect and customize unique in-game items represented as NFTs, which adds a heightened sense of authenticity and exclusivity to their virtual possessions.

The most significant contribution of this thesis lies in the successful incorporation of NFTs into a familiar gaming experience, significantly enhancing overall gameplay and fostering the establishment of a groundbreaking gaming ecosystem. This newly developed game serves as a compelling demonstration of the potential impact of NFTs on the gaming industry, as it introduces genuine ownership and tangible value to in-game assets.

Various challenges emerged throughout the development process, particularly concerning optimizing blockchain performance and user experience. However, these challenges were effectively overcome by leveraging existing technologies and implementing efficient solutions, resulting in a fully functional and captivating NFT-based gaming experience.

In conclusion, this graduation thesis introduces an innovative and pioneering approach to seamlessly integrating NFTs into the gaming landscape, offering players a new level of engagement and interaction with in-game assets. The remarkable success of this project underlines the transformative potential of NFTs in shaping the gaming industry's future, thus paving the way for further research and development in this promising domain.

Student's execution
(Signature and full name)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abriviation	Full Expression
API	Application Programming Interface
CPU	Central Processing Unit
CRUD	Create-Read-Update-Delete
DFD	Data-flow diagram
ERD	Entity Relationship Diagram
ETH	Ethereum
GUI	Graphical User Interface
HTML	HyperText Markup Language
ID	Identification
IDE	Integrated Development Environment
JVM	Java Virtual Machine
NFT	Non-fungible Token
RAM	Random Access Memory
SQL	Structured Query Language
UML	Unified Modeling Language
zkEVM	Zero-knowledge Ethereum Virtual Machine
Zk-Rollup	Zero-knowledge Rollup

CHAPTER 1. INTRODUCTION

1.1 Motivation

The gaming industry has experienced remarkable growth and diversification worldwide. In 2022, the industry generated more than $180 billion in revenue, a figure that's still rising. Electronic games not only provide entertainment but also offer business opportunities and employment for millions of people worldwide.

However, there is a need to maximize gaming's potential to provide more community benefits. Traditional online gaming models still face issues regarding player privacy, user ownership rights, and limitations on control and customization of content. These challenges can lead to copyright disputes and unauthorized trading of game assets outside the publisher's control, which impacts player rights.

To address these issues, NFT games have emerged as a new trend in the gaming industry. NFTs enable game assets to be unique, non-replaceable tokens recorded on the blockchain, offering players transparency and ownership and allowing players to buy, sell, and use in-game assets freely.

For the NFT business model to succeed, developers must implement a reasonable business strategy that ensures fairness and sustainability for players and developers. When executed correctly, the model can yield substantial benefits for both parties. Players will feel treated and motivated to engage and invest in the game, while developers can generate reasonable profits and continue to develop quality products.

Furthermore, the NFT business model can extend beyond gaming to other fields. For instance, in the digital art industry, NFTs have created opportunities for digital artists to sell their works directly to fans without intermediaries, reducing their dependence on publishing companies and enabling them to retain control and value over their creations.

In conclusion, leveraging blockchain technology and cryptocurrencies through NFT games can address various challenges in the gaming industry, opening up new opportunities and benefits for the gaming community and other sectors and paving the way for a sustainable future.

1.2 Objectives and scope of the graduation thesis

Currently, the market offers a diverse range of unique NFT games. Here are some examples of prominent NFT games:

- CryptoKitties: Considered a pioneering NFT game, CryptoKitties allows players to create, breed, and trade virtual cats with distinct traits. Each cat is encoded as an NFT, granting players full ownership and freedom to trade.
- Axie Infinity: Falling under the genre of battling arenas, Axie Infinity enables players to raise and control creatures known as Axies. Thanks to NFT technology, players can fully own and trade their Axies, establishing a thriving in-game economy.
- Decentraland: Built on the blockchain, Decentraland is a virtual world where players can own and build assets, lands, and buildings according to their desires. All land parcels and assets within Decentraland are encoded as NFTs, providing true ownership for players.

These mentioned NFT games demonstrate a positive and diverse development within the gaming industry. User demand for NFT gaming products is increasing steadily. Players value the fairness and flexibility of owning in-game assets as NFTs, as the ability to freely trade and transparency adds uniqueness to their gaming experiences. Additionally, this ownership can lead to income generation through asset transactions.

However, when comparing and evaluating the current state of NFT games, there are still some notable limitations to consider. One crucial limitation is the issue of transaction speed and cost. Utilizing blockchain services for transaction confirmation and asset storage can lead to slow and costly transactions, especially during high transaction volumes.

Moreover, the gameplay experience is another area for improvement in many current games. Some games focus too much on the NFT aspect and overlook the actual value that gaming should offer: entertainment, passion, and an escape from life's stresses.

Furthermore, the lack of regulations and oversight in the NFT gaming field can give rise to issues such as scams, copyright infringements, or violations of user ownership rights. Addressing these concerns is essential to protect players' rights and assets within the expanding NFT gaming landscape.

Boomblock is an innovative game project that aims to overcome many existing limitations in the NFT gaming market. Inspired by the popular game Bomb It, Boomblock promises to be a blockbuster. It belongs to the competitive battling arena genre, where players can control their characters and engage in intense battles to emerge as the last one standing.

A standout feature of Boomblock lies in the unique ownership and trading of in-game characters. Utilizing NFT technology, players can fully own their characters. They can trade them with other gaming community members, which contributes to increased flexibility and added value in the gaming experience, as players can own and manage their assets.

Furthermore, Boomblock emphasizes optimizing gameplay time, with each match lasting a maximum of 5 minutes, which makes the game suitable for players of all types, allowing them to participate in quick matches without investing too much time. This convenience enhances the player experience, enabling them to enjoy the game flexibly and comfortably at any time.

Another vital aspect of Boomblock is adopting Zk Rollup technology, a notable solution for scalability on the Ethereum network using Zero-knowledge proof. With this technology, the game hastens transaction speeds, significantly decreases expenses, and dramatically enhances the gaming experience, enabling players to relish a smoother and more economical experience.

In summary, the Boomblock game project promises an engaging and innovative gaming experience, addressing many of the current limitations in the NFT gaming market. With its unique ownership and trading of characters, flexible gameplay time, and utilization of Zk Rollup technology to enhance transaction efficiency, Boomblock is set to become an attractive choice for gaming enthusiasts seeking a competitive battling experience and wishing to explore the potential of NFT technology.

1.3 Tentative solution

Taking inspiration from popular titles in the traditional gaming market, Boomblock aims to tackle the difficulty of producing a genuinely top-notch game for players. Boomblock is designed following the client-server model, with the game's client built using Cocos-2dx JavaScript. There are two servers implemented in the project. The main server is built with Java and handles all client requests while synchronizing the clients with each other. In order to interact with the smart contract, the main server calls another server deployed with Node.js and utilizes the web3 library to create transactions. The smart contract is written in Solidity and deployed on the Ethereum network.

This thesis's main contributions lie in the client-server architecture's design. Additionally, it includes implementing game synchronization mechanisms and mechanisms to determine asynchrony between clients and between the client and server. These crucial factors optimize the gaming experience and ensure accuracy during

information exchange between the components in the system.

The thesis also proposes using zk EVM (Zero-knowledge Ethereum Virtual Machine) to enhance processing speed and reduce transaction costs. Zk EVM provides data security and allows data verification without revealing sensitive details, resulting in high efficiency in processing transactions and improving the gaming experience for users.

To sum up, the Boomblock thesis aims to create an immersive game. It introduces new technological solutions to improve performance and gameplay in client-server and Ethereum network settings.

1.4 Thesis organization

The remaining part of this graduation thesis report is structured as follows.

Chapter 2 presents comprehensive surveys and an overview of the game's design under development, covering game rules, use case designs, and functional and non-functional requirements.

Chapter 3 introduces the technologies utilized in the thesis, including the client-server architecture, zkRollup solution, tools, and open-source libraries.

Chapter 4 showcases the results after implementing the game designs on the Windows platform. This chapter also includes illustrative images of the game client and performance evaluation results for the server.

Moving on to Chapter 5 delves into notable solutions and contributions of the thesis, including the synchronization mechanisms used in the game, techniques for detecting asynchrony between clients and between the client-server, and solutions to enhance processing speed while reducing costs for transactions on the ETH network, which the project utilizes.

Chapter 6 offers conclusive insights from the thesis work and outlines future development directions.

Chapter 7 comprises a list of the reference documents used by the writer during the thesis implementation.

Finally, an appendix section has been included for those who wish to delve deeper into the practical applications of the technologies discussed in Chapter 5 within the context of this thesis.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

2.1 Game Design

2.1.1 Overview of game and gameplay rules

Boomblock is a real-time strategy and combat game with a top-down perspective. In the game, four players will battle against each other by moving, attacking, and trying to eliminate opponents while striving to survive until the end of the match. Players must have a character and a certain amount of the game's tokens to participate in a match. At the end of the match, the surviving players will share the total amount of tokens deposited by all four players at the beginning of the match.

During gameplay, players can view their information and other players' health, the number of bombs they can place, the bomb blast range, and additional speed. Destructible obstacles will be randomly placed on the map, and players can destroy them with their current weapons to have a chance to collect power-up items or switch to different weapons. The match will last 5 minutes or end earlier if only one player remains alive.

2.1.2 Control mechanism

The movement mechanism in the game will be entirely keyboard-based when playing on a computer and will rely on a joystick when playing on a smartphone. Players will use the arrow keys or the joystick to move the character in all directions: up, down, left, right, and diagonally. Players will use the spacebar on the computer to attack or tap the smartphone screen's attack button. When the character is required to attack, the visual effects and skill damage will be based on the character's currently equipped weapon.

2.2 Functional Overview

2.2.1 General use case diagram

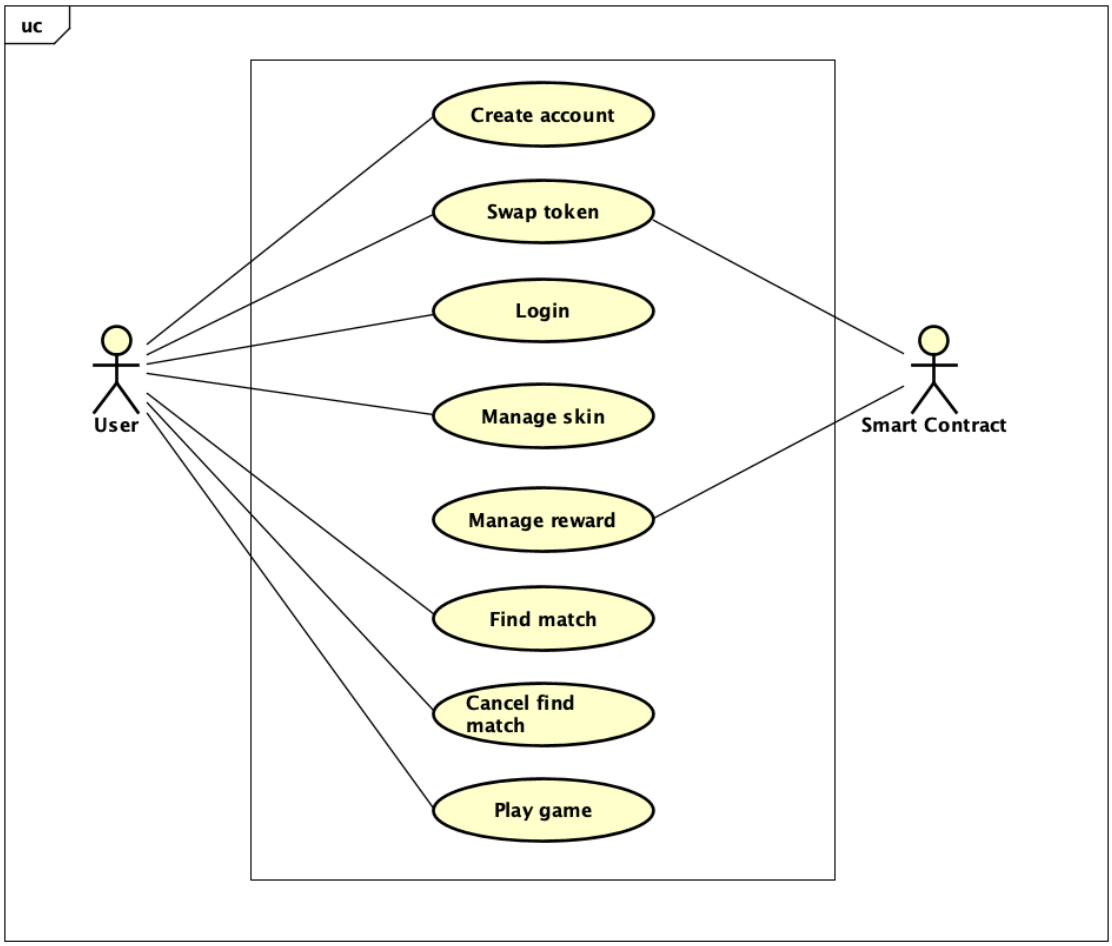

Figure 2.1: Overview usecase diagram

Figure 2.1 depicts the overall use cases of the system. Players can swap tokens for in-game use, which can be used for rewards and purchasing in-game characters. Players are required to log in to access the game. They can purchase characters directly in the game and also have the option to sell them on NFT (Non-Fungible Token) marketplaces like OpenSea. To participate in a match, players need to find a match to pair up with other opponents and can also cancel the pairing if they do not wish to proceed. Once the pairing is complete, players and their opponents will engage in a battle, where they must control their characters to achieve victory.

The smart contract will manage the in-game tokens, enabling players to swap tokens within the game and handling transactions for buying and selling skins, as well as game entry fees and rewards distribution after identifying the winning player.

2.2.2 Use case diagram for ”Play game”

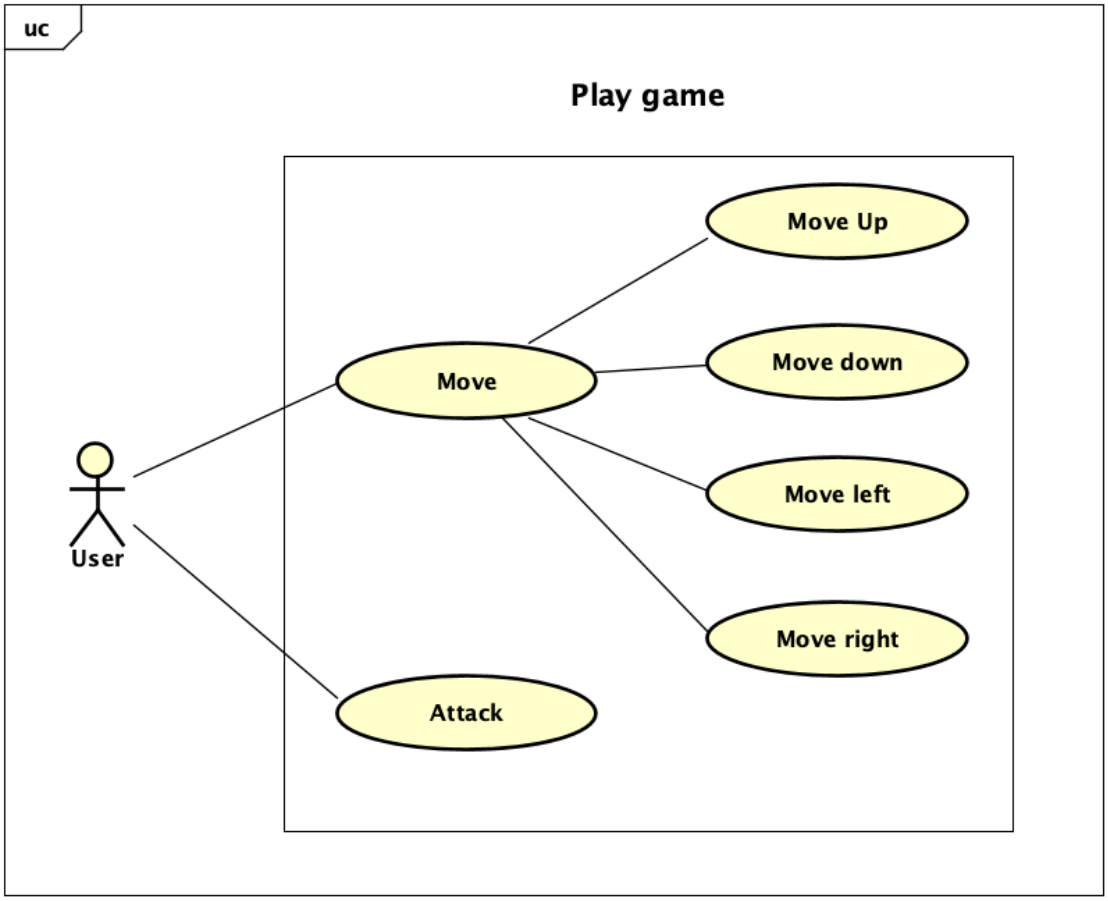

Figure 2.2: Use case diagram for ”Play game”

During the match, the player can move in four directions: up, down, left, right, or diagonally by simultaneously navigating two directions. While moving, the player can destroy destructible obstacles to obtain power-up items or switch to different weapons. When the player wants to attack, the system will use the currently equipped weapon to execute the player’s attack move.

2.2.3 Use case diagram for ”Manage skin”

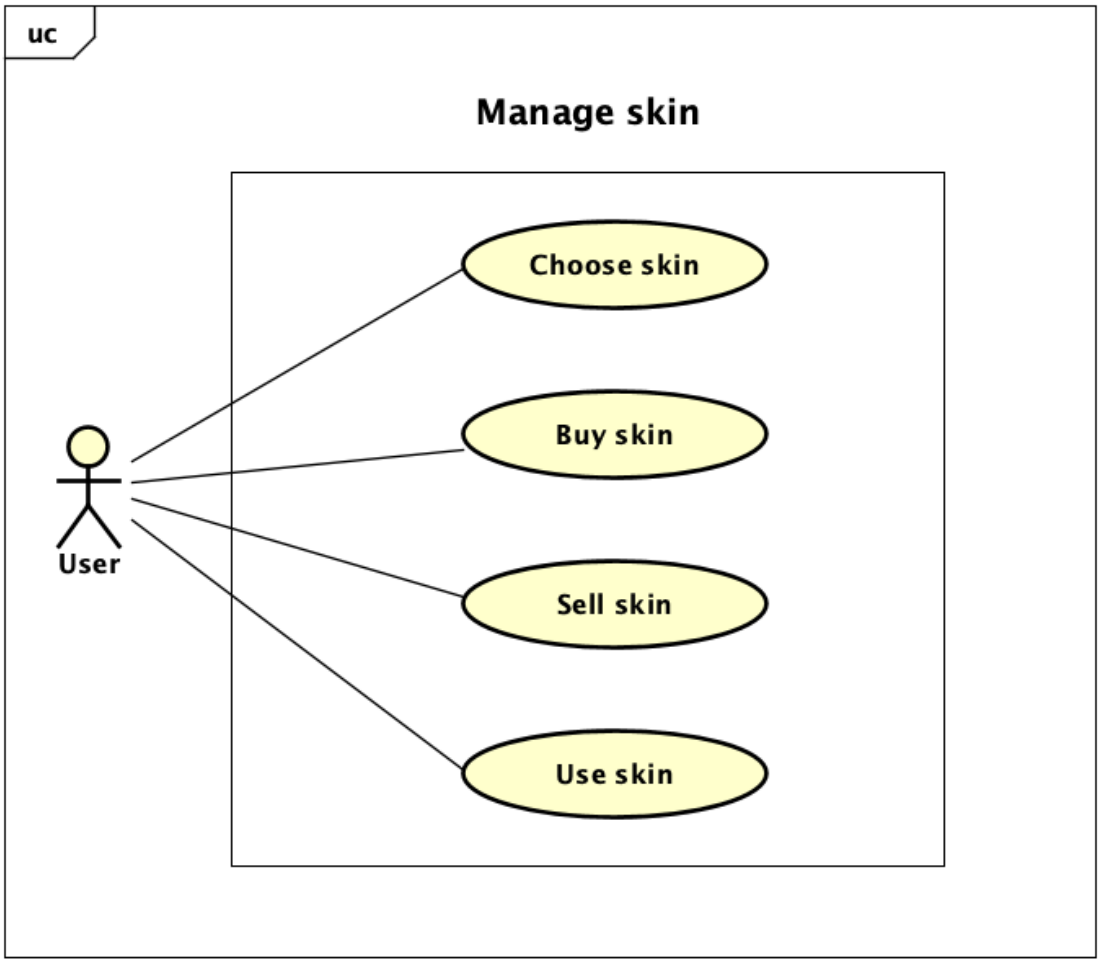

Figure 2.3: Use case diagram for ”Manage skin”

Players can select character skins from the lobby to purchase, sell, and use in-game. Players who buy a character's skin will receive an NFT (Non-Fungible Token) that represents the skin in their wallet. If the user decides to sell the skin, they can list it on various NFT marketplaces like Opensea to put it up for sale. NFTs are unique digital assets that can be bought, sold, and traded on blockchain-based platforms, allowing players ownership and control over their in-game items, including character skins.

2.2.4 Use case diagram for ”Swap token”

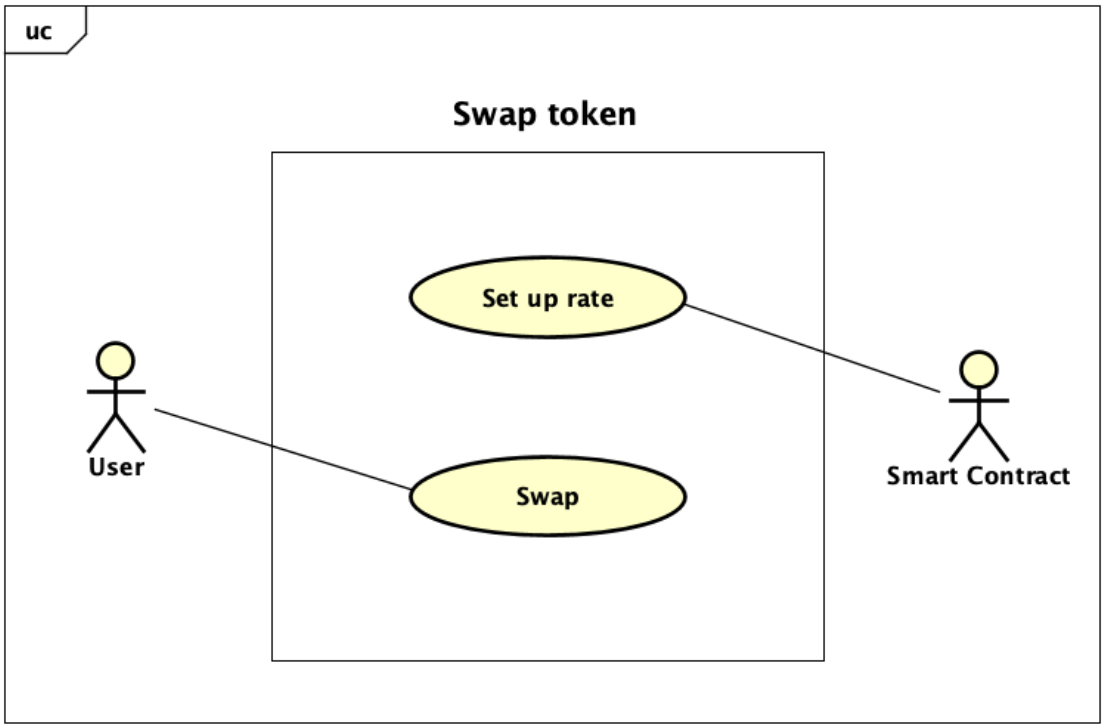

Figure 2.4: Use case diagram for ”Swap token”

To play the game, players must have a minimum amount of tokens to purchase character skins and pay the entry fee to join a match. The token has been pre-configured with an exchange rate, so when swapping a certain amount of native coins, it will be converted to the corresponding amount of tokens based on the system’s default rate.

2.2.5 Use case diagram for ”Manage reward”

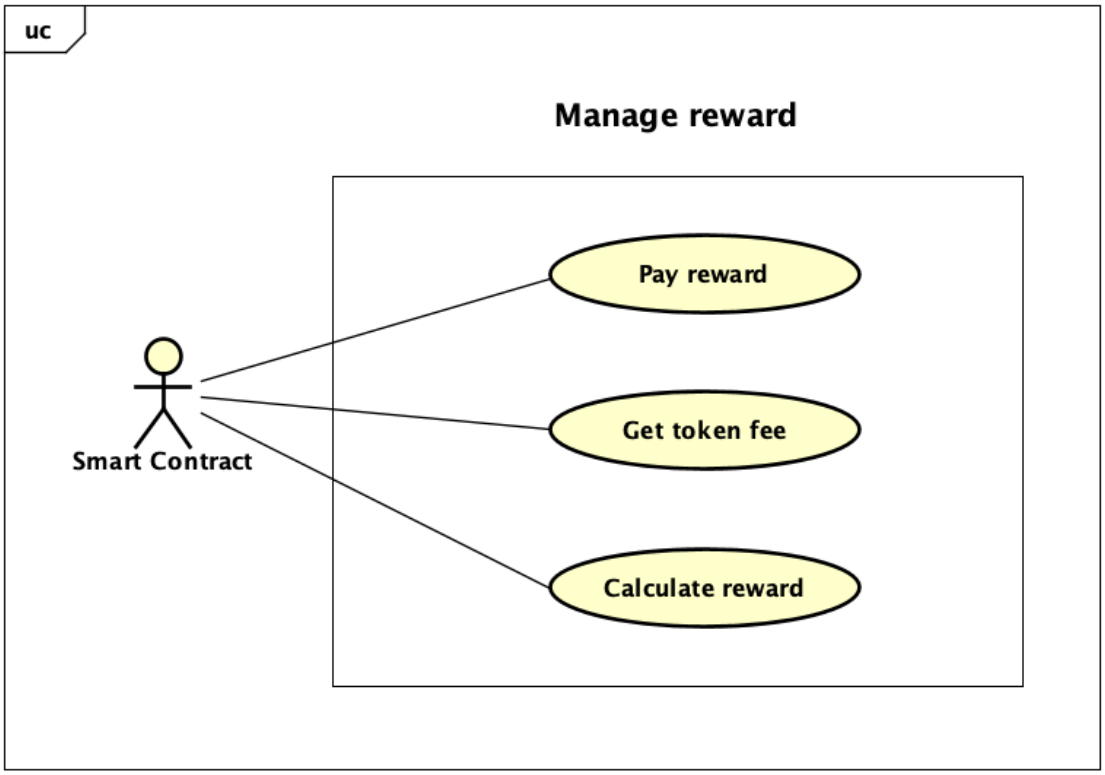

Figure 2.5: Use case diagram for ”Manage reward”

The smart contract will deduct a certain amount of tokens as an entry fee from the user’s wallet when they want to join a match. After the match ends, the smart contract will receive the match result from the server to determine the winning users. It will then calculate the reward amount for the winners and distribute the corresponding rewards to their wallets.

2.2.6 Business process

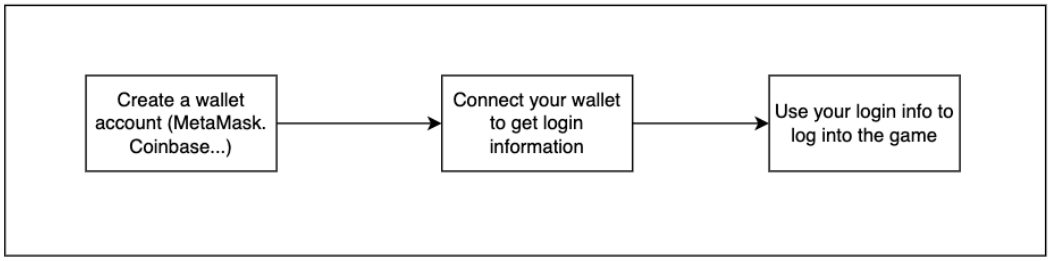

Figure 2.6: Login process

In order to log in to the game, players first need to register a wallet account to store their digital currency. They can utilize popular wallets such as MetaMask or Coinbase. Afterward, they must connect this wallet to the game to receive a login ID. From there, players will use this ID to access the game.

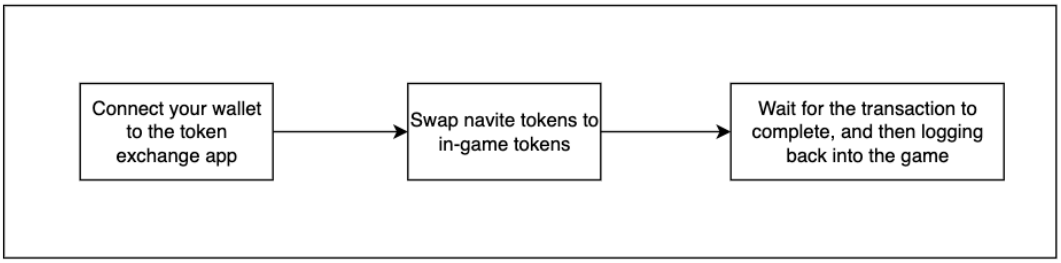

Figure 2.7: Swap token process

To play the game, players need to possess a specific in-game token. Usually, players are required to connect their wallets to token exchange platforms to swap their native tokens for the in-game token. However, due to project constraints, this swapping process will occur directly without involving an exchange platform, and the developers will provide a fixed rate. Once the in-game token has been successfully obtained through the swap, players can log back into the game and engage in various in-game transactions.

2.3 Functional description

2.3.1 Use case description of ”Move up”

<table>
<tr><td>Use case name</td><td>Move up</td></tr>
<tr><td>Actor</td><td>User</td></tr>
<tr><td>Description</td><td>Enable character to move their characters upward</td></tr>
<tr><td>Precondition</td><td>Participated in the match</td></tr>
<tr><td>Main flow of event
(success)</td><td>
<table>
<tr><th>#</th><th>Doer</th><th>Action</th></tr>
<tr><td>1.</td><td>User</td><td>Press the W key or adjust the joystick upwards</td></tr>
<tr><td>2.</td><td>System</td><td>Check if the next position is reachable? Are there any obstacles?</td></tr>
<tr><td>3.</td><td>System</td><td>Move the character upwards</td></tr>
</table>
</td></tr>
<tr><td>Alternative flow of event</td><td>
<table>
<tr><th>#</th><th>Doer</th><th>Action</th></tr>
<tr><td>3a</td><td>System</td><td>If there is an obstacle, the character will remain stationary at its initial position.</td></tr>
</table>
</td></tr>
<tr><td>Post condition</td><td>No</td></tr>
</table>

Table 2.1: Use case ”Move up”

2.3.2 Use case description of ”Attack”

<table>
<tr><td>Use case name</td><td>Attack</td></tr>
<tr><td>Actor</td><td>User</td></tr>
<tr><td>Description</td><td>Enable character attacks</td></tr>
<tr><td>Precondition</td><td>Participated in the match</td></tr>
<tr><td>Main flow of event
(success)</td><td>
<table>
<tr><th>#</th><th>Doer</th><th>Action</th></tr>
<tr><td>1</td><td>User</td><td>Press the Space key or click on the 'Attack' button on the screen</td></tr>
<tr><td>2</td><td>System</td><td>Check if it's possible to attack at that moment?</td></tr>
<tr><td>3</td><td>System</td><td>Detect the current weapon for attacking and inflicting damage</td></tr>
</table>
</td></tr>
<tr><td>Alternative flow of event</td><td>
<table>
<tr><th>#</th><th>Doer</th><th>Action</th></tr>
<tr><td>3a</td><td>System</td><td>If unable to attack, the character will not take any action</td></tr>
</table>
</td></tr>
<tr><td>Post condition</td><td>No</td></tr>
</table>

Table 2.2: Use case ”Attack”

2.3.3 Use case description of ”Buy skin”

<table>
<tr><td>Use case name</td><td>Buy skin</td></tr>
<tr><td>Actor</td><td>User</td></tr>
<tr><td>Description</td><td>Users buy character skins to use in battles</td></tr>
<tr><td>Precondition</td><td>Already logged in</td></tr>
<tr><td>Main flow of event
(success)</td><td>
<table>
<tr><th>#</th><th>Doer</th><th>Action</th></tr>
<tr><td>1</td><td>User</td><td>Choose your desire skin</td></tr>
<tr><td>2</td><td>User</td><td>Click on the 'Buy' button on the skin</td></tr>
<tr><td>3</td><td>System</td><td>Check if the user has enough tokens to make the purchase?</td></tr>
<tr><td>4</td><td>System</td><td>Create a transaction, deduct tokens, and create the skin to the user</td></tr>
<tr><td>5</td><td>System</td><td>Update the user’s token balance and the list of owned skins</td></tr>
</table>
</td></tr>
<tr><td>Alternative flow of event</td><td>
<table>
<tr><th>#</th><th>Doer</th><th>Action</th></tr>
<tr><td>3a</td><td>Client</td><td>If there are not enough tokens, notify an error message</td></tr>
</table>
</td></tr>
<tr><td>Post condition</td><td>No</td></tr>
</table>

Table 2.3: Use case ”Buy skin”

2.3.4 Use case description of ”Pay reward”

<table>
<tr><td>Use case name</td><td>Pay reward</td></tr>
<tr><td>Actor</td><td>System</td></tr>
<tr><td>Description</td><td>The system pays rewards to the user after each match</td></tr>
<tr><td>Precondition</td><td>Match already ended</td></tr>
<tr><td>Main flow of event
(success)</td><td>
<table>
<tr><th>#</th><th>Doer</th><th>Action</th></tr>
<tr><td>1</td><td>System</td><td>Determine the winning player</td></tr>
<tr><td>2</td><td>System</td><td>Calculate the token amount for the number of winning players and reward it to the users' wallets</td></tr>
<tr><td>3</td><td>System</td><td>Update user data and allow the client to exit the match</td></tr>
</table>
</td></tr>
<tr><td>Alternative flow of event</td><td>No</td></tr>
<tr><td>Post condition</td><td>Save transaction</td></tr>
</table>

Table 2.4: Use case ”Pay reward”

2.4 Non-functional requirement

- The game must perform well on a network with a ping ranging from 30ms to 100ms.
- The game server should have stable performance and be scalable for future expansion.

CHAPTER 3. METHODOLOGY

In the context of the graduation thesis, I had the opportunity to explore various technologies and important libraries during the system design and development process. Below are the technologies used and some tools to complete this project.

3.1 Programming languages and Frameworks

Figure 3.1: Programming languages

3.1.1 JavaScript

Since its inception, JavaScript [1] has undergone significant development and has become one of the most popular programming languages worldwide. With continuous advancements in the language and supporting technologies, JavaScript has become an attractive choice for developing online games and can also be used in mobile game development.

Here are some capabilities of JavaScript in game programming:

1. Compatibility: JavaScript is a widely used programming language and is supported on most modern web browsers. This facilitates the development of cross-platform games, as your game can run on different browsers without the need for installing any additional plugins or software.
2. HTML5 and Canvas: HTML5 provides powerful tags and APIs for graphics rendering and image processing through the Canvas element. This allows you to create simple to moderate 2D games without relying on more complex technologies like Flash.
3. Libraries and Frameworks: Several JavaScript libraries and frameworks have been specifically designed for game development, such as Phaser, PixiJS, Three.js, Cocos2d-js, and many others. These libraries help you build games quickly and efficiently, offering built-in functionalities for collision detection,

image and audio management, and much more.

4. Audio and Video Libraries: JavaScript supports libraries and APIs for controlling media, such as audio and video. This enables you to create games with high-quality sound and audio experiences.

3.1.2 Java

Java [2] is a powerful and versatile programming language that can be used to develop games of various structures and scales. Below are the key capabilities of Java in game programming:

1. Cross-platform: Java is a cross-platform language, meaning you can develop games on one platform like Windows, macOS, and Linux and deploy them on these platforms without changing the core source code.
2. Libraries and frameworks: Java comes with many libraries and frameworks supporting game development. Libraries like LibGDX, LWJGL, and JavaFX help alleviate the difficulties in building games and provide features such as image processing, audio handling, window management, 2D and 3D graphics, and much more.
3. Easy integration: Java has excellent integration capabilities with other technologies like databases, user interfaces, and networking systems. This is useful when you need to connect your game to a database to store player data or integrate online gameplay.
4. Security: Java is designed to be highly secure. Running on the Java Virtual Machine (JVM) helps prevent many common security vulnerabilities and restricts unauthorized access to players' computer systems.
5. Performance: While Java is not as performance-optimized as languages like C++ or C#, with optimized libraries and efficient usage, you can still create smooth-running games with good performance.
6. Community and documentation: Java has a large and diverse community with plenty of available documentation, tutorials, and open-source resources. This helps you find support and solutions when encountering issues during game development.

3.1.3 Solidity

In this project, Solidity [3] is the primary language for programming smart contracts. These are the reasons:

1. Support for object-oriented programming: Solidity supports object-oriented

programming, allowing developers to build and manage objects, functions, and variables, making the development and maintenance of smart contracts easier.

2. High reliability: Solidity has been widely used for many years and has undergone numerous tests, ensuring stability and high reliability for smart contracts.
3. Tight integration with Ethereum: Solidity is designed to work well with the Ethereum platform and related technologies. The language provides libraries and functions that facilitate direct interaction with the blockchain and related information, such as handling account addresses, managing cryptocurrency, and communicating with other smart contracts.
4. Readable syntax: Solidity has a syntax similar to programming languages like JavaScript and C++, making it easily accessible and adaptable for developers with programming experience.
5. Popularity and community support: Solidity is a popular language with a large community of developers and experts. This means that developers can easily find information, documentation, and support from the community when encountering issues or needing clarification.

3.1.4 Cocos framework

Figure 3.2: Cocos2d-x Framework

Cocos2d-x [4] is a popular open-source framework for developing mobile and cross-platform game games. Below are some advantages of using Cocos2d-x for game programming:

1. Cross-platform: Cocos2d-x supports game development for multiple platforms, including iOS, Android, Windows Phone, macOS, Windows, and Linux. This lets users quickly develop once and deploy their game on multiple devices and operating systems.
2. Programming languages: Cocos2d-x supports multiple programming languages such as C++, Lua, JavaScript, and more. Although JavaScript may not be

as powerful as C++ (the primary programming language in the traditional Cocos2d-x version), Cocos2d-x JS has been optimized to ensure good performance, especially when running on mobile devices.

3. Community and abundant documentation: Cocos2d-x has a large and active community, which means users can easily find documentation, tutorials, and help from the community when facing challenges during the development process.

4. Supporting tools: Cocos2d-x provides various useful supporting tools, such as Cocos Studio for designing user interfaces and game elements, Cocos Inspector for monitoring and debugging during game runtime, and many other tools.

5. Easy to learn and use: Cocos2d-x is simple for beginners and provides a solid foundation for building games. Users can quickly learn how to use it and start developing their games efficiently.

6. Technical support: Cocos2d-x is actively developed and maintained, so users can trust that they will receive technical support and regular updates to keep their game compatible with the latest versions of operating systems and mobile devices.

In summary, Cocos2d-x is a powerful and flexible tool for developing mobile and cross-platform games, with solid support from the community and advanced features that help users efficiently build high-quality games.

3.1.5 NestJS framework

NestJS [5] is a powerful open-source framework for building scalable server-side applications using TypeScript and Node.js. It follows a modular architecture with dependency injection, promoting code reusability and maintainability. Key features include TypeScript support, compatibility with Express.js, extensive use of decorators for configuration, extensibility through various modules, and built-in testing support. Overall, NestJS is well-suited for creating complex server-side applications, microservices, and APIs with a focus on maintainability and developer productivity.

3.2 Polygon zkEVM

Figure 3.3: Polygon zkEVM

Polygon zkEVM [6] is a decentralized Ethereum Layer 2 scalability solution that uses cryptographic zero-knowledge proofs to offer validity and quick finality to off-chain transaction computation, also known as a ZK-Rollup. The ZK-Rollup executes smart contracts transparently, by publishing zero-knowledge validity proofs, while maintaining opcode compatibility with the Ethereum Virtual Machine. This project's smart contracts will be deployed on Polygon zkEVM to increase transaction speed and reduce costs for each transaction.

3.3 Designing tools

3.3.1 Visual Paradigm

Figure 3.4: Visual Paradigm

Visual Paradigm [7] is a user-friendly and comprehensive modeling tool that offers real-time collaboration, supports model-driven development, integrates well with other tools, automates documentation generation, and provides continuous updates and support. It enhances productivity and efficiency in software development processes.

3.3.2 Astah UML

Figure 3.5: Astah UML

Astah UML [8] is a lightweight UML editor that integrates ERD, DFD, CRUD, and software development features. It assists in designing class and general use case diagrams quickly.

3.3.3 Draw.io

Figure 3.6: Draw.io

Draw.io [9] is a website that provides a platform for users to draw various diagrams, models, and simple charts. Especially users can use it online without the need for installation, with no limitations on usage, and completely free.

3.4 Development tools

3.4.1 IDE Intellij

Figure 3.7: IDE Intellij

In this thesis, IntelliJ IDEA [10] was used. This Integrated Development Environment (IDE) is mainly used for Java programming and other languages like Python and JavaScript. In this case, Java was used for the server side and JavaScript for the client side.

3.4.2 IDE Visual Sudio Code

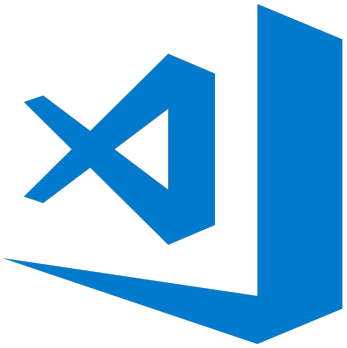

Figure 3.8: IDE Visual Sudio Code

As a popular code editor, Visual Studio Code [11] is frequently utilized by programmers due to its speed, lightweight design, cross-platform compatibility, abundance of features, and open-source nature. VS Code is specialized in creating a private server for communicating with smart contracts and a web interface for connecting users' wallets in this project.

3.5 Source code management tool

3.5.1 Github

Figure 3.9: Github

GitHub [12] is a popular service that provides Git source code repositories for software projects. GitHub includes all the features of Git and also adds social features to allow developers to interact with each other. GitHub is the primary tool to manage the source code throughout the project's completion process.

4.1 Architecture design

4.1.1 Software architecture selection

In this discussion, the forthcoming focus lies on an upcoming gaming application that distinguishes itself from conventional GUI app architectures by adopting a client-server architecture. Additionally, a comprehensive exploration will be conducted into the synchronization and asynchronous processing mechanisms, which are anticipated to assume a critical role in facilitating the game's functionality and overall user experience.

4.1.2 Concepts

"Tick" is a fundamental concept in discrete-event simulation. Discrete-event simulation is based on the basic idea of divide and conquer. In discrete-event simulation, the timeline is divided into small intervals, denoted as "dt," which can be equal or different depending on the purpose. After each dt interval, the simulation engine will update the state of the objects it is controlling based on the rules and constraints implemented within the simulation. Thus, a "tick" is defined as one instance of updating the state of objects in the simulation within the time interval of dt. The game engine used in the project is a discrete-event simulation.

Game action (commonly referred to as "action") is an event that occurs at all client and server sides in the same tick order. For example, in a game with four clients, when a player at client 1 moves their character at tick 100, all clients (2, 3, 4) and the server must also move that player's character at the exact tick 100. The synchronization ensures that the action is executed simultaneously across all relevant entities in the game to maintain consistency in the game state.

4.1.3 Game client architecture

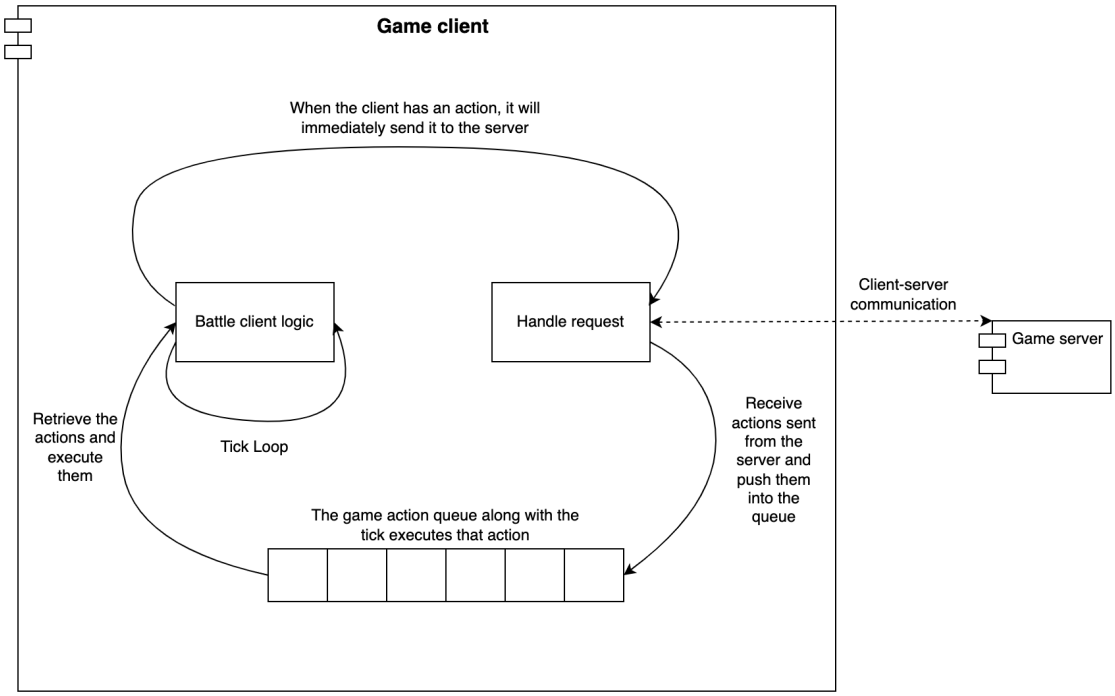

Figure 4.1: Game client architecture

Figure 4.1 illustrates the overall architecture of the game client. Whenever any action is requested from the client, it will be sent to the server through the ”handle request” layer. After processing the request, the server will send back an action queue and the corresponding tick order for all clients in that game session.

4.1.4 Game server architecture

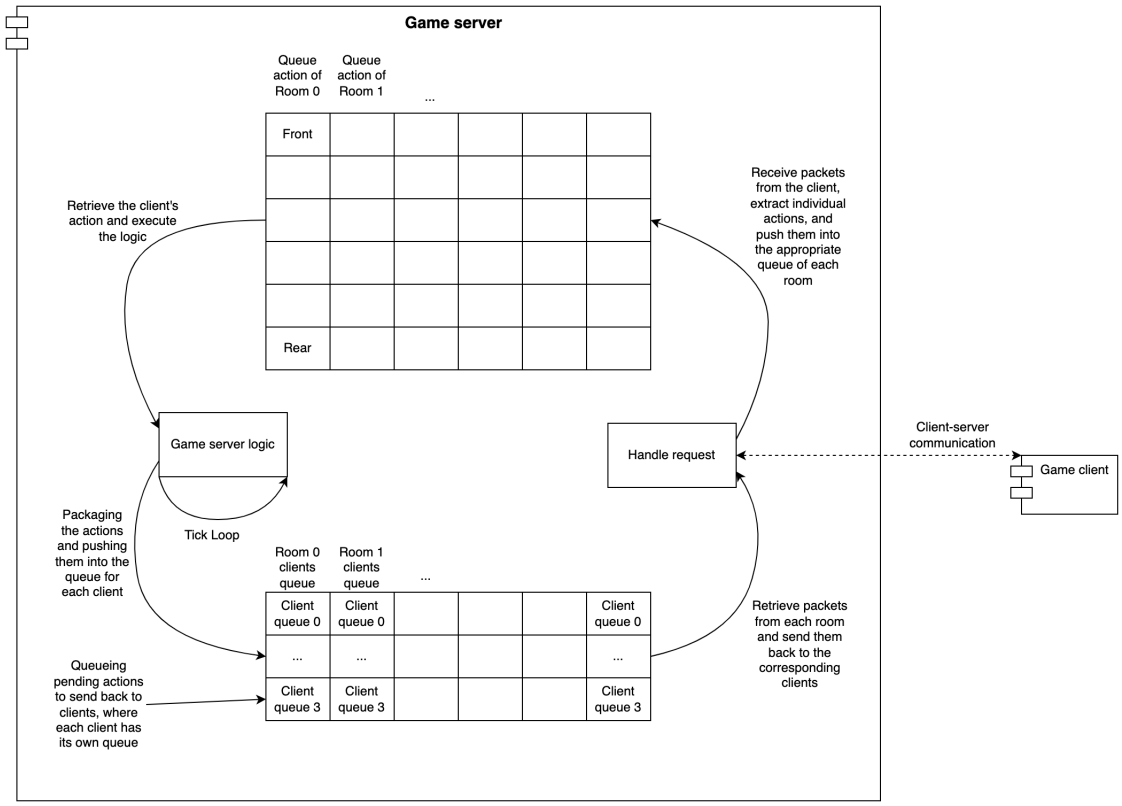

Figure 4.2: Game server architecture

Figure 4.2 illustrates the overall architecture of the game server. Each user on the server will be mapped to their respective current game session. Due to the synchronization mechanism (which will be described in chapter 5, the server will push these actions into separate queues corresponding to each user when the server receives action requests from clients. After verifying the feasibility of these actions, the server will place the activities along with the tick order for their execution into the respective queues for each client and then send them back to the client.

4.1.5 Overall design

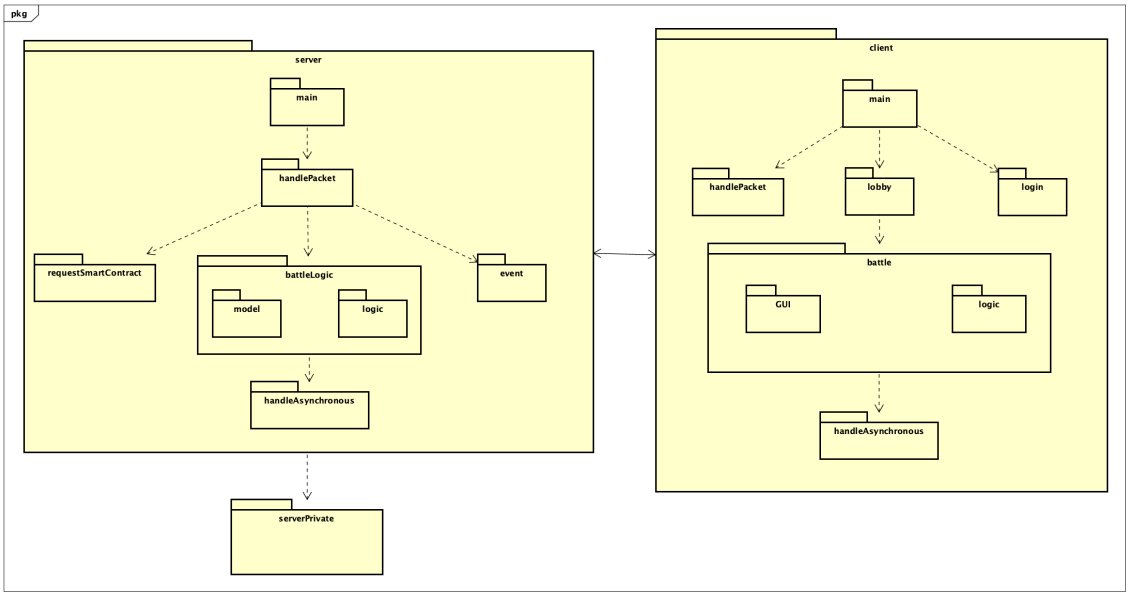

Figure 4.3: UML package diagram

The game design follows a client-server architecture, and the diagrams above will depict the organization of package on both sides. The communication between the two sides will be done through packets, and each side will have functions to handle these packets within the ”handlePacket” package. To support the synchronization mechanism in the game, the battle logic must run the same on both sides. Hence, the structure of the ”battle” package will be similar on both client-server sides. The difference is that the client side will have an additional ”GUI” package within the ”battle” package to handle user interface rendering. Moreover, to determine the timing and location of asynchronous events in the game, both the client-server sides will have mechanisms to handle asynchronous processes within the ”handleAsynchronous” package. Whenever the server needs to interact with a smart contract, it will be handled through classes in the ”requestSmartcontract” package. The server will make requests via a private server to create transactions and receive responses from the smart contract.

4.1.6 Detailed package design

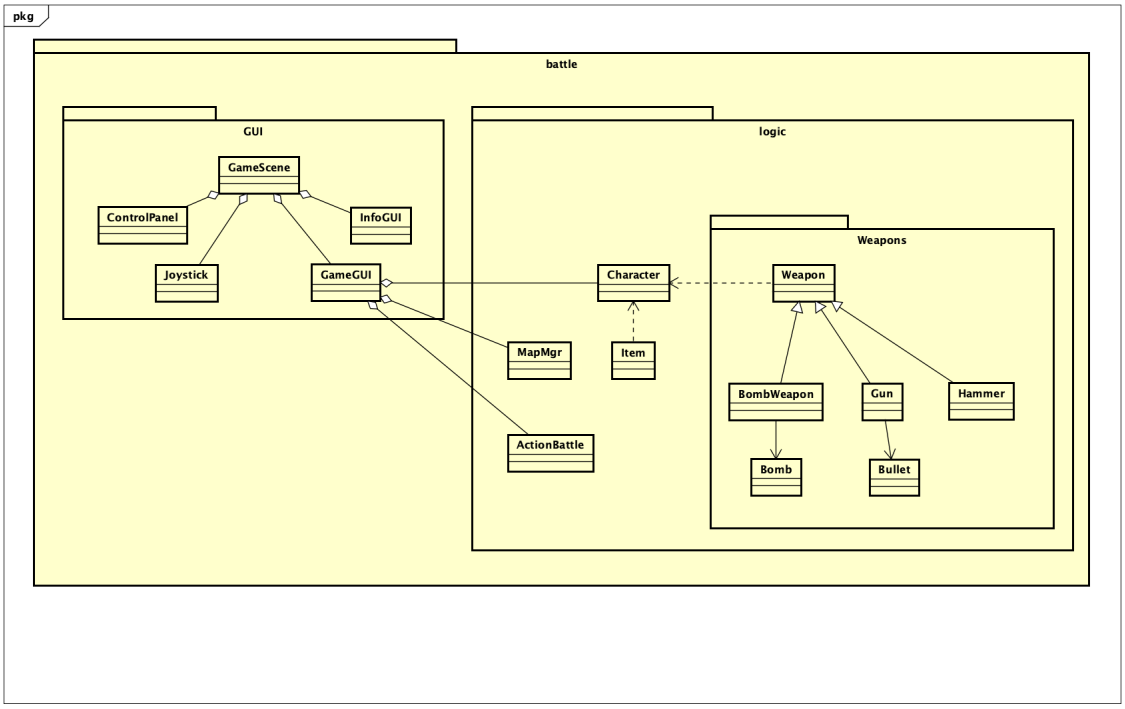

Figure 4.4: Battle package diagram

Figure 4.4 illustrates the class design in package battle on the client side. In general, this package design is similar to the design on the server side. The most significant difference is that the server won't require the "GUI" package for user interface rendering. When a match is started, a GameScene will be initialized. The GameScene will encompass information about the players, functional keys for user interaction, and the battlefield map - displayed by GameGUI. GameGUI will include all player characters, obstacles, items, and various weapons used in the match.

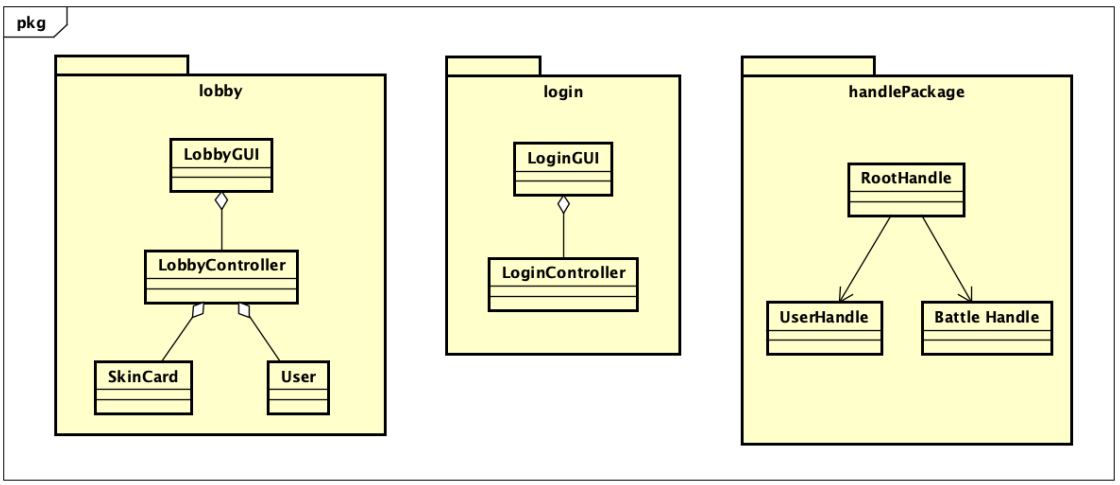

Figure 4.5: Other package diagram

Figure 4.5 illustrates the main package designs on the client side, including lobby, login, and handlePackage. The packages will be designed similarly to the server side, with the only difference being the absence of GUI classes.

4.2 Detailed design

4.2.1 User interface design

The screen parameters and interface design that the game aims for are as follows:

1. A simple and user-friendly interface with a clear and optimized results display tailored for devices with the proposed screen size.
2. The screen resolution is 1920 x 1080.
3. Vibrant colors create a joyful gaming experience.
4. The interface elements are logically arranged with a balanced layout, creating harmony and clearly representing the project's targeted functionalities.

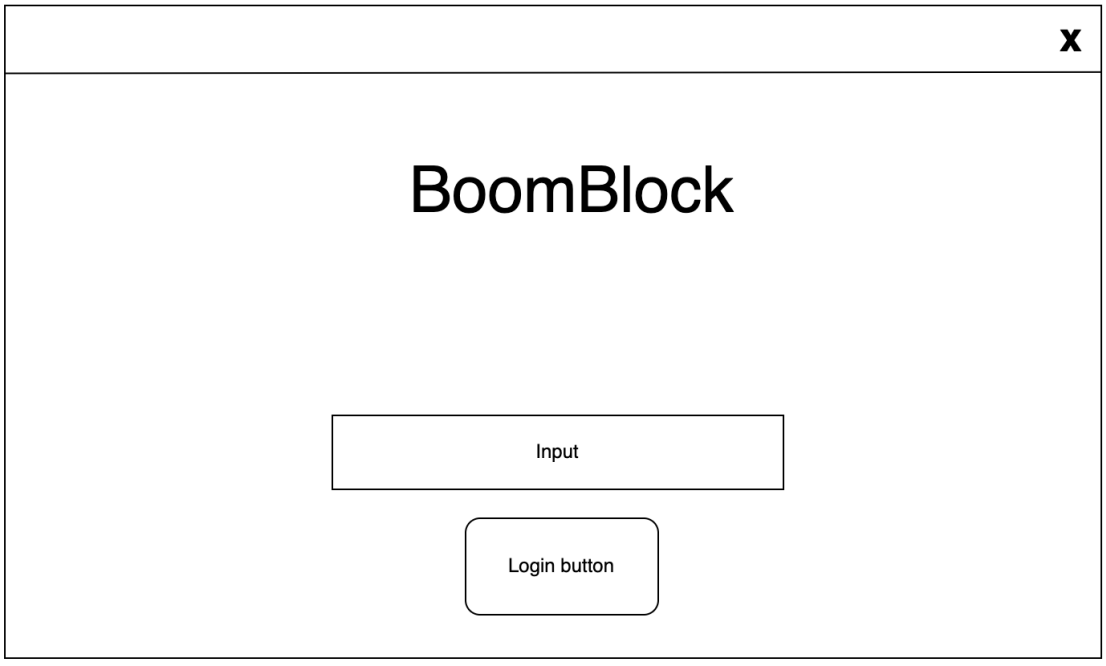

Figure 4.6: Login GUI design

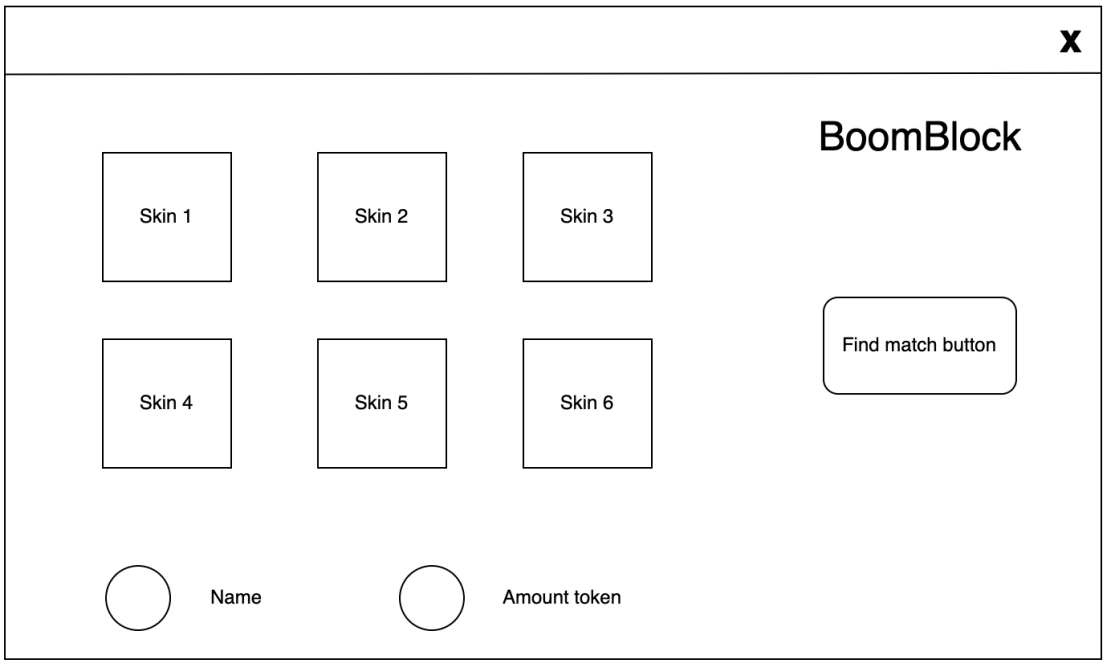

Figure 4.7: Lobby GUI design

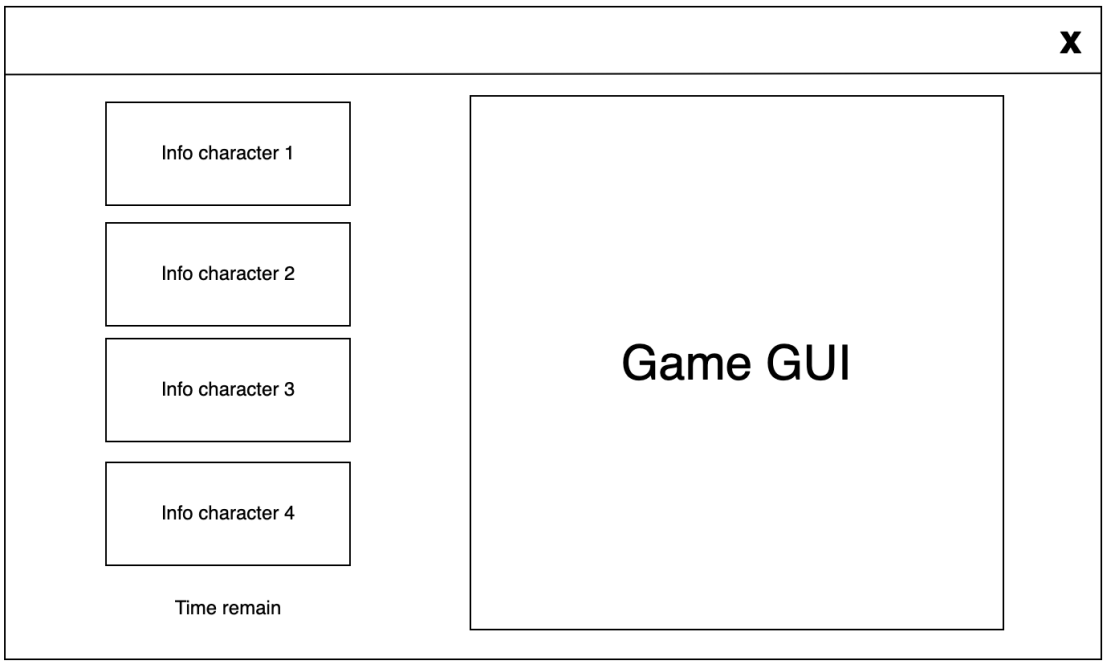

Figure 4.8: Battle GUI design

4.2.2 Layer design

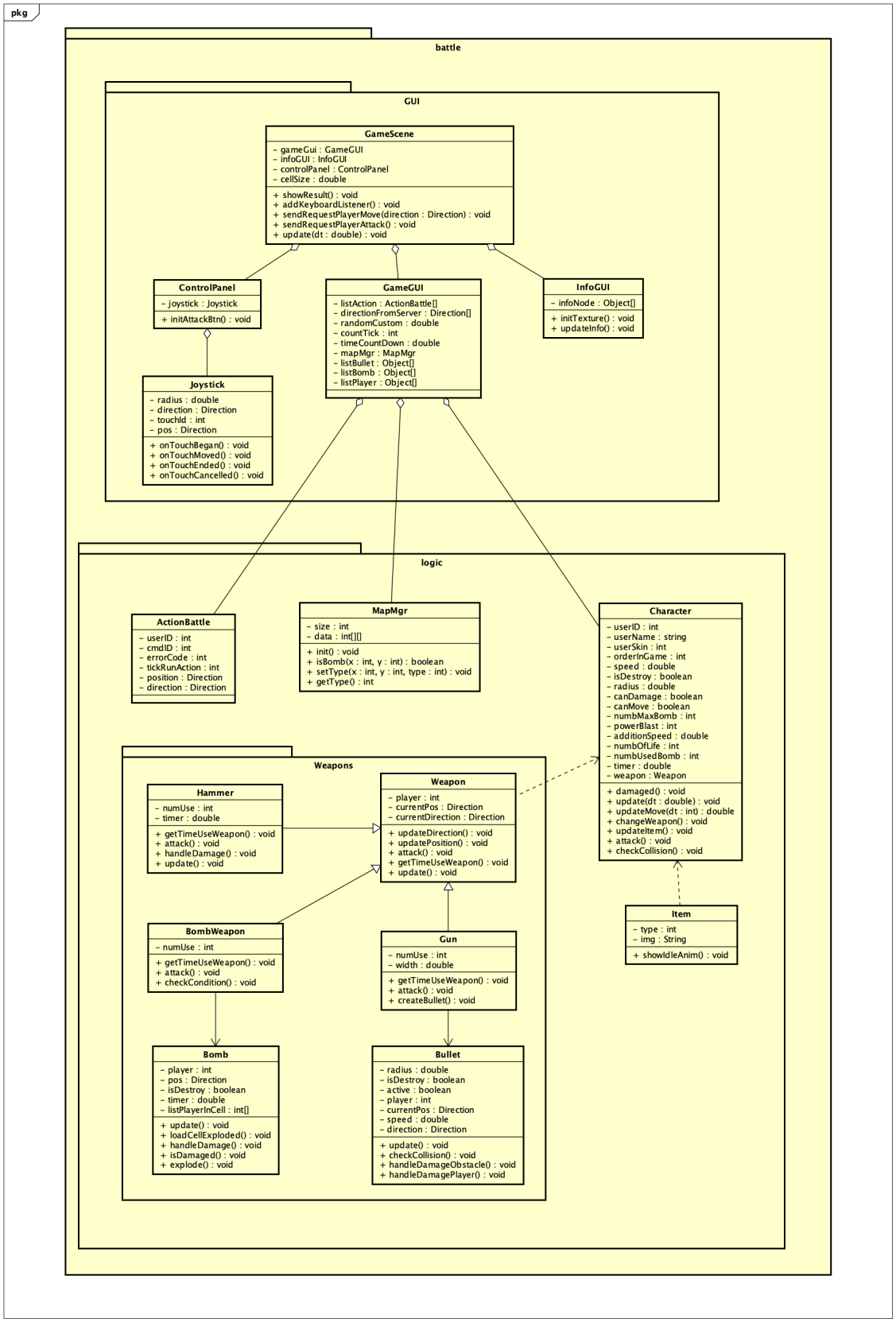

Figure 4.9: Detail battle class diagram

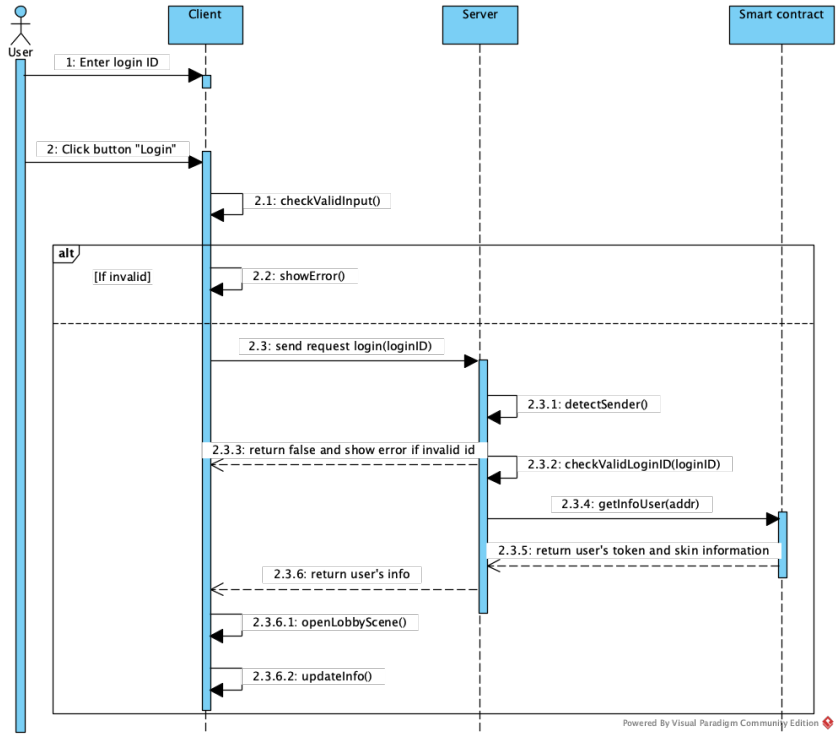

Figure 4.10: Login sequence diagram

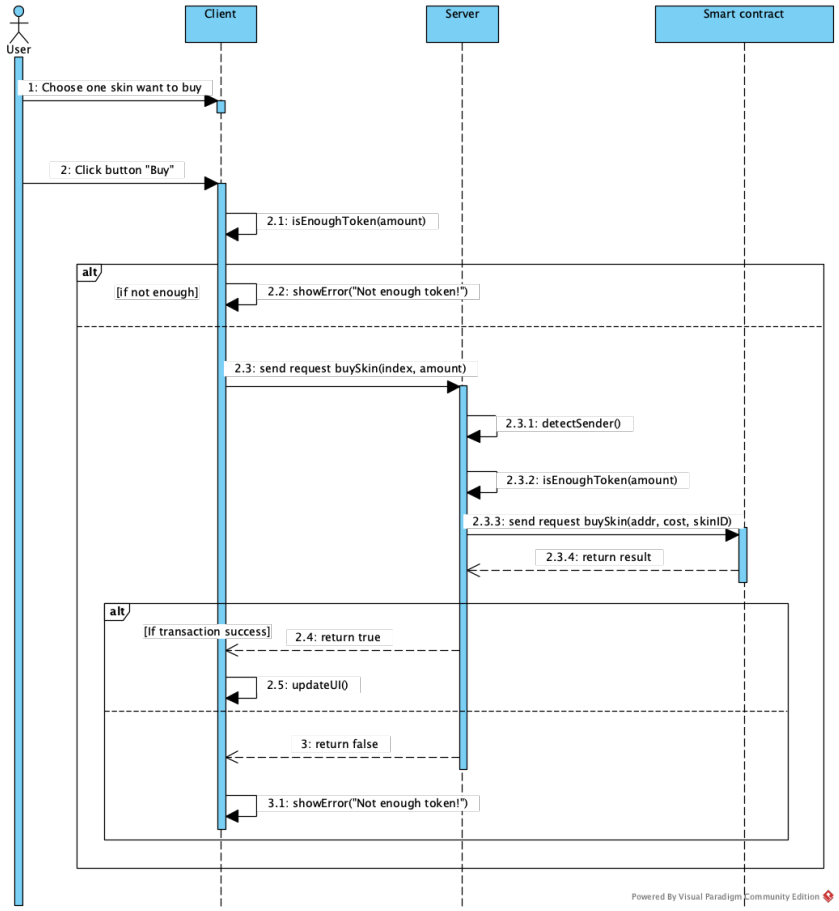

Figure 4.11: Buy skin sequence diagram

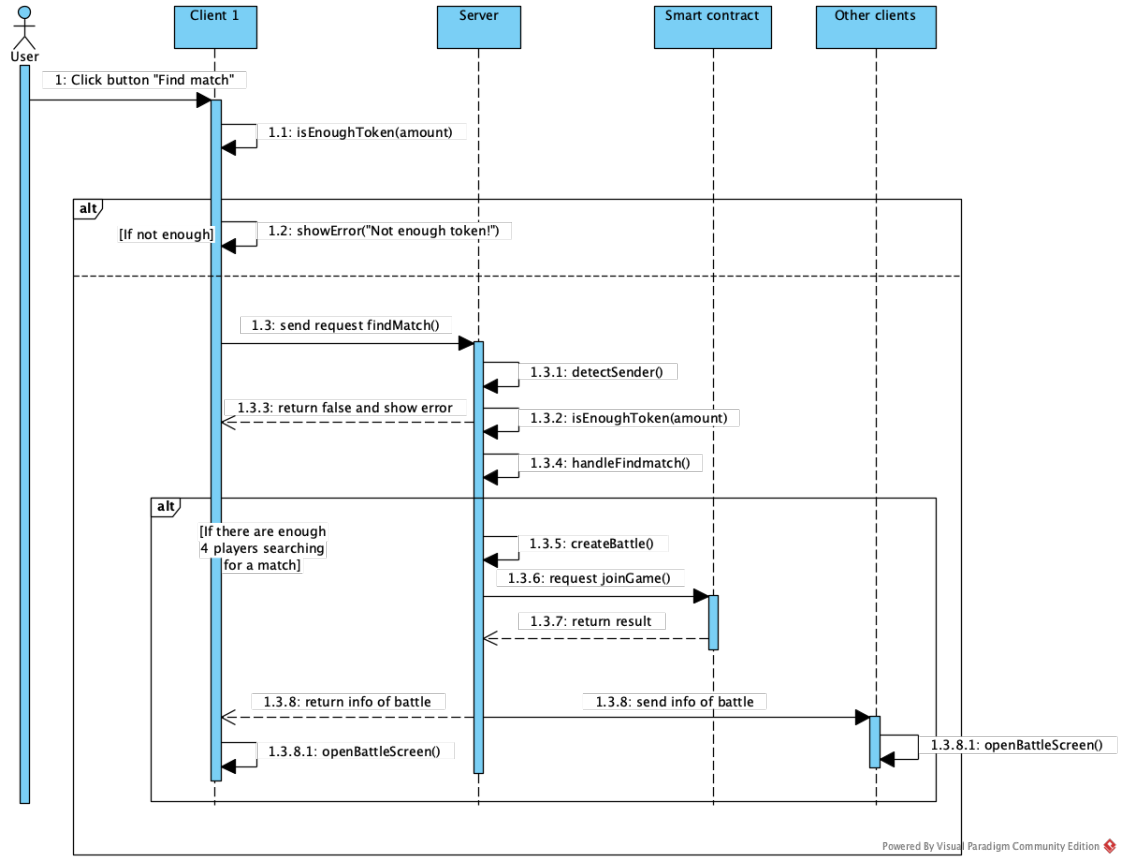

Figure 4.12: Find match sequence diagram

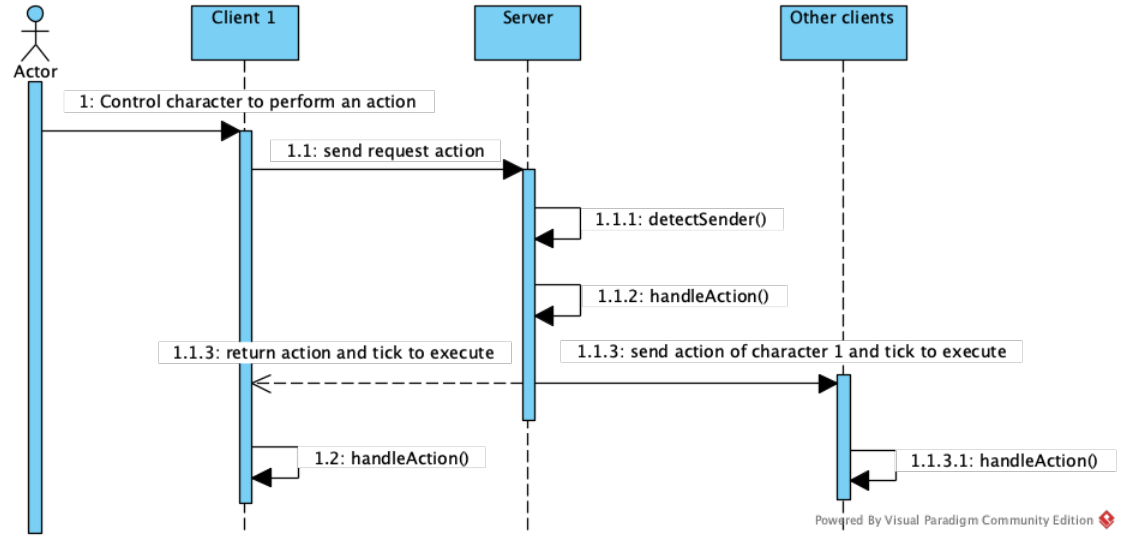

Figure 4.13: Execute actions sequence diagram

4.2.3 Database design

#	Property	Type	Content
1	player_id	String	Id of the player
2	name	String	Name of the player
3	amount_token	Number	Amount token of the player
4	address_wallet	String	Address wallet of the player
5	signature	String	Signature of the player when connecting their wallet

Table 4.1: Player database structure

#	Property	Type	Content
1	skin_id	String	Id of the skin
2	cost	Number	Cost of the skin

Table 4.2: Skin database structure

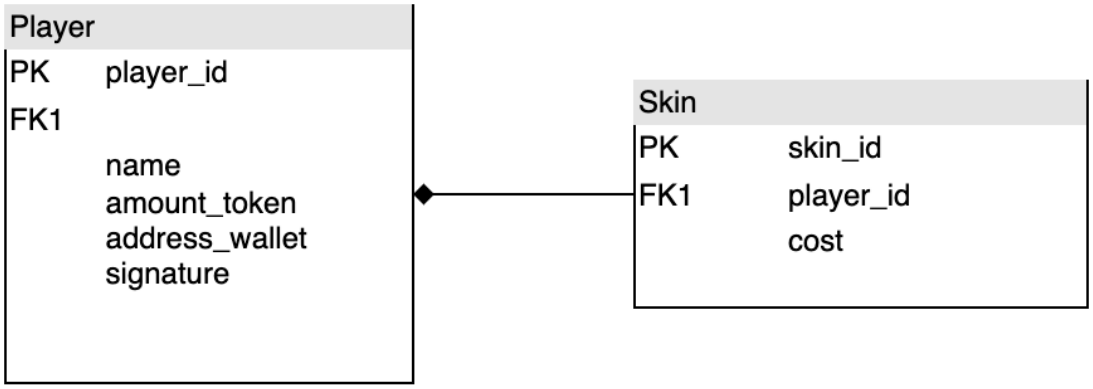

Figure 4.14: E - R diagram

4.3 Application Building

4.3.1 Libraries and Tools

All the tools and libraries project used for developing the project are mentioned in the table below.

Purpose	Libraries/ Tools	Link
Code editor	IntelliJ IDEA 2020.3.4	https://www.jetbrains.com
	Visual Studio Code 1.80.1	https://code.visualstudio.com
Framework	Cocos-2dx JS	https://docs.cocos2d-x.org
	Nestjs	https://nestjs.com
Cache	Memcached	https://memcached.org

Table 4.3: Libraries and Tools

4.3.2 Achievement

After completing the Boomblock game project, some of the achievements project have obtained are:

- User-friendly and intuitive interface as expected.
- Fulfillment of all the functionalities identified in the system requirements analysis.
- Stable performance without encountering any critical errors during runtime.

The project details are listed in the table below:

Content	Achieved results
Line numbers for code	8436
Number of class	134
Number of package	34
Total source code size	1.15 GB
Time for research and coding	4 months

Table 4.4: Project information

4.3.3 Illustration of main functions

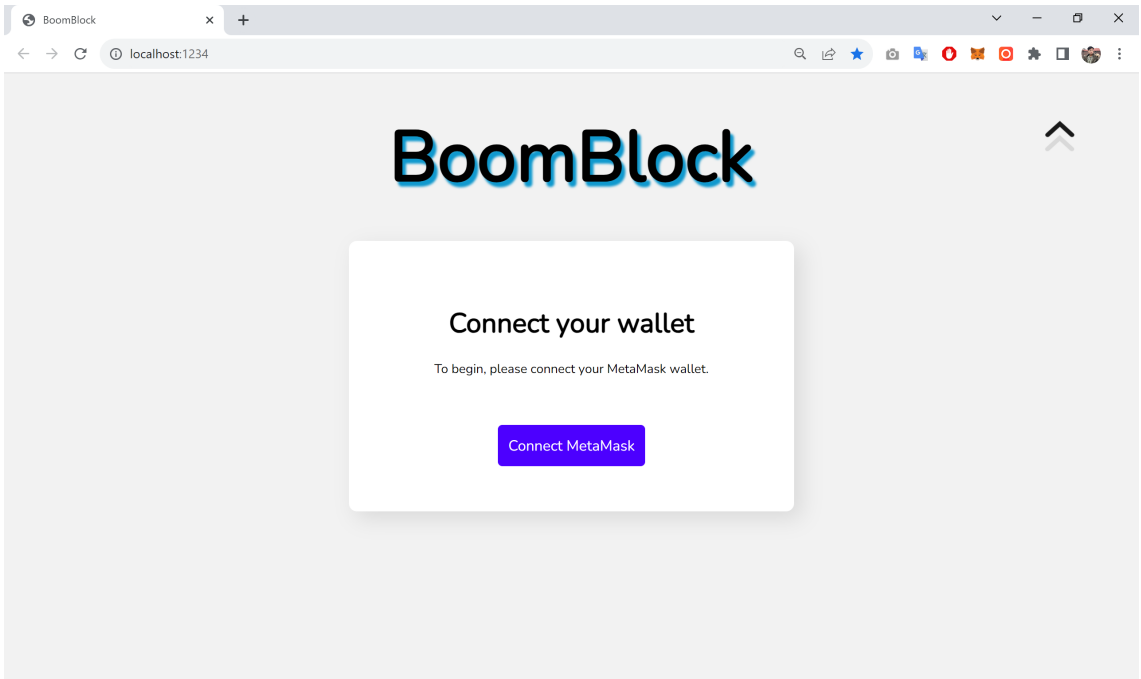

Figure 4.15: Connect wallet screen

Since it is an NFT game, players need to connect their wallets to play the game and perform various transactions within the game.

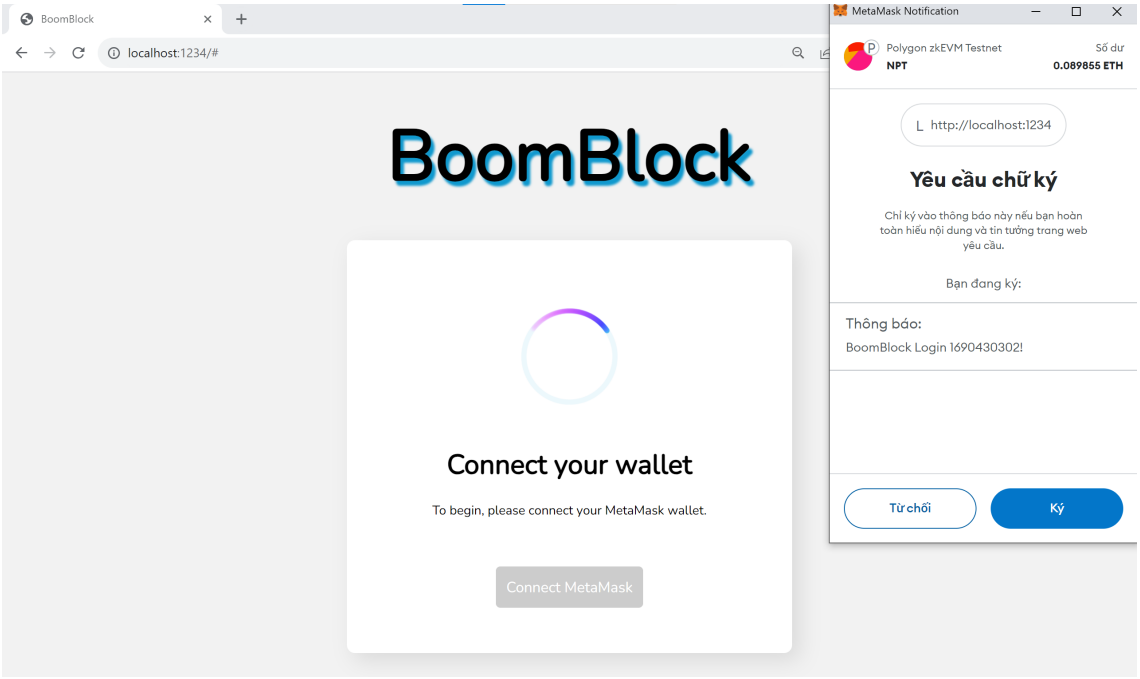

Figure 4.16: Request signature screen

Figure 4.16 shows the screen when the system requests a signature to be used during the gaming process after the user's wallet has been connected.

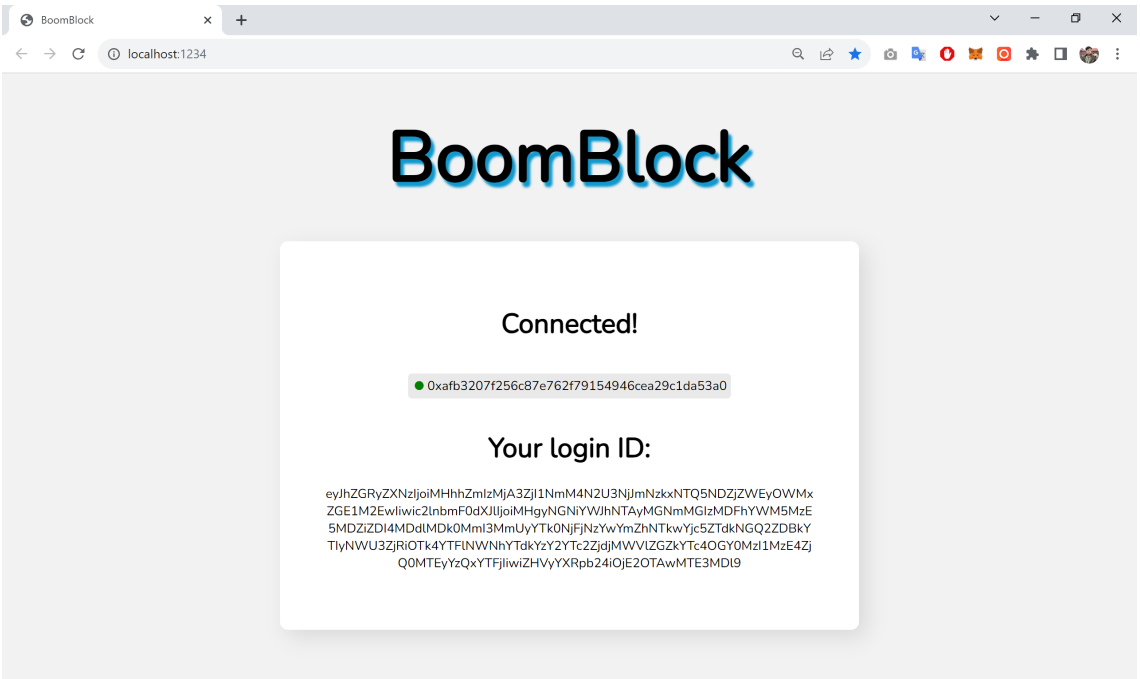

Figure 4.17: Connect wallet screen

After successfully connecting their wallet, the system will display a login ID string on the screen for the player. The player will use this login ID to log in to the game.

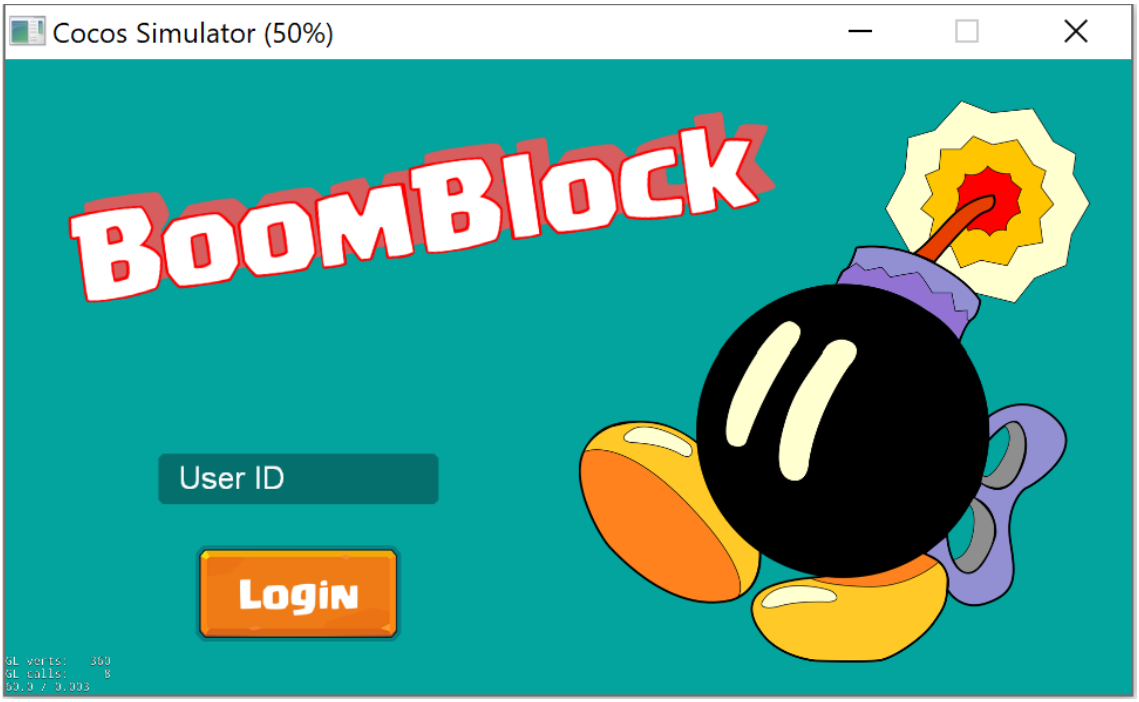

Figure 4.18: Login screen

The image above shows the login screen when players want to log in to the game. Players must log in using the login ID provided when connecting to their wallet.

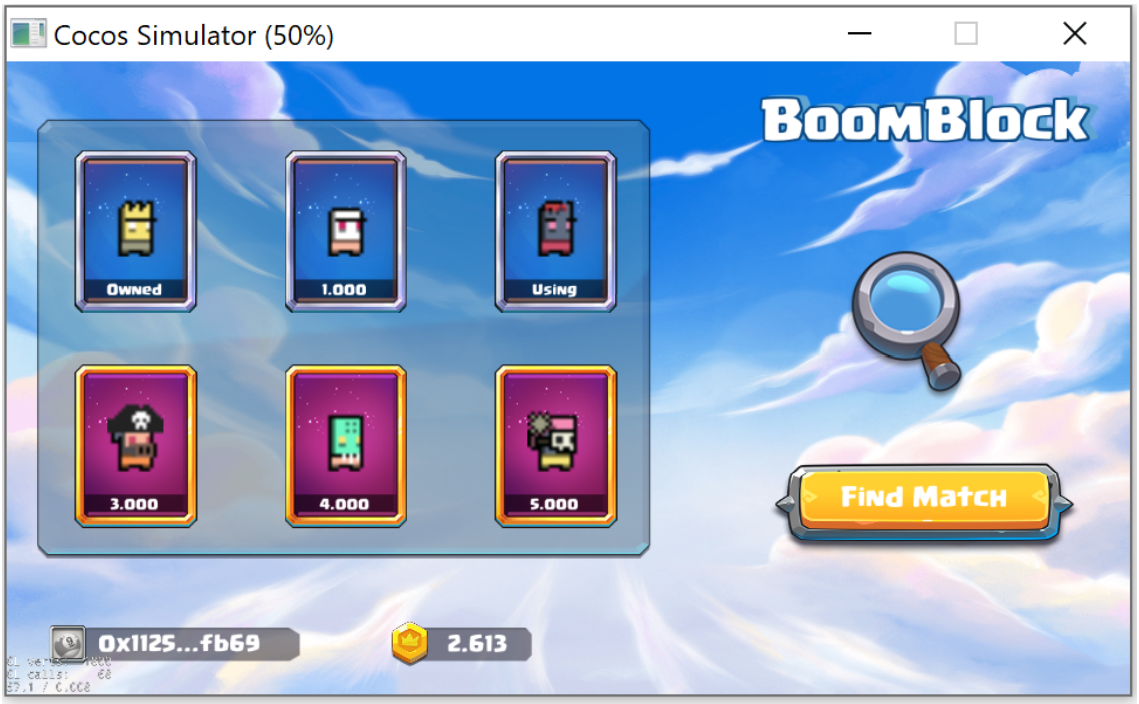

Figure 4.19: Lobby screen

The image above displays the main lobby when a new player logs into the game. The lobby will consist of player information, a ”find match” button, details about

the player's current skin, and the selling price of skins that the user don't own yet. Here, the user can change their current skins or purchase skins they don't already own.

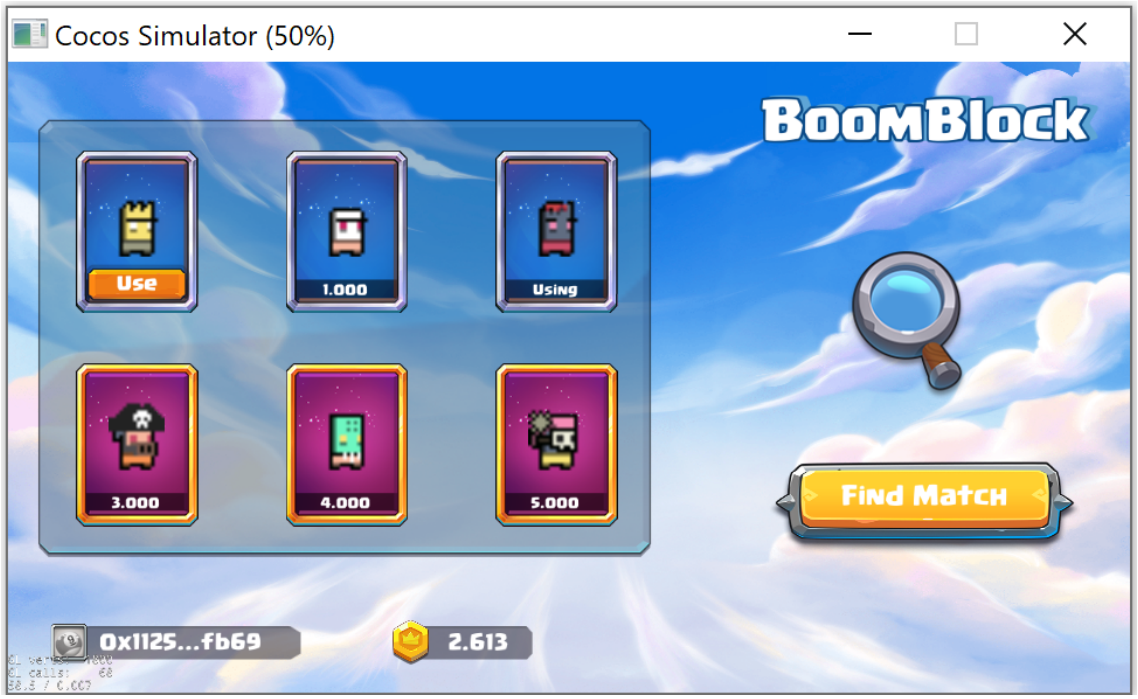

Figure 4.20: Lobby screen

The "Use" button is displayed when the player clicks on a skin they already own. When the player clicks the "Use" button, the current skin of the character in the game will be changed to this selected skin.

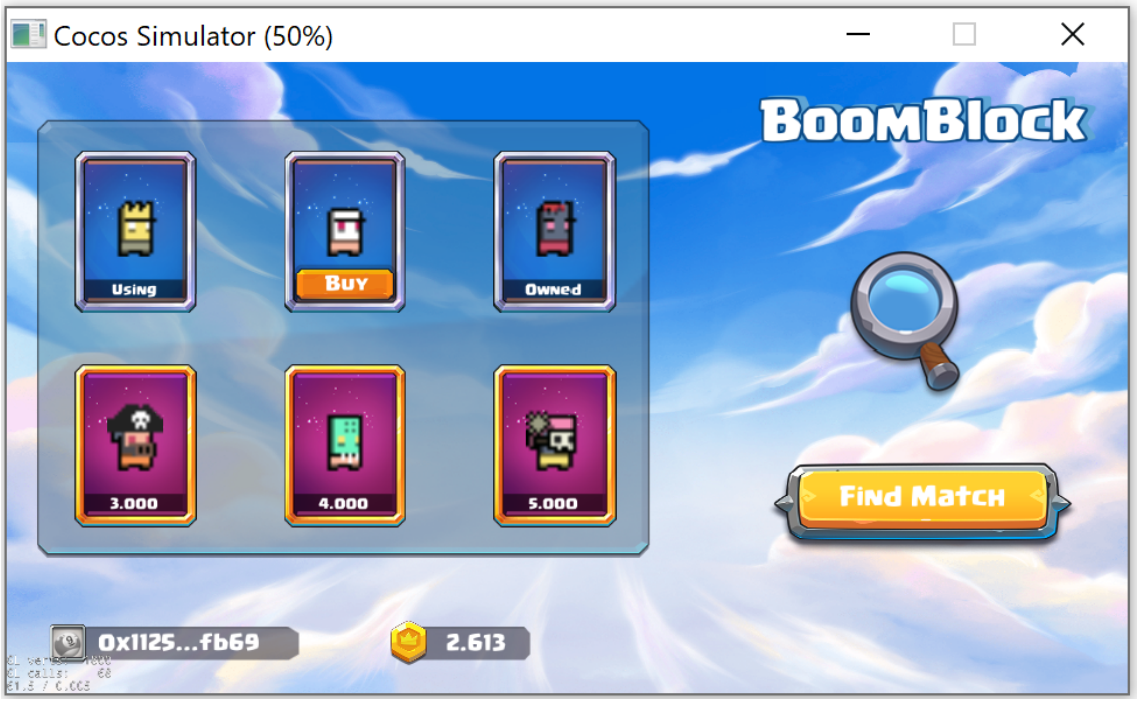

Figure 4.21: Lobby screen

The ”Buy” button is displayed when the player clicks on a skin they don’t own yet. When the player clicks the ”Buy” button, the system will check if the user has enough tokens to perform the transaction. If the user has enough tokens, the transaction will be executed, and the player needs to wait until the transaction is completed. After a successful purchase, the player will own the skin, and the current skin of the character in the game will be changed to this newly purchased skin.

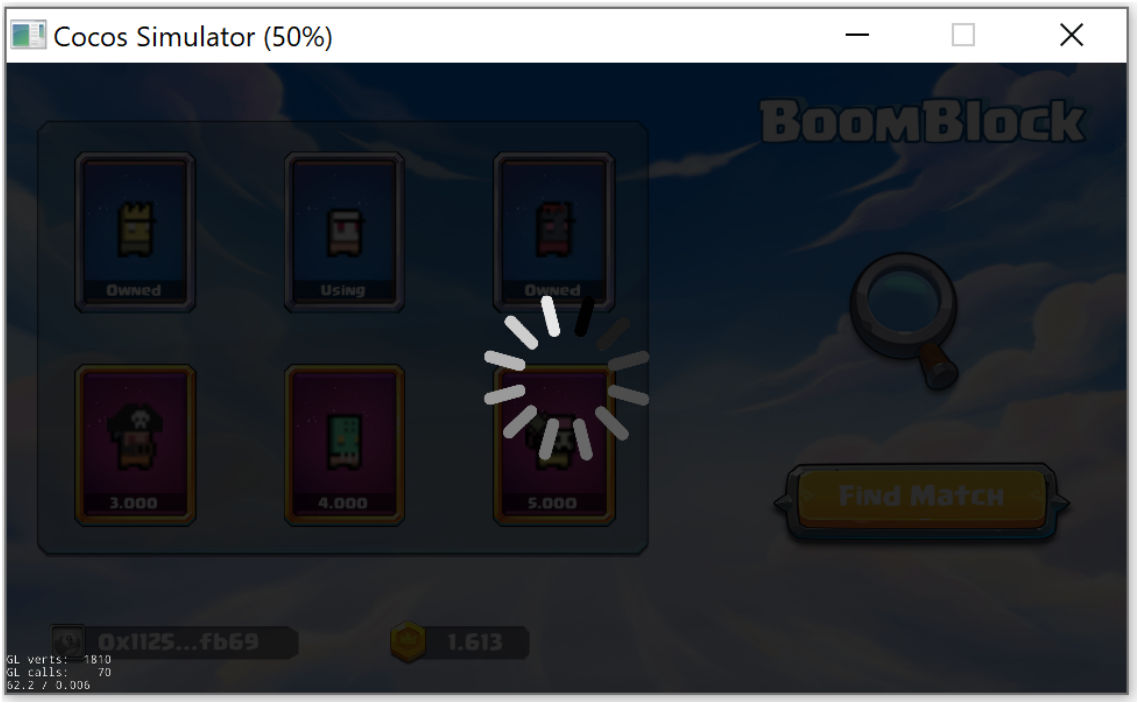

Figure 4.22: Lobby screen loading

The image above shows the screen when the player is purchasing a skin, and the system authenticates the information and processes the transaction for the player.

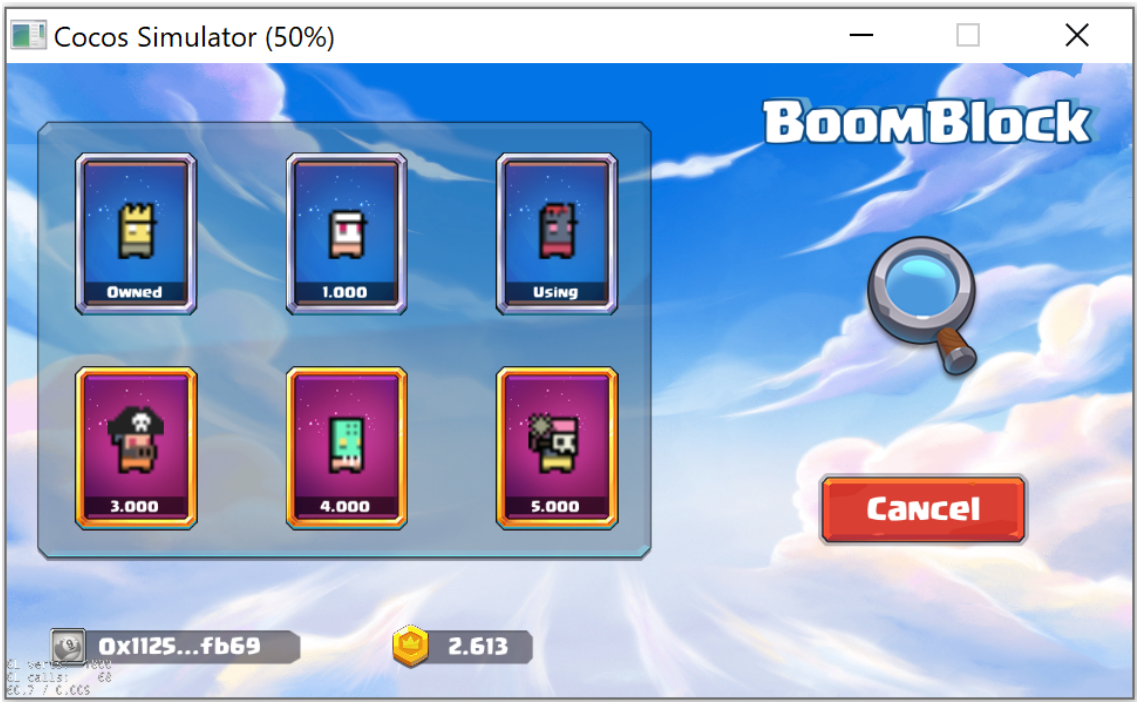

Figure 4.23: Lobby screen

When the player clicks the ”Find Match” button, the system will start matchmaking for the player. During this process, the player can click the ”Cancel” button to abort the matchmaking process. The ”Cancel” button will only appear when the player has clicked the ”Find Match” button and initiated the matchmaking process.

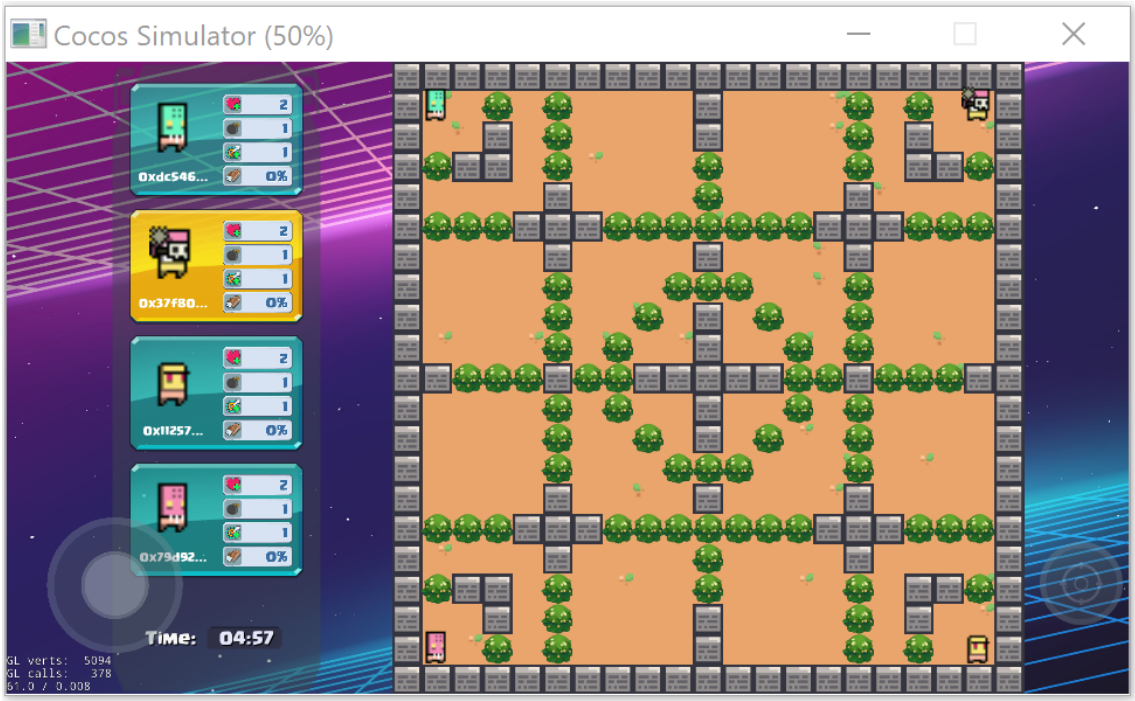

Figure 4.24: Battle screen

The image above shows the in-game battle screen, which appears when the

player has been successfully matched and enters a game session. The screen includes a section displaying information about the players in the game, the main battle arena, and various buttons for player interactions. At the start of the game, the four players will be positioned at the four corners of the arena, and a countdown timer will begin, counting down from 5 minutes.

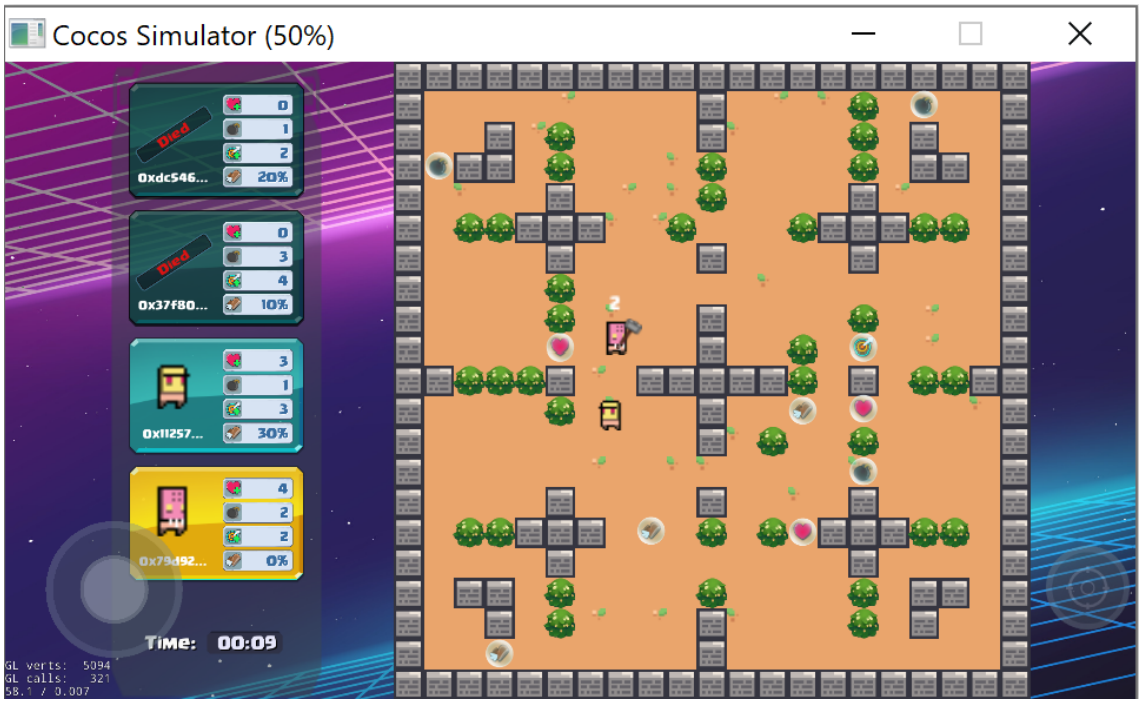

Figure 4.25: In-game screen

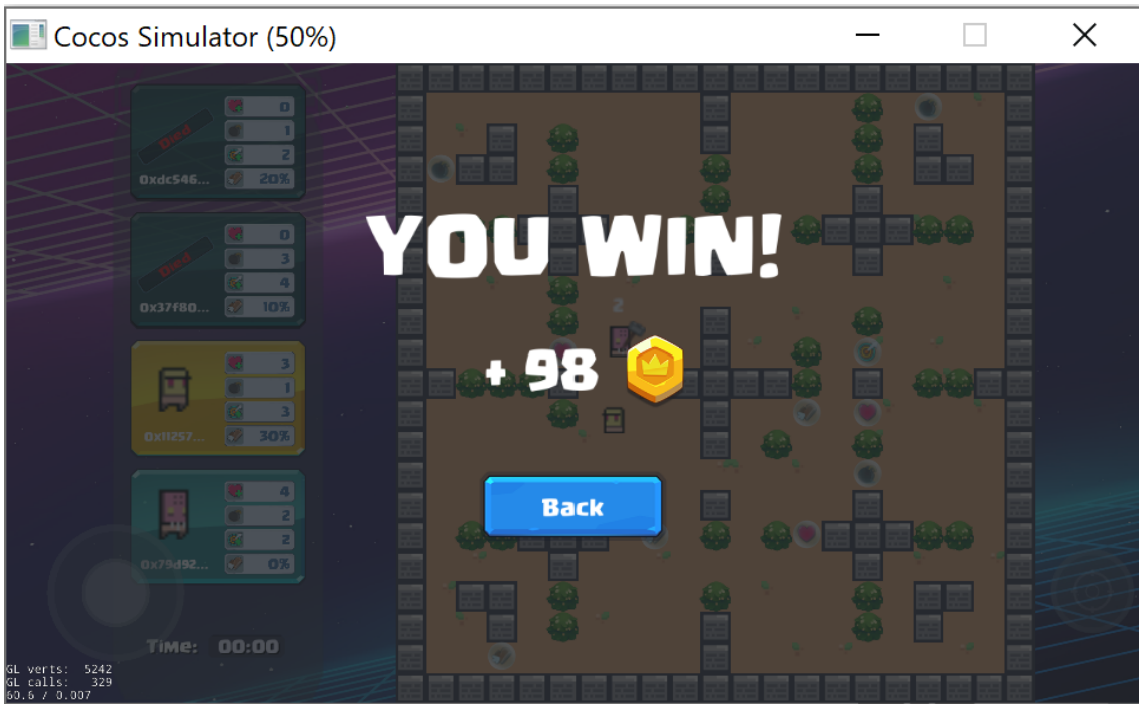

Figure 4.26: Winning screen

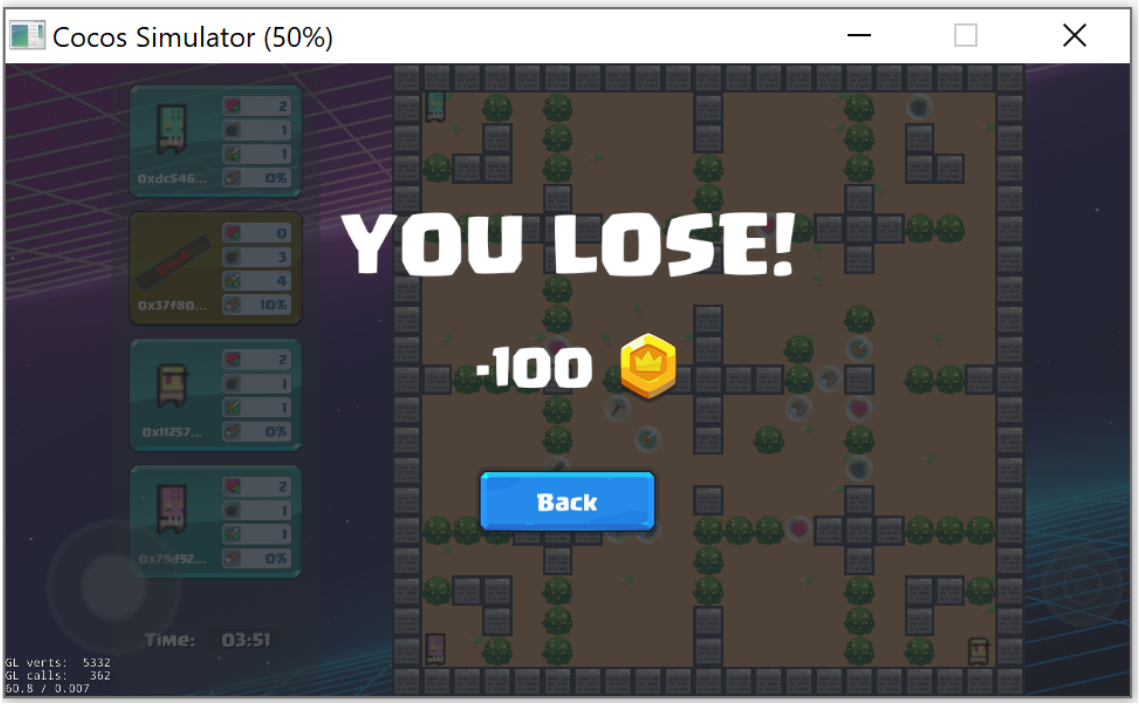

Figure 4.27: Losing screen

4.4 Testing

The testing technique used in this thesis is branch coverage. Branch coverage is a type of white-box test, and its objective is to ensure that each one of the possible branches from each decision point is executed at least once, thereby ensuring that all reachable code is executed.

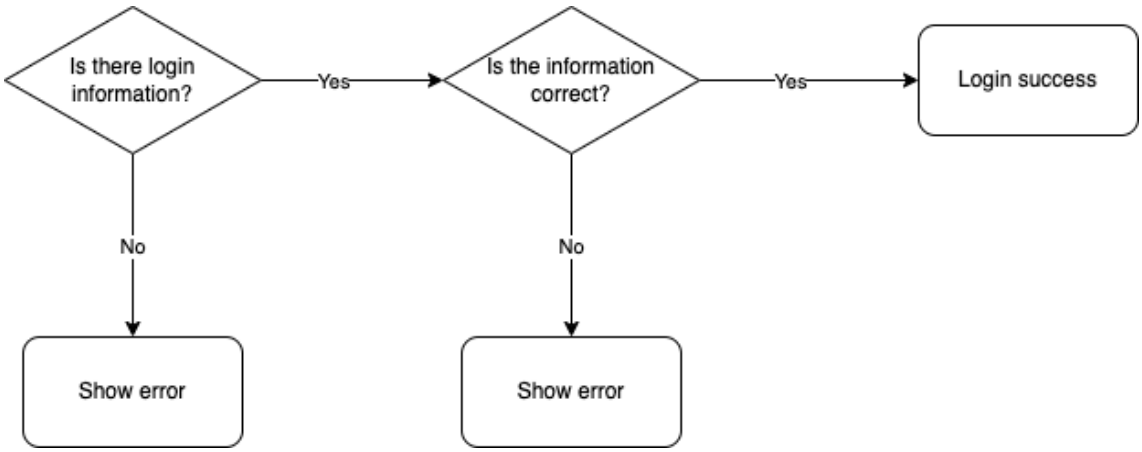

Figure 4.28: Control flow test of ”Login function”

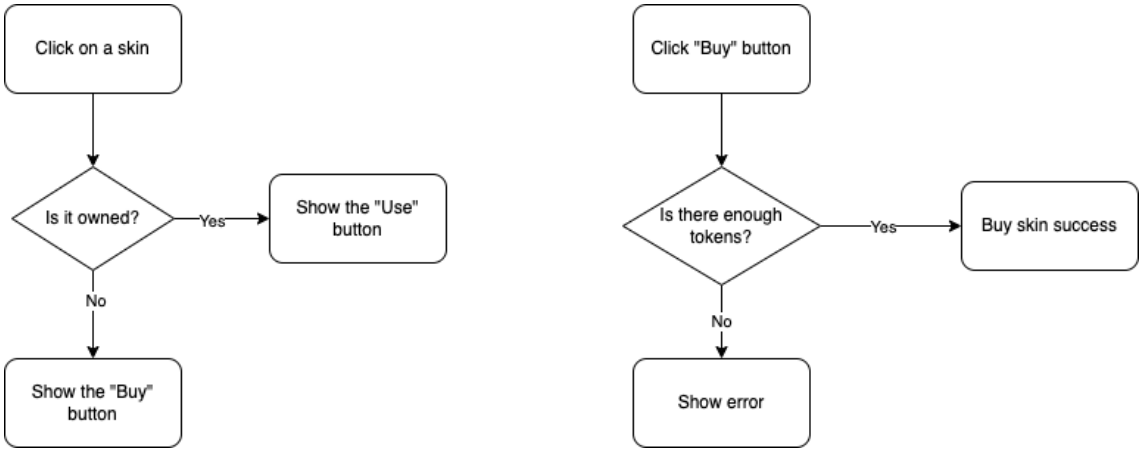

Figure 4.29: Control flow test of ”Buy/ change skin function”

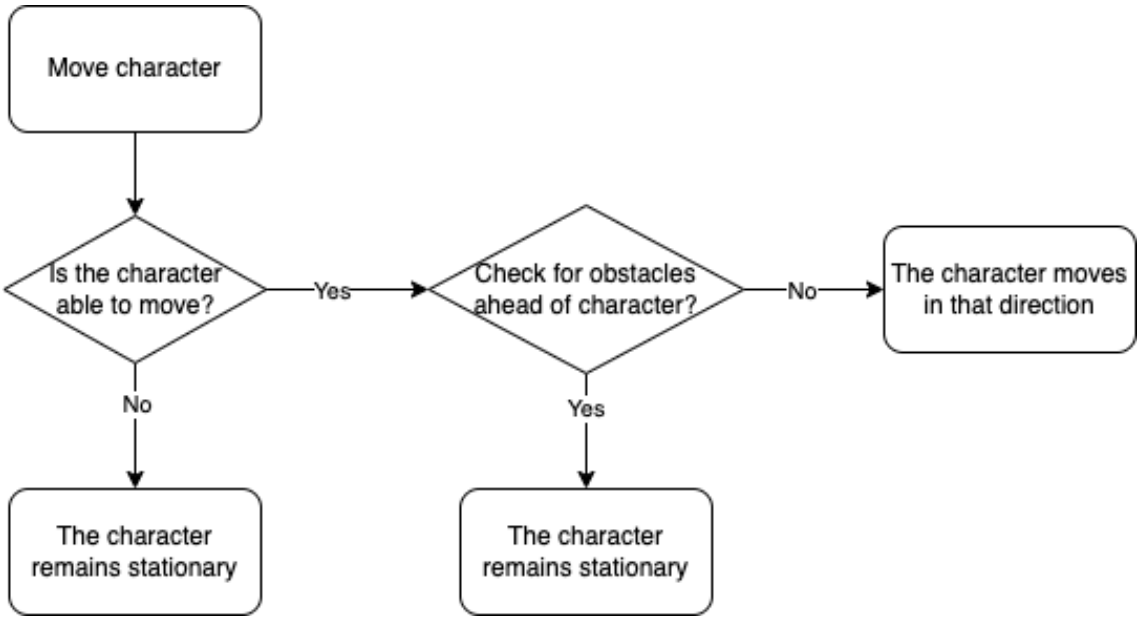

Figure 4.30: Control flow test of ”Character move function”

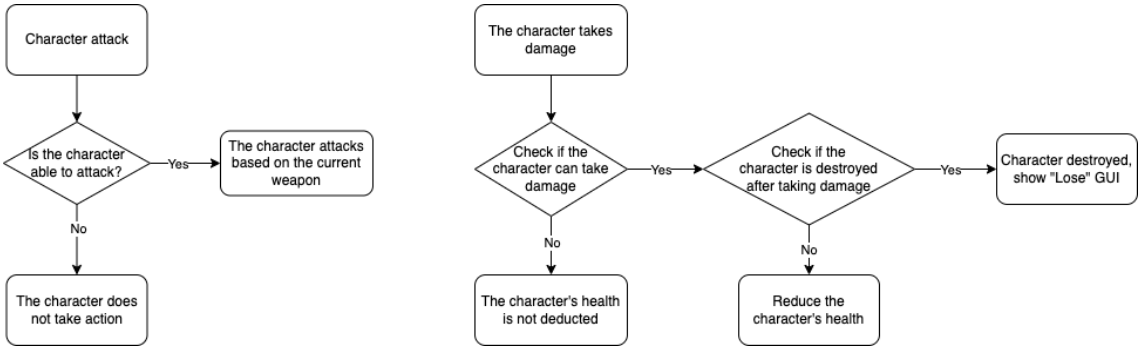

Figure 4.31: Control flow test of ”Character attack function”

4.5 Deployment

The game server is deployed on personal computer with the following technical specifications:

- CPU: Intel(R) Core(TM) i5-8257U CPU @ 1.40GHz (8 CPUs)
- RAM: 16.0 GB
- Storage: 256.0 GB
- Operating System: Windows 10 Pro, x64-based processor

The game client is an application on the player's personal device that displays the user interface and sends requests to the server. The client's configuration needs to be sufficiently powerful to run the game smoothly, and at the same time, to support real-time gameplay, the network connection between the client and server must be stable and have sufficient bandwidth. In this project, Win 32 simulation software on personal computer was utilized to emulate multiple clients participating in the game.

CHAPTER 5. SOLUTION AND CONTRIBUTION

5.1 Sychronization mechanisms

5.1.1 Issues

Currently, there are various synchronization mechanisms used in online games. These mechanisms have their advantages and limitations, and the choice of which mechanism to use depends on the game's requirements and the type of project being developed. Some of these mechanisms include Prediction, Rollback, Entity Interpolation, etc. These mechanisms provide users with smooth and responsive experiences; however, they can be highly complex and require significant time to perfect. Therefore, based on the project's scale and the time needed for completion, selecting the appropriate mechanisms presents significant challenges for developers, and the results may be similar. For these reasons, I would like to present a more straightforward synchronization mechanism that provides excellent efficiency, as explained below.

5.1.2 Solutions

I want to present the synchronization mechanism primarily based on the Lock Step mechanism. Lock Step is a method used in multiplayer game synchronization to ensure the game state's accuracy and consistency across all network participants. In this model, all participants in the game must agree on how to process actions and events, and the results of these actions must be displayed simultaneously on all sides.

From this idea, I have made some improvements. Instead of waiting for participants (clients) to agree with each other, I will enforce that all clients must agree with a single server running similar logic to the clients. In this way, the game experience will be smooth or not, depending on the network speed of each player, rather than relying on all participants in the game.

Before delving into the details of the mechanism, I need to provide some basic information about my game. The game updates its visuals and logic 60 times per second, equivalent to 60 ticks per second. The server will always run ahead of clients by five ticks, meaning that clients will always experience at least a 5-tick delay in the best-case scenario. When receiving actions from clients, the server will check, update logic, and execute that action in the next loop if the action is allowed to be performed.

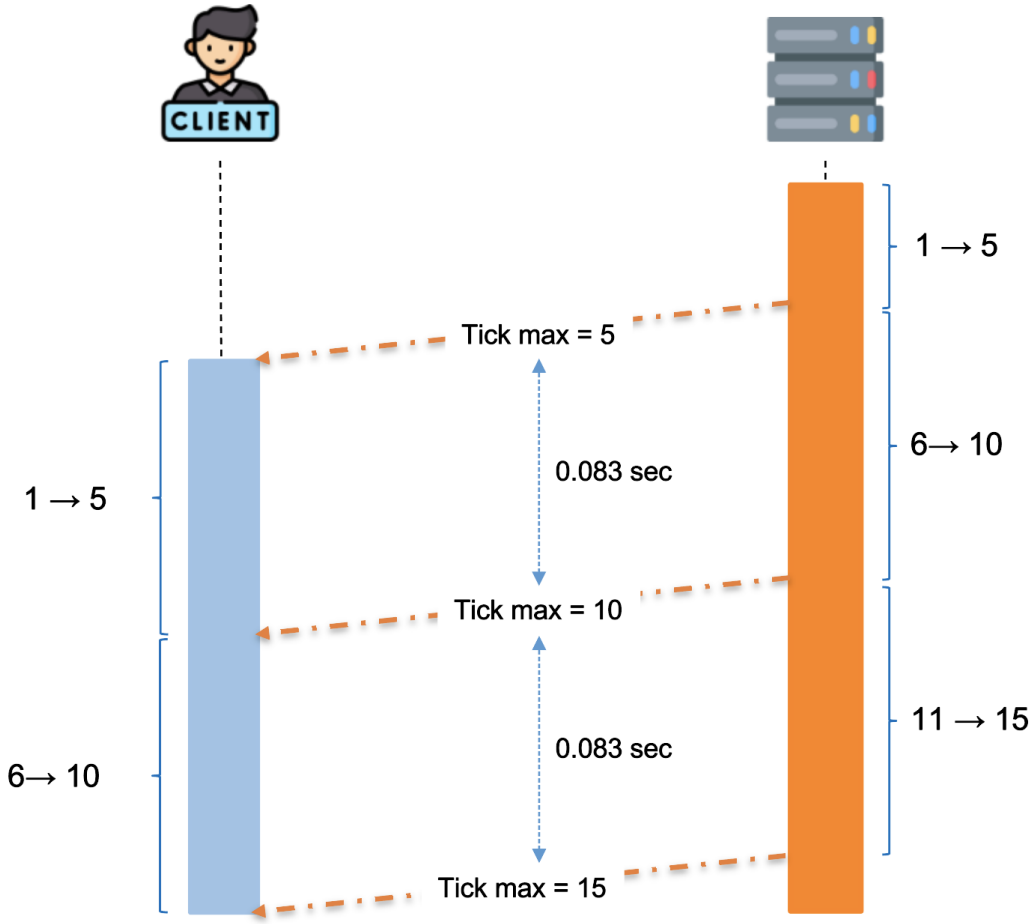

Figure 5.1: Synchronization mechanism demonstration

Figure 5.1 provides an overview of the synchronization mechanism used in the game. The server always runs ahead of the client and processes the client's actions from the previous ticks. The client must wait for the server to return the following tick and can only execute logic up to that tick. Then it continues to wait throughout the game process. It can be seen that in the first five ticks of the match, the client will not perform any action. This issue is not a significant concern as each set of 5 ticks is equivalent to only 0.083 seconds, and players will hardly notice this level of delay. Now, let me detail the steps in the synchronization process for both cases: when the client has an action and when the client does not.

Figure 5.2: Synchronization mechanism step 1

When the match starts, the server will run ahead of the client by five ticks. During this time, the client will not be allowed to run and will wait for the packet that allows it to execute up to the fifth tick sent from the server.

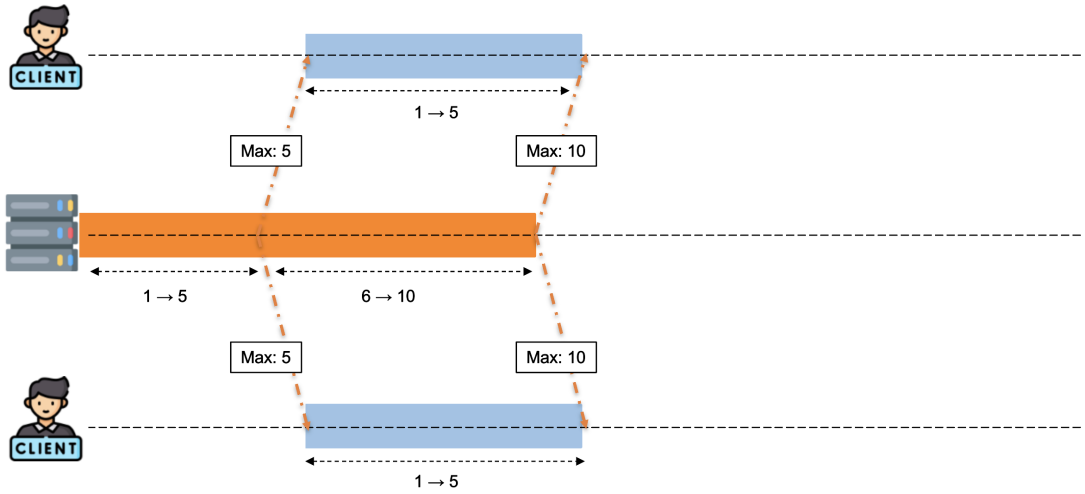

Figure 5.3: Synchronization mechanism step 2

The clients will start running when the client receives the packet from the server, allowing it to execute up to the 5th tick. Meanwhile, the server will receive actions from clients, if any. Since the client does not run during the first five ticks during this step, there will not be any actions to process. When the server needs to send the next tick packet to allow clients to run, the server will execute logic from the 6th to the 10th tick in one loop (without waiting after each tick's time interval). After this execution, the server will send the next tick packet, allowing the client to run up to the 10th tick.

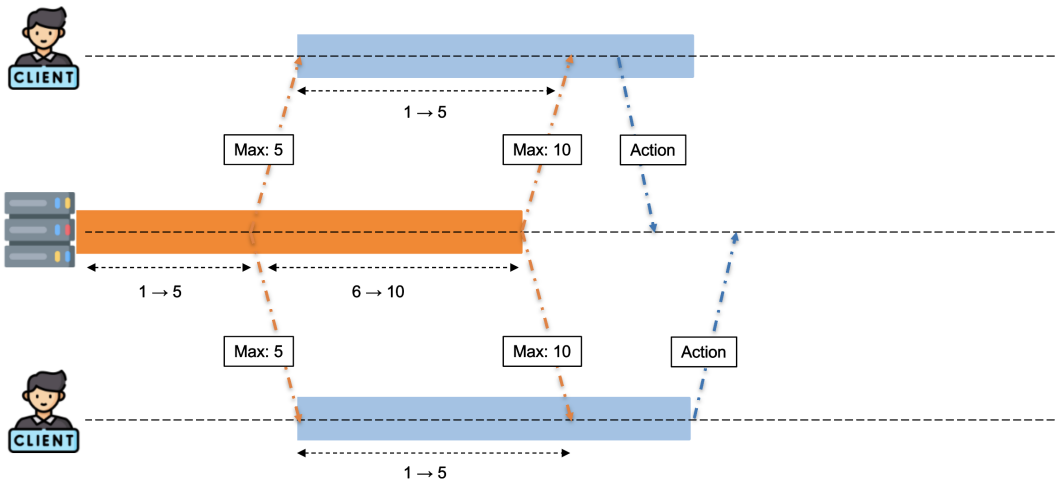

Figure 5.4: Synchronization mechanism step 3

When the client runs from tick 6 onwards, it starts receiving actions from the player. At this point, the client will send these actions to the server, but it is not allowed to execute them locally.

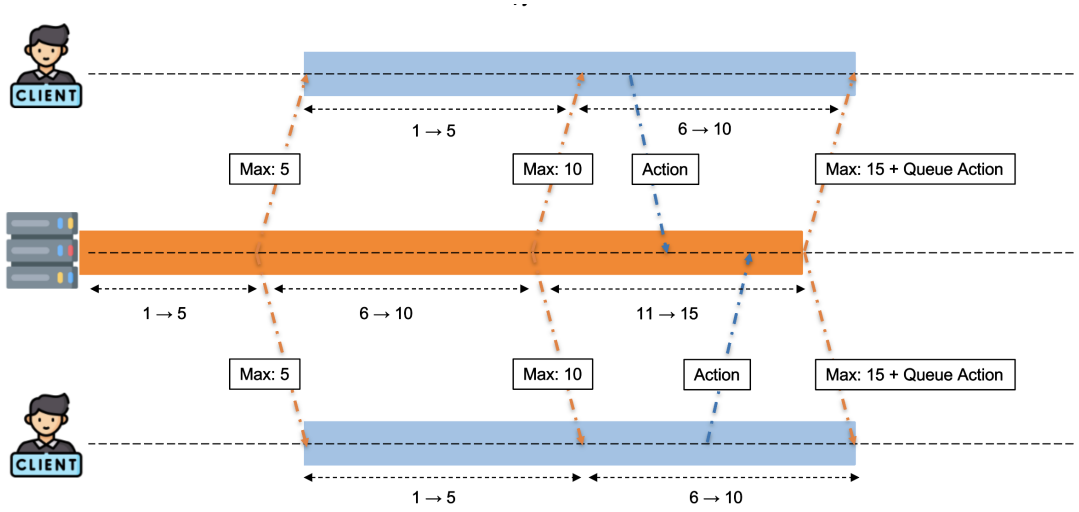

Figure 5.5: Synchronization mechanism step 4

At the point where the server needs to send the packet to allow the client to run to the next tick (in this case, tick 15), the server will execute logic from tick 11 to 15 in one loop while processing the actions in a queue received from the client earlier. After completing the loop, the server will send the packet to allow the client to run up to tick 15, along with the action queue and the timing for each action to occur on the client side.

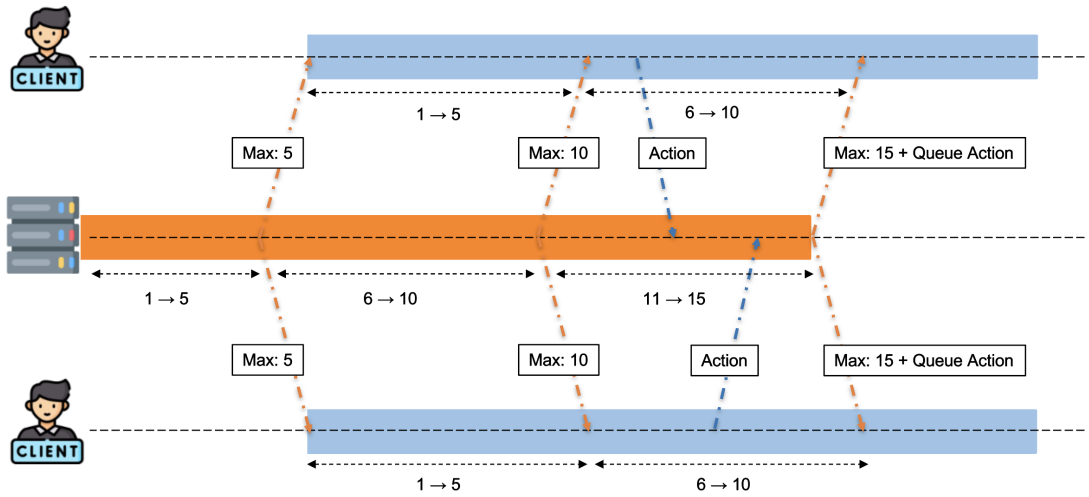

Figure 5.6: Synchronization mechanism step 5

Once allowed to run up to tick 15, the client continues to execute logic and perform the actions sent by the server. While running the logic, the client simultaneously receives actions from the player and then sends them back to the server. The match continues in this manner, and the subsequent steps will repeat similarly to the steps mentioned above.

5.1.3 Result

When applying this synchronization mechanism to the game, player actions have almost zero delay. Actions from different players are also synchronized almost instantly. Moreover, using this method completely prevents cheat hacks from players because all logic runs on the server before being executed on the client, and the server thoroughly checks each received action before being considered for execution. Another noteworthy advantage of this method is its ease of deployment. The time to complete the entire mechanism is quick, yet it delivers the expected results accurately.

5.2 Desynchronization Detection Mechanism

5.2.1 Issues

Detecting asynchrony between clients and servers or between clients is crucial. Once this problem occurs, the match's outcome may differ, leading to data conflicts among players in the game. Asynchrony can happen with any object and at any time. Therefore, there is a need for a genuinely effective mechanism to promptly identify the timing and location of asynchrony among the participating parties.

5.2.2 Solutions

During the game's synchronization process, the desynchronization detection mechanism I use is implemented by logging critical attributes and their sums at each game tick. The logged features include characters, map states, bullets, and bombs on the map.

For characters, I log their positions, speed, health points, current weapons, and their states, including idling, moving, being damaged, or being defeated.

For the map state, I log information about obstacles present on the map, positions, and states of items currently displayed on the map.

For bullets, I log their positions, speed, and movement direction when fired.

For bombs, I log their positions and count down the remaining time before detonation.

Due to the synchronization mechanism used in the game, the server always runs ahead of the client by a certain number of ticks. Therefore, the client can always compare the current tick's log with the previously stored log at each tick. At each tick, the client compares the log sum from the client's side with the log from the server throughout the game process. If any discrepancies are detected between the two logs, it indicates that desynchronization has occurred. When desynchronization is detected, I log the tick order and the desynchronized objects. This helps identify the specific objects and ticks where inconsistencies between the client and server occur.

This desynchronization detection mechanism is crucial during developing and maintaining multiplayer games. It helps improve the player experience and ensures fairness and accuracy of the game data.

5.2.3 Result

When applying the above mechanism, I discovered numerous desynchronization errors in the game and promptly addressed them. Up to the current moment, after undergoing multiple tests, I have almost not encountered any desynchronization cases during gameplay.

5.3 Handling the unstable network connection

5.3.1 Issue

Because the game is designed with a client-server architecture, the network connection quality is a crucial factor in ensuring smooth gameplay. The network quality for each client varies depending on the device and the playtime, resulting in

unstable network latency. When the network latency is unstable, packets sent and received are inconsistent, leading to game stuttering or lag for the players' devices.

5.3.2 Solutions

To address the issue, I have implemented some techniques to mitigate the situation partially. Essentially, I allow the client to run slightly behind the server, up to a maximum of 0.3 seconds (18 ticks). This may cause a noticeable delay for the users, but it significantly reduces the occurrence of game stuttering or lag. This approach involves a trade-off, especially when players have poor network conditions. Additionally, I have applied logic fast-forwarding at the client side to catch up with the server if the network conditions improve later on. This way, the gameplay experience can be enhanced when the network stability improves.

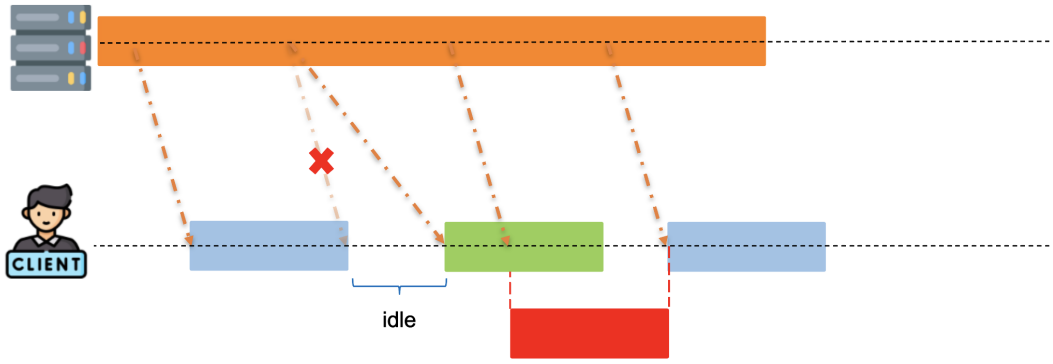

Figure 5.7: Example of an unstable network connection

Figure 5.7 illustrates an unstable network connection, resulting in uneven arrival times of packets. In this situation, the client will be frozen for a while until receiving the next packet to proceed to the tick order allowed by the server.

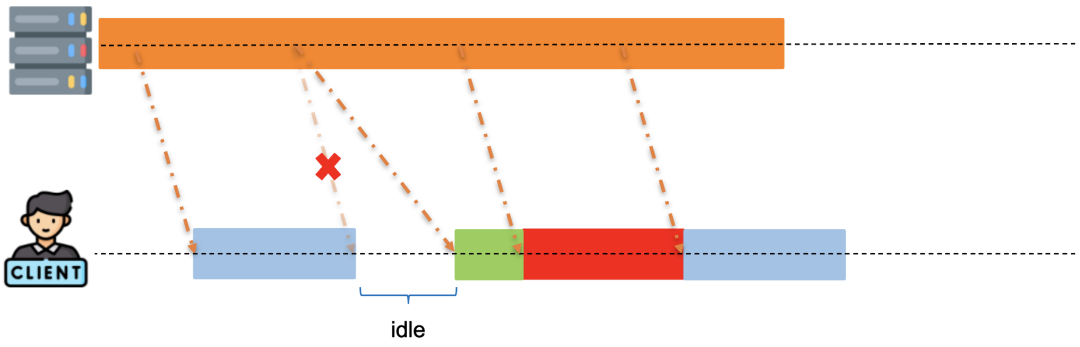

Figure 5.8: Example of an unstable network connection

Figure 5.8 depicts the scenario when the client has not reached the tick order from the previous packet, but the subsequent packet has already arrived. When

this situation occurs, players will experience stuttering in the gameplay due to the significant gap in updating the game state between two ticks.

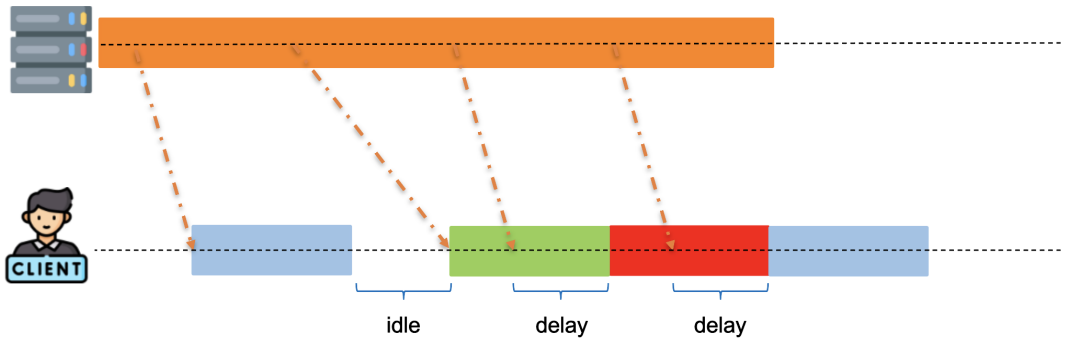

Figure 5.9: Handling an unstable network connection

Figure 5.9 illustrates the state when the client is allowed to run slower than the server up to a fixed maximum duration. In this situation, the client will run slowly and steadily until it reaches the highest tick order among the received packets without fast-forwarding or skipping ticks. This approach helps to reduce the impact of network instability and ensures a smoother and more consistent gameplay experience for the players.

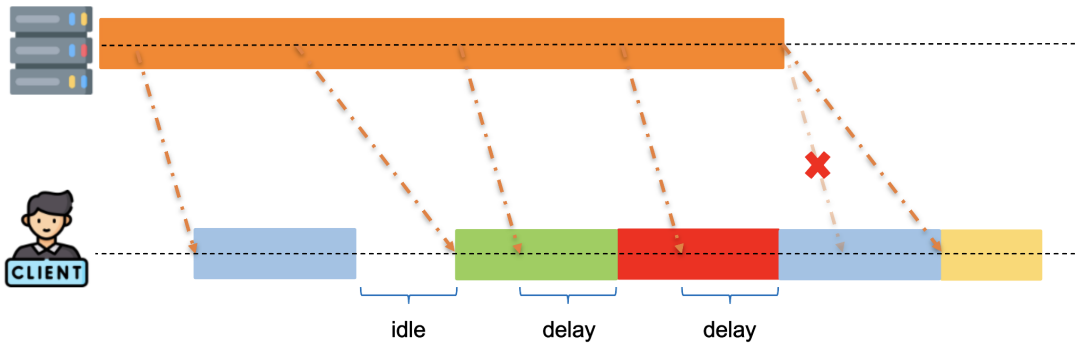

Figure 5.10: Handling an unstable network connection

Therefore, in the future, when there is a delayed packet again, the client will not freeze or wait for the packet to arrive because the client is intentionally running much slower than the server.

5.3.3 Result

When applying all of the above techniques, the issues of frame shuttering and freezing have been significantly reduced during testing on multiple devices with different network conditions.

5.4 Using Rollup to optimize transactions on the blockchain

5.4.1 Issues

As mentioned above, the transaction cost and processing time on the Blockchain network have significantly limited NFT games. NFT games typically involve many transactions, and spending significant costs on these transactions could be more efficient. Therefore, I have chosen to use Rollup to address this issue.

5.4.2 Solution

All of the smart contracts in this project are deployed on the Polygon zkEVM Testnet. Deploying on this network is similar to other popular networks. To delve deeper into this solution, I will provide detailed information below.

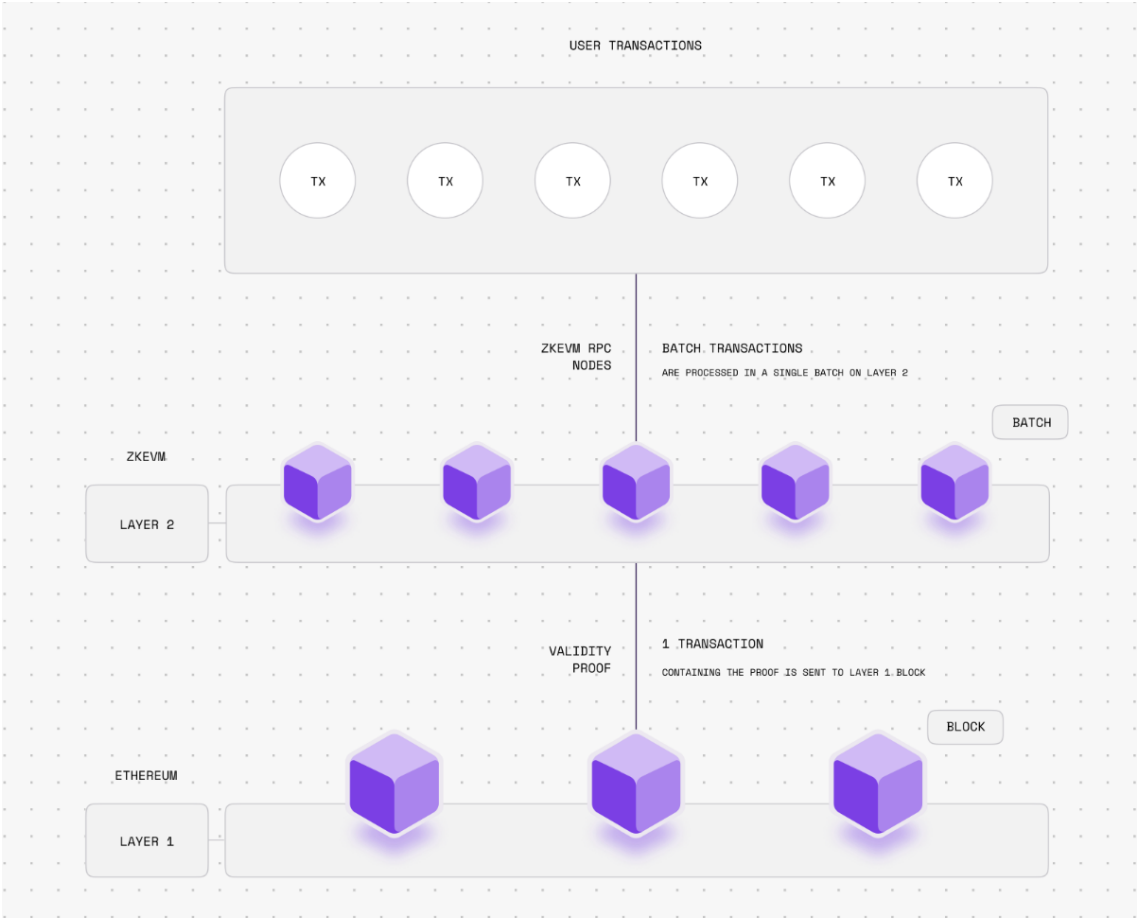

Figure 5.11: The simple operational mechanism of Polygon zkEVM

Polygon zkEVM is an Ethereum scaling solution within the Polygon Network, similar to other layer 2 zk-Rollup solutions. It utilizes zero-knowledge technology to enhance verification, decentralization, and speed. However, zkEVM aims to optimize the verification process further, improving user experience and compatibility with the Ethereum Virtual Machine (EVM) while expanding the ecosystem for Polygon.

Polygon zkEVM utilizes zero-knowledge proofs as validity proofs in transactions to demonstrate the accuracy of off-chain computations. These zero-knowledge proofs (zk-proofs) are based on complex algorithms designed to provide verification and confidentiality for off-chain transactions. Due to the sophisticated nature of zero-knowledge proofs in layer 2, the validity proofs can be easily verified, ensuring the security and trustworthiness of the off-chain computations.

With its functionality as a State Machine, zkEVM technology brings innovation to the operational process, particularly in the execution process when users transact on the network, then follows the process of generating validity proofs to demonstrate the accuracy of computations conducted off-chain. This innovative approach ensures the integrity and trustworthiness of off-chain computations, enhancing the overall efficiency and security of the network.

5.4.3 Result

The zkEVM Testnet has yielded highly feasible results, despite not being fully deployed. Transaction processing time has been significantly improved, while transaction costs have been substantially reduced. This has led to a whopping 90% reduction in on-chain data fees.

CHAPTER 6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This project was achieved through self-study and hands-on experience with various programming languages and frameworks, including Java, JavaScript, Solidity, Cocos2d-x, and Nestjs. Notably, a synchronous mechanism for real-time multiplayer games and an implemented roll-up solution to optimize transactions within the blockchain have been developed. The client-server model, commonly used in software development, stands out as one of the notable architectures.

Professionally, this project has achieved valuable insights into the operation and business process involving the development system. In order to gain this success, much self-study has been engaged to fully understand and meet the requirements necessary to enhance the system's performance and capabilities.

Throughout this project, there have been significant milestones, resulting in a deeper understanding of the software development process and the supplementation of practical insights. The game was skillfully designed and built with user-friendly interfaces, ensuring smooth and efficient user interactions. Moreover, it has acquired expertise in data validation, effectively maintaining data integrity and security. Additionally, the knowledge of database management systems has expanded, focusing on query optimization and preventing SQL injection vulnerabilities.

However, it is essential to acknowledge the limitations within the system. Effective management of large data scales requires further attention. While progress has been made in addressing network-related issues, such as minimizing lagging or freezing in games under poor network connections, a comprehensive solution to this problem is yet to be achieved.

6.2 Future work

The future will focus on developing and experimenting with diverse functions to enrich the gaming experience. The primary objective entails the introduction of a feature that empowers players to buy and sell attack effects during matches, thereby introducing a new dimension of strategy to the game. Implementing this feature is expected to offer thrilling opportunities for players to tailor their gameplay and employ distinctive tactics, fostering a more engaging and dynamic gaming environment.

Furthermore, another focal point of the future development strategy is the diver-

sification of gaming modes to heighten the overall variety of the game. Introducing distinct ways of play is intended to enable players to embrace novel challenges and relish fresh experiences, thus ensuring the game remains captivating and appealing to a broader audience.

In addition to this, there is a plan to implement a ranking system that evaluates players based on their winning points. This system will serve as a means to reward players based on their performance while influencing matchmaking processes, thereby promoting fairness and competitiveness in gameplay.

Enhancing the user interface stands as a paramount aspect of the forthcoming plans. The objective is to achieve a more user-friendly, intuitive, and visually appealing interface to deliver players a seamless and enjoyable gaming experience.

Last but not least, the significance of user feedback is duly recognized, leading to the development of a feature to collect user experience evaluations. This valuable feedback will play a pivotal role in identifying areas for system improvement, thus fostering continuous enhancements and an overall superior gaming experience.

Through the successful implementation of these innovative functions and enhancements, the commitment remains steadfast in creating an immersive and dynamic gaming environment capable of captivating players and ensuring the game's sustained success and growth.

REFERENCE

[1] Brendan Eich, *Javascript*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript (visited on 07/29/2023).

[2] Oracle, *Java*. [Online]. Available: https://www.java.com (visited on 07/29/2023).

[3] Ethereum Foundation, *Solidity*. [Online]. Available: https://soliditylang.org (visited on 07/29/2023).

[4] Cocos, *Cocos2d-x*. [Online]. Available: https://www.cocos.com/en (visited on 07/29/2023).

[5] NestJS, *Nestjs*. [Online]. Available: https://nestjs.com (visited on 07/29/2023).

[6] Polygon, *Polygon zkevm*. [Online]. Available: https://zkevm.polygon.technology (visited on 07/29/2023).

[7] Visual Paradigm International, *Visual paradigm*. [Online]. Available: https://www.visual-paradigm.com (visited on 07/29/2023).

[8] Change Vision, Inc., *Astah uml*. [Online]. Available: https://astah.net (visited on 07/29/2023).

[9] JGraph Ltd, *Draw.io*. [Online]. Available: https://www.diagrams.net (visited on 07/29/2023).

[10] JetBrains, *Intellij ide*. [Online]. Available: https://www.jetbrains.com/idea (visited on 07/29/2023).

[11] Microsoft, *Visual studio code ide (vs code)*. [Online]. Available: https://code.visualstudio.com (visited on 07/29/2023).

[12] GitHub, Inc., *Github*. [Online]. Available: https://github.com (visited on 07/29/2023).