Design and Implementation of a Secure and Synchronized Multiplayer Game Using Unity Netcode for GameObjects

A Graduation Thesis in the Field of Computer Engineering

for the Degree of Undergraduate in Computer Engineering

Recep Tayyip Erdogan University

June 2025

T.C.

RECEP TAYYİP ERDOĞAN UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

DEPT. OF COMPUTER ENGINEERING

Design and Implementation of a Secure and Synchronized Multiplayer Game Using Unity Netcode for GameObjects

SAMET KAYA – VEYSEL GÖRGEN

Members of the Jury

Supervisor: YASİN YALÇIN

Mermber :

Member :

Bölüm Başkanı: YASİN YALÇIN

Jan 2025

RİZE

Table of Contents

GENERAL INFORMATION

1. Introduction

1.1.1 The Development and Importance of Multiplayer Games

Multiplayer games, where two or more can interact within one virtual game environment, are a big deal in the video game industry. Stemmed from the 1960s and grown into a big global phenomenon, it influences not just leisure and sport but also culture, economic life, and even psychology.

1.1.2 History of Development

The very first well-known multiplayer game was "Spacewar!", developed in the year 1962; it gave the opportunity for two players to handle spaceships in a computer-simulated battle [1]. In the 1970s, arcade games like "Pong", a sport similar to tennis, became incredibly popular. The gameplay involved playing on machines in public places, while social contact was emphasized [2]. In the 1990s, Local Area Networks allowed people to connect their computers to play games such as "Doom", which is a first-person shooter. This decade has also followed the development of Massively Multiplayer Online Games (MMOGs). Started in 1997, "Ultima Online" introduced persistent virtual worlds wherein thousands of players can interact [3].

With the adoption of the internet in the 2000s, it made it possible for global multiplayer gaming to start. Online games, like "World of Warcraft" in 2004, developed vast virtual worlds, and "Call of Duty" introduced competitive multiplayer modes that became a staple of modern gaming [3]. By the 2010s, it evolved into cross-platform and free-to-play

models, enabling players to join in from multiple devices with popular games such as "Fortnite" in 2017 and "Among Us" in 2018, among many others, and has now made multiplayer gaming much simpler [4].

1.1.3 Economic Significance

Today, multiplayer games form the economy's backbone in the gaming industry. In 2021 alone, it recorded revenues of $180 billion [5]. A huge chunk of that comes from multiplayer games due to in-game purchases, subscriptions, and microtransactions. Such games as "League of Legends" and "Fortnite" make millions through cosmetic upgrades, battle passes, among many other means. The rise of competitive gaming, better known as esports, has transformed multiplayer games into a spectator sport, with tournaments offering prize pools worth millions of dollars and attracting millions of viewers worldwide [4].

1.1.4 Psychological and Social Impact

Multiplayer gaming does not stop at mere amusement because the gamers accrue several benefits from such games, socially and psychologically. In fact, in the post-COVID-19 era, these have become ways through which society keeps in contact when physical presence is nil. The game "Animal Crossing: New Horizons" allows gamers to build and share virtual worlds with friends and thus has become a bridge to community in isolation [5][4].

Psychologically, a multiplayer video game has the advantage of team spirit, solving problems, and generally improving communication. Indeed, in cooperative games, the players have to share their strategies towards achieving a common goal and hence develop

cooperation. As for competitive games, they enhance reflexes and speed up decisions in stressful conditions. The other side of this coin happens when excessive gaming becomes addictive and contributes to social withdrawal, which calls for reminder balances.

1.1.5 Cultural Influence

Multiplayer games have reached culturally phenomenal dimensions, as the in-game actions sometimes blur boundaries with other fields of entertainment, like live concerts for instance, thanks to "Fortnite" and Travis Scott [4]. It is with such games that one sees the rise of a medium of creativity-creation, personalization, and sharing of content. Communities bloom around such games, forging associations and networks that extend beyond the life of the game itself.

From a very small and localized interaction, multiplayer games have grown into a worldwide industry that affects entertainment, culture, and personal relationships. They have ceased to be mere games and instead have become platforms for interaction, creation, and even professionalism. More seasons in this evolution will definitely ensure that its effect on the modern world is enlarged.

1.1.6 The Impact of Network Topologies on Gaming Experience

Network topology, which refers to the way devices connect and communicate with each other, is a critical factor in multiplayer gaming. It directly affects the speed, security, and scalability of a game. Two primary network topologies are commonly used in multiplayer games. Each offers unique benefits and challenges, influencing the overall gameplay experience.

1.1.7 Peer-to-Peer (P2P) Topology

In a Peer-to-Peer (P2P) network, players' devices are connected directly to each other without the need for a central server. For instance, if a player sends a move or action in the game, it is transmitted directly to other players' devices. This system is often used in smaller multiplayer games or those with a limited number of players [1] [6].P2P works well for small or medium games, but the traffic grows with the square of the player number, so it is hard to scale to very big crowds [7].

The advantages of P2P include faster communication since there is no middle computer or server [1]. It is also cost-efficient for developers because they do not need to maintain servers. However, there are significant disadvantages. P2P networks are vulnerable to security risks because players' IP addresses (unique numbers identifying devices on the internet) are exposed. This can lead to hacking or cheating [3] [6]. Schiele and his team also explain that P2P needs extra work for security, self-organization and game fairness [8]. Additionally, the performance of the game can be disrupted if one player has a slow or unstable internet connection, causing lag or desynchronization, where players see different versions of the game [3].

1.1.8 Client-Server Topology

In a Client-Server network, all players connect to a central server. The server is a powerful computer that handles critical tasks such as processing game actions, maintaining fairness, and ensuring that all players see the same game updates. For example, if one player takes a shot at another, the server verifies the action and informs all players of the result [5].

Pellegrino and Dovrolis show that in Client-Server games the data leaving the server grows very fast (by the square of players), so one server can become a bottleneck [7].

Client-Server networks are highly secure because the server controls all actions, making it harder for players to cheat [1] [5]. They also ensure consistent gameplay, as the server synchronizes all players' game states. This topology is ideal for large-scale multiplayer games such as World of Warcraft or Call of Duty, where thousands of players may be connected simultaneously [6]. However, maintaining servers can be expensive for developers, and latency (delay in communication) can occur if the server is overloaded or far from the players [3].

1.1.9 Hybrid Networks

Some games use a hybrid network, combining elements of P2P and Client-Server topologies. Hybrid plans try to mix P2P’s low cost with the strong control and security that Schiele says a big game needs [8]. For instance, the server might manage critical actions such as preventing cheating, while player devices handle less important tasks like sharing visual data. This approach reduces server load and improves the game's overall speed and efficiency. Hybrid networks aim to balance the strengths of P2P and Client-Server systems [1].

1.2 Key Factors Influencing Game Experience

When selecting a network topology, developers consider several important factors:

- Latency: P2P networks typically have lower latency for small games, while Client-Server systems may introduce delays due to server processing [3].
- Security: Client-Server networks are more secure because the server monitors all activities, while P2P networks are prone to cheating and hacking [1] [3].
- Scalability: Client-Server systems are better suited for large games because they can handle more players, but they require expensive infrastructure [6].
- Fairness: Centralized servers ensure fair gameplay by verifying all actions, whereas P2P networks may face inconsistencies if players have unstable connections [3].

Pellegrino's study proves that when many players join, one server can get slow in both delay and bandwidth [7]. Schiele notes that in P2P, open IP addresses, cheating and data errors are serious risks [8].

1.3 Scope and Objectives of the Thesis

This thesis aims to improve the security and reliability of multiplayer games by focusing on the Client-Server network topology. Multiplayer games, where multiple players interact in a shared environment, require strong network structures to ensure a smooth and fair experience. However, many games still use Peer-to-Peer (P2P) systems due to their simplicity and lower costs, despite their significant security and performance issues. This

thesis examines the weaknesses of P2P systems and demonstrates how the Client-Server model, implemented with Unity's Netcode for GameObjects, can provide a more secure alternative [1] [6]. Because of the delay and load-balance problems that Pellegrino shows for P2P, this thesis chooses the Client-Server way [7].

The P2P topology allows players' devices to connect directly, making it cheaper to set up and faster for small-scale games. However, it has critical vulnerabilities. For instance, every player's device is given equal authority over the game, which means that cheating becomes easier, as there is no central system to validate actions. Additionally, P2P systems expose players' IP addresses, allowing malicious users to target them with hacking or attacks. These issues make P2P unsuitable for games that require fairness, security, and a professional competitive environment [6] [3].

The Client-Server topology, on the other hand, uses a central server to control the game. All player actions are sent to the server, which processes them, ensures they follow the game's rules, and then shares the results with all players. This centralized approach provides better security because the server can detect and prevent cheating. It also ensures that all players see the same version of the game at the same time, solving many of the synchronization issues present in P2P systems. However, the Client-Server model requires more resources, as developers must maintain powerful servers to support large numbers of players [5] [1]. The central server also meets Schiele's rules for clear authority and easy cheat detection [8].

This thesis takes a practical approach by developing a prototype multiplayer game using Unity's Netcode for GameObjects, a library designed to simplify the creation of Client-Server-based games. The game will include features like server-side validation, where the server checks every action to prevent cheating, and centralized control to ensure a fair and consistent experience for all players. By comparing this prototype to what would happen in a P2P-based game, the thesis will demonstrate why the Client-Server model is the superior choice for secure multiplayer gaming [4] [6].

The study also examines specific scenarios that highlight the differences between these two systems. For example, in P2P games, lag or a poor internet connection from one player can disrupt the entire game for everyone else. In contrast, a Client-Server game isolates such problems to individual players, maintaining the experience for others. Similarly, while P2P games can handle only a limited number of players due to the strain on individual devices, Client-Server games can scale up by using more powerful servers or server clusters [3] [1].

The primary objective of this thesis is to show that a secure multiplayer game can be developed using the Client-Server model and Unity's Netcode for GameObjects. By testing the game in different conditions, such as high player counts and attempts at cheating, the study will prove that this approach is safer and more reliable than a P2P system. Additionally, the thesis aims to provide practical guidelines for game developers on how

to implement similar systems, making multiplayer games both enjoyable and secure [1] [6].

In summary, this thesis addresses the critical issues faced by multiplayer games, such as cheating, synchronization problems, and scalability challenges. Through detailed comparisons and a practical prototype, it highlights the benefits of adopting a Client-Server approach over P2P. This work is not only theoretical but also provides a real-world example of how developers can create secure, fair, and professional multiplayer games for modern audiences [1] [3].

1.4 Literature Review

Pellegrino and Dovrolis (2003) analyze bandwidth requirements and state consistency challenges in multiplayer game architectures, focusing on Client-Server (CS) and Peer-to-Peer (P2P) models. The CS architecture centralizes communication through a server, facilitating state consistency and cheat prevention, but server bandwidth demand grows rapidly as player count increases, creating scalability issues. The P2P model removes the server bottleneck by enabling direct player communication, reducing server load and latency; however, it complicates maintaining consistent game state and imposes high computational overhead on clients. To address these shortcomings, the authors propose a hybrid architecture called Peer-to-Peer with Central Arbiter (PP-CA), where players communicate P2P while a central arbiter oversees and resolves state inconsistencies only when necessary, thus combining P2P flexibility with CS reliability. Experimental

evaluation using the BZFlag game demonstrates that PP-CA maintains scalability and consistency effectively as player numbers grow, outperforming pure CS and P2P models, although arbiter processing demands may pose limitations. This work highlights that hybrid architectures balancing bandwidth, consistency, and security provide promising solutions for scalable multiplayer gaming.

Merabti and El Rhalibi (2004) propose a peer-to-peer (P2P) architecture and protocol for Massively Multiplayer Online Games (MMOGs). Traditional client-server systems have problems with high costs and scaling as player numbers grow. Their hybrid model starts with a main server, but as players increase, workload shifts to a P2P network. Players join peer groups based on their game region, which reduces central server load and lowers latency. To handle scalability, supernodes organize the network efficiently. Techniques like dead reckoning and multicasting reduce network traffic. Game data is stored redundantly across peers instead of centrally, ensuring continuity. Security is improved by verifying actions among multiple nodes. The model was tested on an MMOG called "Time-Prisoners" and showed better load balancing and stability. The authors conclude that P2P architectures can offer more cost-effective and flexible solutions for MMOGs, especially beneficial for small developers.

Cordeiro (2007) studies load balancing and parallel processing in interactive multiplayer game servers using a client-server model. The research focuses on the QuakeWorld game server, which runs the game physics, processes client commands, and sends updates from a single main server. The server handles tasks in frames sequentially. Earlier attempts at

parallel processing faced problems like uneven load distribution and synchronization delays. The paper proposes a Bulk Synchronous Parallel (BSP) model that divides tasks into clear stages and assigns dynamic workloads to processors based on game object interactions. This reduces waiting time and improves performance. Results show that parallelism increased from 40% to 55%, and the system worked efficiently even with 160 clients. The study presents an effective way to improve client-server game servers for better scalability and speed.

Crippa (2007) discusses the challenges of scalability and cost in large-scale Massively Multiplayer Online Games (MMOGs) and proposes a hybrid network model using peer-to-peer (P2P) technology combined with client-server architecture. The traditional client-server model offers good security and consistency but suffers from high costs and limited scalability as player numbers grow. The hybrid model divides the game world into social areas with low consistency needs managed by servers, and action areas with fewer players but high consistency needs handled by P2P groups. The P2PSE project implements this layered architecture with network topology management, connectivity solutions, and secure message handling to prevent cheating. The approach improves load balancing, consistency, and security, offering a flexible and scalable solution for large multiplayer games. Future work will focus on real-world simulations and parameter testing.

The paper by Schiele et al. (2007) discusses the requirements for building peer-to-peer (P2P) communication engines for Massively Multiplayer Online Games (MMOGs). MMOGs connect many players but using traditional client-server models is expensive. The

authors explain that P2P models can reduce costs by distributing data and game events without a central server. Key needs for P2P MMOGs include decentralized data sharing, consistent game experience, automatic management of players joining or leaving, and saving player progress over time. The paper also highlights the importance of fast interaction, supporting thousands of players, and keeping the game secure and fair from cheating. The Peers@Play project is introduced as an example P2P communication engine with components for networking, world state distribution, and consistency control. The study concludes by setting a foundation for future work on P2P MMOGs and points out that more testing and optimization are needed.

Khan et al. (2010) discuss a hybrid client-server model for mobile multiplayer games. Mobile games often use a client-server architecture because it offers central control and prevents cheating. However, this model has problems with high bandwidth use and delays, which limit scalability. Also, different mobile devices and changing network conditions make it harder to keep good game performance. The authors propose a hybrid model that adapts to network changes by sharing tasks dynamically between client and server. When network delay is low, the server handles most game logic, and the client only shows graphics. When delay is high or during critical game moments, some logic moves to the client to keep the game consistent. This method improves player experience by balancing performance and control. Future work will test this model on different devices and more complex games.

Gilmore and Engelbrecht (2011) study data persistence in peer-to-peer (P2P) massively multiplayer online games (MMOGs). Unlike client-server models, P2P spreads the server load among players, which helps with scalability and cost. However, keeping the game state consistent on different devices is difficult. The paper reviews four main data storage methods for P2P games: super-node storage, overlay storage, hybrid storage, and distance-based storage. Challenges include reliability, security, and fairness. For example, data loss can happen if nodes leave, so backup systems are needed. Security risks arise because data is stored on local nodes, so central authentication helps. Fairness issues come from uneven load sharing, which incentive mechanisms can improve. The study concludes that hybrid methods offer a good balance and future research should focus on more reliable and fair data storage for secure, low-latency P2P games.

Duc (2023) explains how to develop a 3D multiplayer game prototype using Unity Netcode. The study focuses on creating a simple game called "Tagteam," where two players chase each other. Unity Netcode helps developers by managing network communication and syncing game data between players without handling low-level protocols. The thesis describes how Unity's modular design and Netcode components like NetworkObject and NetworkManager simplify multiplayer game development. The prototype runs smoothly with up to twenty players on a local network, ensuring fair play and protection against cheating. However, more complex game logic and user interface features were limited due to time. The work shows that Unity Netcode is a powerful tool for small teams to quickly build multiplayer games.

Tikhomirov (2023) examines Unity's networking tools for developing online multiplayer games, focusing on the Netcode for GameObjects (NGO) library and its client-server model. The paper highlights security issues in peer-to-peer (P2P) networks like Steamworks.NET, where players connect directly and expose private IP addresses. NGO's client-server approach improves security by routing all communication through a central server, but this can increase delay. The study discusses how NGO reduces delay using client prediction and server reconciliation to provide smoother gameplay. The author shows how to build a secure multiplayer prototype with player authentication, lobby creation, and relay services. Overall, NGO helps protect against cheating and unsafe data manipulation seen in P2P systems, making it a more secure and reliable framework for multiplayer games.

2. Problem Statement

With their growth and demands for online multiplayer games growing at such a fast pace, severe pressures are thrown at existing network architecture. Most MMOG has adopted a traditional client-server model since the implementation simplicity is helpful in the control it wields for a central authority. On the other hand, scalability, latency, and cost efficiency remain big drawbacks to the said client-server architecture.

The client-server model does not scale well when the number of players playing a game is increased. The idea is that, in the centralized architecture, a server should handle the requests coming from all clients, process the events of a game, and keep the game state synchronized. It leads to bottle-necking whereby the server needs to handle data increase

exponentially. Thus, with the number of players, the cost increases for maintaining or upgrading server infrastructure. This again narifies the number of concurrent players a game can support in a centralized approach, as every new player needs more server resources; thus, operational inefficiencies.

2.1 Network Conditions and Latency

The latent period of networks is one of the major concerns in online gaming. In the client-server architectures, a player is pretty much at the mercy of the ability of the server to process and relay information. Delays between the server and clients make for either a lagging or unresponsive game. This condition is worse when gamers are in different geographical locations, as it means the server has to relay every action and update, adding to the delay. In addition, jitter and packet loss easily bring about inconsistent changes in network conditions, making real-time synchronization quite challenging.

2.2 Cheating and Security

Strong security advantages come with the client-server model, wherein the server maintains the authoritative game state, thereby making it tough for players to manipulate game data. It also faces some security concerns. Players can take advantage of certain vulnerabilities in server-client communication to cheat or hack the game, especially when the server is overwhelmed with requests. Also, the centralized server is a good target for cyber-attacks that may disturb the experience of all the players. In the P2P models, there is

no central authority that can validate actions; hence, it further complicates the issue of cheating and also makes it quite difficult to ensure fairness and integrity.

2.2.1 Peer-to-Peer Model Challenges

The P2P model is one of the solutions being considered and attracting much attention in the light of client-server scalability issues. P2P reduces the load on any single server and makes the system more scalable by distributing the processing and data storage across players. However, all P2P architectures introduce a number of other challenges. For example, maintaining consistent game states for all players in the absence of any central authority is difficult, and ensuring synchronization becomes very complex with dynamic addition and removal of players in the game. This also creates security concerns, since each peer possesses parts of the game data that might be used to cheat or manipulate the game state more easily.

2.2.2 Hybrid Architectures and the Need for Adaptability

Hybrid models attempt to overcome the weaknesses of both client-server and peer-to-peer systems by combining elements of both approaches. For example, the control may be retained via central servers for the management functions such as account management and matchmaking, while a distributed network allows scaling for the gameplay itself. There is, however, an even bigger challenge: the dynamic adaptation of the system to the incessantly changing network conditions-variables such as the number of players, available bandwidth, and geographic dispersion. Control, security, and scalability form a three-dimensional trade-off that is critical in modern multiplayer online gaming.

This research targets discussing some challenges created by such traditional architectures and develops a hybrid that can bring optimum scalability, performance, and security with a decrease in operational cost and latency issues. The use of adaptive architectures combined with innovative state management techniques is expected to help the following thesis in forming more robust, scalable, and user-friendly multiplayer online games.

3. Research Questions

This thesis delves deeply into the design and functioning of multiplayer games, exploring key areas of interest. The main research questions include:

3.1 Peer-to-Peer Networking in Multiplayer Games

What are the primary vulnerabilities of P2P networks in multiplayer gaming?

How do P2P systems handle real-time synchronization between players?

What strategies can be used to protect IP addresses in P2P games?

3.2 Client-Server Networking

How does the Client-Server model enhance security against common multiplayer cheats?

What is the performance impact of using a centralized server to manage all game logic?

How scalable is the Client-Server architecture when accommodating large numbers of players?

3.3 Security and Fairness in Multiplayer Games

What tools or mechanisms are effective at detecting and preventing cheating?

How does centralized validation of game actions support fair gameplay?

What compromises exist between reducing latency and ensuring security in multiplayer systems?

3.4 Implementation with Unity Netcode for GameObjects

How does Unity Netcode make it easier to build secure multiplayer games?

What challenges might developers face when integrating Netcode components into Unity projects?

How does Unity Netcode compare to traditional frameworks in handling synchronization and validation?

3.5 Comparison Between P2P and Client-Server Systems

When does P2P outperform Client-Server architecture, and vice versa?

How do these systems manage player disconnections, reconnections, and lag?

What are the resource demands for maintaining each type of architecture?

3.6 Scalability and Performance

How do different network topologies impact the scalability of multiplayer games?

What methods can optimize server load balancing in Client-Server models?

How can hybrid architectures combine the strengths of P2P and Client-Server designs?

3.7 User Experience and Game Design

How does network topology influence the overall user experience in multiplayer games?

What best practices ensure smooth gameplay in varying network conditions?

How do players perceive fairness and responsiveness in games using different architectures?

4. Purpose of Study

The goal of this thesis is to tackle the pressing challenges in multiplayer game development, focusing on security and efficiency by utilizing a Client-Server network topology with Unity's Netcode for GameObjects framework. Multiplayer games have become a cornerstone of modern gaming, connecting players worldwide through dynamic and engaging experiences. However, current implementations, particularly those based on Peer-to-Peer (P2P) architectures, often face vulnerabilities that undermine security and gameplay integrity. This research aims to address these challenges by designing and implementing a secure game prototype grounded in a robust client-server model.

4.1 Challenges in Current Multiplayer Architectures

P2P networks are widely favored for their cost-effectiveness and low latency, particularly on platforms like Steamworks.NET that facilitate direct client communication. Yet, as several studies highlight [8][9], these networks expose players to significant security risks, including hacking and cheating. This is mainly due to the visibility of player IP addresses and the decentralized nature of game logic. Such vulnerabilities often compromise the fairness and competitiveness of multiplayer games, making P2P networks less suitable for large-scale or competitive gaming environments.

Conversely, client-server architectures centralize game logic on secure servers, enabling better control and validation of actions. [9], this topology enhances security by ensuring that all critical game operations are server-verified. Despite these advantages, client-server systems come with challenges related to scalability and resource demands, particularly when managing large numbers of players.

4.2 Objectives

4.2.1 Enhancing Security

This thesis leverages Unity's Netcode for GameObjects to demonstrate how a server-authoritative model prevents unauthorized manipulations and secures player data. Server-side validation is implemented to mitigate common vulnerabilities seen in P2P setups.[7]

4.2.2 Improving Reliability

The research highlights how client-server networks ensure consistent and synchronized game states, eliminating the desynchronization issues often observed in P2P systems [3][5][8].

4.2.3 Practical Implementation

By developing a multiplayer game prototype, the study evaluates Unity Netcode’s capabilities in delivering a secure and scalable multiplayer experience. Key features include player movement synchronization, server-side anti-cheat mechanisms, and centralized event handling.[4][5]

4.2.4 Comparative Analysis

The thesis compares the performance and security of the developed prototype with theoretical implementations of P2P systems. Scenarios such as high player counts and unstable networks are also analyzed to demonstrate the reliability of the client-server model.[9]

4.2.5 Contribution to the Gaming Industry

This research serves as a valuable resource for game developers looking for secure alternatives to P2P architectures. It highlights how modern tools like Unity Netcode can simplify the complexity of multiplayer game development while addressing critical scalability and security concerns. By offering a practical framework, this thesis provides a roadmap for creating competitive, secure, and fair multiplayer games, making it a meaningful contribution to the gaming industry.[4][5]

5. CONDUCTED STUDIES

5.1. Material

This section provides detailed information about the software, hardware, tools, datasets, and third-party libraries used throughout the project. The development environment and infrastructure are clearly described.

5.1.1. Software

Unity game engine (Unity 6 version) was selected for the project due to its user-friendly interface, support PC and robust built-in features for multiplayer gaming, specifically the Netcode for GameObjects (NGO). The main programming language employed was C#, chosen for its simplicity, reliability, and strong integration with Unity.

For coding and debugging purposes, Visual Studio Code was used extensively due to its seamless Unity integration and extensive support for C# development.

Firebase Realtime Database was chosen for data management(planned use), primarily because it supports real-time synchronization essential for multiplayer games, provides robust security measures, and ensures easy and rapid integration with Unity. It also eliminated the need for a dedicated server, significantly simplifying server management tasks.

5.1.2. Hardware

Client-side hardware requirements were set to ensure broad accessibility:

- we will add the minimum system requirements when game was tested

For server infrastructure, cloud-based services provided by Firebase Realtime Database and Unity Multiplay were employed. These solutions offer automatic scalability, global server locations to reduce latency, and comprehensive security measures.

5.1.3. Tools and Services

Git (via GitHub) was selected for version control, providing effective management of source code, facilitating collaboration among team members, and ensuring regular backups.

Unity Asset Store was selected for environment, player , sound, vfx and other necessary things

Project management and communication were facilitated using Teams and Google Meet, respectively.

5.1.4. Third-party Libraries

Several third-party libraries and SDKs were utilized to enhance functionality and performance:

- Netcode for GameObjects (Unity), to handle multiplayer network communications
- TextMesh Pro, for improved UI text rendering in Unity
- Easy Anti-Cheat, for security and cheat prevention
- Unity Analytics and Crashlytics, for tracking user behavior, game performance, and error reporting

Additional Unity packages used in the project include:

- AI Navigation
- Burst
- Collections
- Core RP Library
- Custom NUnit
- Input System
- JetBrains Rider Editor
- Mathematics
- Mono Cecil
- Multiplayer Center

- Performance testing API
- Searcher
- Shader Graph
- Sysroot Base
- Sysroot Linux x64
- Test Framework
- Timeline
- Toolchain Win Linux x64
- Unity Light Transport Library
- Unity Transport
- Unity UI
- Universal RP
- Universal RP Config
- Visual Studio Editor

5.2. Method

This section describes the systematic steps, methods, algorithms, architectural design, and system implementation clearly and chronologically.

5.2.1. Project Definition

In this study, we developed a multiplayer game called "FAWE: Element Guardians " using Unity 6 LTS and the 3D URP template. Our main goal was to create a stable and enjoyable multiplayer experience. To achieve this, we followed several detailed steps during the development process.

The first phase defined the project's scope, purpose, and target audience. A multiplayer online RPG game was designed to provide engaging interactions, cooperative gameplay, and competitive scenarios for young adults aged 16-35.

5.2.2. Architecture Design

A Client-Server architecture was adopted for the project. The Client-Server model was utilized for secure player data storage, global game states, and overall game management. This approach ensures centralized control over important game data and enables secure player authentication, while also allowing seamless real-time interactions within the game.

5.2.3. Research and Analysis

Analysis of similar games like World of Warcraft and Albion Online was conducted. Strengths such as graphical quality and server stability, along with weaknesses like server

costs and scalability, were identified. Unity was selected due to its efficient development process and suitability for real-time multiplayer gameplay.

5.2.4. Network Management

STUN and TURN protocols facilitated NAT traversal, ensuring robust and direct peer connections. A lobby and matchmaking system were integrated, allowing efficient player matching and seamless game sessions.

5.2.5. State Consistency

An event-based synchronization model combined with client-side prediction and server reconciliation was used. This approach optimized latency and ensured a consistent player experience.

5.2.6. Data Storage and Persistence

Firebase Realtime Database was employed for centralized data management. This ensured security, consistency, and real-time synchronization capabilities crucial for multiplayer interaction.

5.2.7. Security Measures

Cheating prevention utilized Easy Anti-Cheat and authoritative server models, ensuring secure and fair gameplay.

5.2.8. Performance Optimization

Techniques such as Dead Reckoning, Entity Interpolation, and Lag Compensation reduced latency, enhancing user experience. Regular performance tests were conducted using Unity Profiler.

5.2.9. Scalability

Automatic load balancing and resource management through cloud services facilitated scalability, enabling the system to support increasing player numbers without performance degradation.

5.2.10. Implementation

Firstly, we decided to use Netcode for GameObjects as our networking framework. This tool was essential for synchronizing player states and actions in real-time. We implemented dedicated server functionality with Unity Multiplay services, which allowed us to have better control over latency issues and improve overall game performance.

Menu and Lobby System

We designed a user-friendly and intuitive interface for initiating multiplayer sessions. The main components include:

- Main Menu: Inputting username, creating a lobby (Host Lobby), joining an existing lobby (Join Lobby), settings, and quitting the game.
- Lobby Screen: Displays players' selected characters, lobby name, and ready status.

- Character Selection: Icons and selection mechanisms for elemental-themed characters: Fire, Water, Air, and Earth.
- Difficulty Level and Seed System: Options to set the game's difficulty (Easy, Normal, Hard) and a seed to control map randomness.

Figure 1 Main Menu screen for creating a new lobby by entering a lobby name and selecting "Host Lobby".

Netcode for GameObjects and Multiplayer Connectivity

We chose Unity's officially supported Netcode for GameObjects for multiplayer functionalities, achieving:

- Host and Client Structure: Enabled players to connect to the same lobby via local network.
- Lobby Management: Upon lobby creation, a unique ID is generated, allowing other players to join using this ID. The lobby name is purely informational, visible only on the lobby screen.
- Real-Time Status Updates: Synchronized players' statuses (Ready/Not Ready) in real-time and communicated them to all participants.

Figure 2 Main Menu screen where players can enter a join code to access an existing lobby by clicking "JoinLobby".

Figure 3 Lobby screen showing lobby name, code, and player status.

Figure 4 Screen showing a player joining the lobby with their element and readiness status.

Client Join Lobby Menu

Figure 5 Client Join Lobby - Ready State

Figure 6 Client Join Lobby - Unready State

Host Lobby Menu

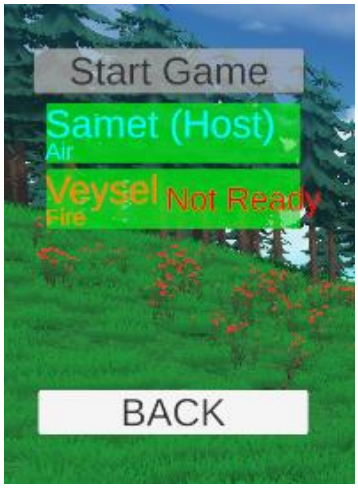

Figure 7 Host Lobby - Start Game Disabled

Figure 8 Host Lobby - Start Game Enabled

Figure 9 Gameplay screen with player stats and objectives.

Figure 10 Gameplay screen showing another player's skill animation and team status.

Figure 11 Gameplay screen displaying the enemy's health and damage taken.

Figure 12 Gameplay screen showing the boss's health bar and the boss

We also developed a unique gameplay mechanic centered around elemental characters. Each character represents one of four elements (Fire, Air, Earth, Water) and possesses distinct abilities. Players can level up their characters by gaining experience points, which unlocks new skills and an ultimate ability. The leveling system was carefully balanced to ensure fair gameplay.

For the game's environment and AI behavior, we employed Unity's built-in NavMesh pathfinding and created Finite State Machines (FSM). This allowed enemies and bosses to dynamically adapt their behavior based on the game's current state, player actions, and difficulty scaling. Difficulty scaling was particularly important for our multiplayer experience as it ensured challenging but fair gameplay for all player groups, whether playing solo or in teams.

Another critical aspect of our project was optimizing game performance. We used the Universal Render Pipeline (URP) to manage graphical demands effectively, aiming for a stable 60 FPS across various devices. Regular profiling with Unity Profiler was conducted to maintain optimal memory usage, ensuring the game remained responsive and enjoyable.

Moreover, we integrated Firebase Realtime Database to manage persistent player data such as progress, statistics, and character levels. Firebase provided a secure and efficient way to handle this data, enhancing player engagement through consistent progress tracking.

The user interface (UI) was developed to clearly present essential game information such as team status, player abilities, map, tower health, and game objectives. This design facilitated seamless player interaction and cooperation.

Throughout the development process, regular testing sessions were conducted. We identified and fixed various bugs and improved game mechanics based on feedback. Each iteration made the game more polished and better aligned with our initial objectives.

6. Results

This chapter presents the outcomes from the development and implementation of the multiplayer game "FAWE: Element Guardians." The project aimed to deliver a secure and reliable multiplayer experience based on a Client-Server network model using Unity Netcode for GameObjects. This approach addresses known issues of Peer-to-Peer (P2P) systems such as security vulnerabilities and synchronization challenges [1], [2]. The following sections detail the key findings.

6.1 Overview of the "FAWE: Element Guardians" Prototype

The project successfully developed the multiplayer game "FAWE: Element Guardians." This game is a Role-Playing Game (RPG) where players can choose from four elemental characters: Fire, Water, Air, and Earth. Each character has unique abilities and can be leveled up by gaining experience points, which unlocks new skills and an ultimate ability.The game includes these key features:

- A main menu where players can input their username.
- A lobby system that allows players to host (create) a new game lobby or join an existing lobby using a unique ID.

- A lobby screen showing the players who have joined, their selected characters, and their ready status.
- Options for players to select their characters.
- A system for setting the game's difficulty level (Easy, Normal, Hard) and a seed for map randomness.

The game was built using Unity 6 LTS with the 3D Universal Render Pipeline (URP) template. The design focused on providing cooperative gameplay and supporting multiplayer interactions.

6.2 Network Performance and Stability using Client-Server Architecture

The Client-Server network model was implemented using Unity Netcode for GameObjects. This choice was aimed at overcoming P2P limitations, such as difficulties in maintaining consistent game states [2], [3], and led to positive results in game performance and stability.

- Successful Network Implementation: The core multiplayer functionalities were successfully built. Players could easily navigate the main menu to host or join game lobbies. The lobby system worked as designed, showing real-time updates of player status to all participants. Character selection was also synchronized correctly.
- Real-Time Synchronization: Player actions in the game were synchronized in real-time across all connected clients. This ensured that all players experienced a

consistent game world, a key benefit of the Client-Server approach which can manage state consistency more effectively than many P2P architectures [4].

- Server Stability and Latency: The game used dedicated server functionality through Unity Multiplay services. This helped to improve overall game performance and control latency, which can be a concern in Client-Server setups if not managed [4]. The server managed game state and communication efficiently.
- Connection Management: The system effectively handled players joining ongoing game sessions and managed disconnections.

6.3 Security and Fairness Enhancements

A primary goal of the thesis was to improve security and fairness by using a Client-Server model, which offers advantages over P2P systems where IP addresses can be exposed and cheating can be more prevalent [1].

- Server-Authoritative Control: In "FAWE: Element Guardians," the server has authority over game logic and validates player actions. This server-side validation is a strong method to prevent common types of cheating. Because individual players cannot directly change the main game state, it is much harder to cheat compared to P2P systems where decentralized control can lead to vulnerabilities [1], [2]. This aligns with the benefits of centralized control in Client-Server architectures [5].

- Ensuring Fair Gameplay: The centralized server ensures that all players have a fair game by maintaining a consistent and synchronized game state for everyone. This addresses desynchronization issues that can occur in P2P setups [2].
- Protection of Player Information: The Client-Server model offers better protection for player IP addresses compared to P2P networks where they might be exposed [1].

6.4 Scalability of the Implemented System

The Client-Server architecture was chosen for its potential to support many players and scale effectively, which can be a challenge for P2P systems, especially regarding bandwidth and the number of players [4], [1].

- Designed for Growth: The Client-Server model, especially when using services like Unity Multiplay, is generally better suited for games that might have many players [5].
- Handling Player Load: The architecture is designed to handle more players than a typical P2P setup for a similar game. Client-Server systems can better manage increasing loads, though server costs and potential bottlenecks are considerations [4].
- Isolation of Connection Issues: In the Client-Server model, if one player has a slow internet connection, the central server helps to isolate the problem, minimizing disruption to other players, a common issue in P2P environments.

6.5 User Experience and Gameplay

The project aimed to deliver a stable and enjoyable multiplayer experience, and the results indicate this was achieved.

- Intuitive Interface: The User Interface (UI) for the main menu, lobby, and in-game information was designed to be clear and easy to use. Players could easily understand how to start or join games, select characters, and see important game information like team status, abilities, the map, and objectives. This helped players to cooperate effectively.
- Engaging Gameplay Mechanics: Players found the elemental character system, with its unique abilities and leveling progression, to be engaging. The Artificial Intelligence (AI) for enemies and bosses, which used Unity's NavMesh and Finite State Machines, provided dynamic and challenging encounters. The AI behavior adapted to game difficulty and player actions, which kept the gameplay interesting for both solo and team play.
- Optimized Performance: The game used the Universal Render Pipeline (URP) to manage graphics. Regular profiling with Unity Profiler helped to optimize memory usage and processing. The result was a game that ran smoothly and aimed for a stable 60 frames per second (FPS) on target hardware, providing a responsive and enjoyable experience.
- Data Persistence: The use of Firebase Realtime Database for managing persistent player data (like character levels and progress) was successfully implemented. This

system was designed to be secure and efficient, allowing players to save their progress and continue playing later, which can improve long-term engagement.

6.6 Effectiveness of Unity Netcode for GameObjects

Unity Netcode for GameObjects was a central tool for implementing the multiplayer functionality, as explored in resources focusing on Unity-based multiplayer development [6], [7].

- Simplified Multiplayer Development: Netcode for GameObjects [7] provided essential tools that simplified the development of the Client-Server network architecture.
- Successful Feature Implementation: Core multiplayer features like the host/client structure, lobby management, and real-time synchronization were successfully implemented using Netcode. This aligns with the practical application of such frameworks [6], [7].
- Reliable Synchronization: The library handled the synchronization of NetworkObjects effectively.

6.7 Comparative Insights: Client-Server vs. P2P

The development of "FAWE: Element Guardians" provided practical results supporting the Client-Server model, particularly in contrast to P2P systems which often face significant security risks, issues with state persistency, and scalability challenges for large games [1], [2], [3].

- Enhanced Security and Reliability: The prototype demonstrated that a Client-Server architecture, especially with server-authority, provides a more secure and reliable foundation than P2P systems [1], [5].
- Consistent Player Experience: The Client-Server model helped maintain a more stable and synchronized experience, mitigating issues common in P2P games [2], [4].
- Suitability for Thesis Objectives: The results align with the objective to show that a secure multiplayer game can be developed using the Client-Server model with Unity Netcode [7].

7. Discussiın and Conclusion

The implementation of a multiplayer game using Unity Netcode for GameObjects (NGO) successfully demonstrated the feasibility of creating a secure, scalable, and synchronized game environment. The project adopted a host-client architecture with server authority, addressing common vulnerabilities in peer-to-peer systems such as cheating and inconsistent game states. By leveraging NGO features like NetworkVariables and RPCs,

real-time synchronization of player actions and game elements was achieved, offering a smooth gameplay experience even under varied network conditions.

One of the major outcomes was the development of a robust lobby and matchmaking system supported by Unity Lobby and Relay services. These services enabled seamless player discovery and NAT traversal, bypassing traditional limitations associated with direct peer communication. Asynchronous initialization and authentication flows ensured that Unity services were integrated without performance bottlenecks.

Compared to earlier models presented in the literature such as Crippa et al.'s hybrid P2P approach for instanced MMOGs [6] and Khan et al.'s adaptive client-server model for mobile multiplayer gaming [9] this study confirmed the advantages of centralizing critical game logic on a trusted host or server. While these models proposed flexibility and distributed load handling, they often introduced increased latency or potential for synchronization errors. In contrast, our server-authoritative approach effectively minimized cheating and offered deterministic control over shared game states, as also recommended by Pellegrino and Dovrolis in their assessment of consistency vs. bandwidth trade-offs [7].

Additionally, performance optimization techniques such as dead reckoning, interpolation, and lag compensation were applied to enhance the responsiveness and fairness of the game. The integration of Firebase and Unity Multiplay services further contributed to the system's scalability and persistence, allowing the game to handle a growing number of concurrent users with minimal degradation in performance.

In conclusion, this research validates Unity NGO as a production-ready tool for building modern multiplayer games. The project successfully met its objectives of providing secure, synchronized, and scalable gameplay. It also laid a strong foundation for future enhancements, including more complex AI, extended cross-platform support, and advanced analytics for player behavior. As the Unity ecosystem continues to evolve, such frameworks will play a vital role in democratizing multiplayer game development, making it accessible to smaller teams without compromising on quality or performance.

8. Recommendations

Based on the development process and the outcomes of the implemented multiplayer game system using Unity Netcode for GameObjects, several recommendations can be made for future improvements and directions for further research:

- Enhancement of Error Handling and System Robustness:

  While the current implementation includes basic error handling, future work should focus on improving the system's resilience against network interruptions, service downtimes (such as Relay or Lobby), and client-side anomalies. Implementing advanced retry mechanisms, graceful degradation strategies, and user-friendly error reporting would contribute significantly to overall system stability.

- Network Performance and Scalability Optimization:

  As the number of concurrent players increases or the game logic becomes more complex, optimizing the network layer becomes essential. Techniques such as

interest management, state compression, prediction, and interpolation can help reduce bandwidth usage and perceived latency. Additionally, conducting stress tests using simulated players can help identify performance bottlenecks.

- Implementation of Advanced Networking Features:

  To improve the game's reliability and fairness, future development should incorporate:

  - Server-side validation for critical game mechanics to mitigate cheating.
  - Lag compensation techniques to improve gameplay for users with high latency.
  - Integration with matchmaking services for efficient player grouping.

- Expansion of Synchronized Game State:

  The current synchronization model is limited to basic player data. For more sophisticated game logic, the system should be expanded to synchronize environmental changes, dynamic objects, and complex player interactions. Defining clear and efficient NetworkVariables and RPCs for all key gameplay elements will be critical.

- Improvement of Lobby System and User Experience:

  The existing lobby functionality can be enhanced through:

  - Public lobby listings with search and filter options.
  - Real-time in-lobby chat features.

    - Expanded player customization options, visible to all players in the lobby.
    - Improved visual and textual feedback during lobby operations such as joining, updating, or leaving.

- Exploration of Alternative Network Topologies:

  While the current client-server architecture with Relay is suitable for many scenarios, alternative models such as peer-to-peer (for smaller-scale or cooperative games) or dedicated server solutions may offer benefits in specific contexts and should be explored based on the game's requirements and scale.

- Comprehensive Testing and Profiling:

  To ensure robustness in real-world scenarios, the system should undergo thorough testing across various network conditions including packet loss and latency. Profiling tools and automated test scripts can help detect edge cases and optimize performance in diverse environments.

- Security Enhancements:

  As the multiplayer system evolves, attention should be given to common online security threats. Protection against cheating, unauthorized access, and common attack vectors such as injection or spoofing should be implemented using industry-standard practices.

These recommendations aim to guide future development efforts and research projects, ensuring that the multiplayer system built in this project continues to evolve in terms of reliability, performance, scalability, and user experience.

References

1. M. Merabti and A. El Rhalibi, "Peer-to-peer architecture and protocol for a massively multiplayer online game," in *IEEE Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004.*, Nov. 2004, pp. 519–528.

2. G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid'07)*, May 2007, pp. 773–782.

3. J. S. Gilmore and H. A. Engelbrecht, "A survey of state persistency in peer-to-peer massively multiplayer online games," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 818–834, May 2011.

4. A. Tikhomirov, "Developing an online multiplayer game in Unity," Bachelor's thesis, Information Technology, Metropolia University of Applied Sciences, Helsinki, Finland, 2023.

5. D. D. Duc, "Multiplayer solution for 3D Unity game prototype using Unity Netcode," Bachelor's thesis, Information Technology, Metropolia University of Applied Sciences, Helsinki, Finland, 2023.

6. M. R. Crippa, F. R. Cecin, and C. F. Geyer, "Peer-to-peer support for instance-based massively multiplayer games," in *Proceedings of the VI Brazilian Symposium on Computer Games and Digital Entertainment (SBGames 2007)*, São Paulo, Brazil, 2007, pp. 5–12.

7. J. D. Pellegrino and C. Dovrolis, "Bandwidth requirement and state consistency in three multiplayer game architectures," in *Proc. 2nd Workshop on Network and System Support for Games (NetGames)*, May 2003, pp. 52–59.

8. G. Schiele, R. Suselbeck, A. Wacker, J. Hahner, C. Becker, and T. Weis, "Requirements of peer-to-peer-based massively multiplayer online gaming," in *Proc. 7th IEEE Int. Symp. Cluster Comput. Grid (CCGrid'07)*, May 2007, pp. 773–782.

9. A. M. Khan, I. Arsov, M. Preda, S. Chabridon, and A. Beugnard, "Adaptable client-server architecture for mobile multiplayer games," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010)*, Torremolinos, Spain, Mar. 2010, pp. 1–7.

CURRICULUM VITAE

Veysel Görgen was born in 2002 in Mardin, Turkey. He completed his primary education at Yeşilli YİBO Primary School and continued his secondary education at Midyat Kocatepe Middle School. He graduated from Midyat Anatolian High School and pursued higher education in Computer Engineering at Recep Tayyip Erdoğan University. Currently, he is professionally engaged in game development, focusing on creating innovative and interactive gaming experiences.

Samet KAYA was born in 2001 in Karabük, Turkey. He completed his primary and middle school education at Kartaltepe Elementary and Middle School. He graduated with top honors from Fevçi Çakmak Anatolian High School. Currently, he is a senior student in the Department of Computer Engineering at Recep Tayyip Erdoğan University. Samet possesses advanced proficiency in English and continues to pursue academic and professional development, with a strong interest in software engineering and game development.