Distributed Systems in Multiplayer Games

Lizeth Valdovinos Rodriguez
027798535
College of Engineering
California State University, Long Beach
Lizeth.ValdovinosRodriguez01@student.csulb.edu

Megan Kang
027625427
College of Engineering
California State University, Long Beach
megan.kang@student.csulb.edu

***Abstract*– The project explores the integration of distributed systems in the development of an online multiplayer game using the Godot game engine. Distributed systems refer to a network of interconnected computers that collaborate to achieve a common goal, and in the context of multiplayer games, they play a pivotal role in facilitating seamless player interactions across different locations. We aimed to implement a functional multiplayer game using Godot, highlighting the significance of distributed systems in tackling the complexities of online games. We chose Godot as our game engine due to the controversy of Unity and also having previous knowledge of the game engine. The engine offers flexibility, and support for networking features and offers a built-in tool for handling multiplayer functionalities, making it suitable for our distributed systems-oriented project. Through Godot's high-level networking API, we established a client-server architecture where one player acts as the server, hosting the authoritative game state, and other players connect as clients. We worked to emphasize the significance of distributed systems in game development and the challenges that come with it.**

I. Background

The evolution of online multiplayer gaming has paved the way for a new era of interconnected experiences by bringing players from diverse locations together. In the realm of multiplayer online games, a wide range of distributed systems architectures play important roles in the making. First, we need to get a better understanding of various architectures in order to find what best fits our goals. The client-server architecture centralizes control, facilitating efficient management of game state and logic, ensuring fairness and consistency. In contrast, peer-to-peer models decentralize authority, enabling direct communication among clients and potentially reducing latency. Hybrid architectures combine both of these concepts, creating a balance between centralized control and decentralized interactions. Furthermore, microservice architecture breaks down game functionalities into modular services, fostering scalability and maintainability. The Entity-Component-System (ECS) model efficiently processes game entities, enhancing scalability and supporting modular design and federated architectures involves multiple independent servers collaborating to create a broader network, promoting scalability and fault tolerance. Edge computing architectures distribute game logic to nodes closer to players, minimizing latency for real-time interactions. The Colyseus architecture specializes in distributing gameplay state with low-latency constraints. Collectively, these architectures contribute to the scalability, fault tolerance, low-latency interactions, flexibility and decentralization crucial for delivering immersive multiplayer gaming experiences. Choosing between these architectures is dependent on how you want your game to function, how large the server is, and the type of game you are making.

II. Introduction

As multiplayer games start hosting to a larger scale (such as mass online multiplayer games (MMOs), or various games allowing personal servers), the demand for robust and scalable distributed system architectures becomes increasingly significant in game development. Drawing inspiration from the

various distributed architectures in gaming and leveraging insights from established networking protocols, we set out to design and implement a distributed system tailored for a 2D multiplayer game using the Godot game engine. This project not only serves as a practical exploration of distributed architecture but also encapsulates our research on server based multiplayer games. We navigated through topics such as managing connections, remote procedure calls, channels, and the nuances of HTTP requests. The sections ahead will discuss our architectural blueprint that combines client-server architecture, Godot, and performance optimization. We will also discuss our process in making our game, and how we incorporated a client-server structure to allow players to host and join a server within the game. This research project serves to connect the academic course material discussed and real-world application, emphasizing the importance of distributed systems in the history of online multiplayer gaming as well as its future.

III. Network Architecture and Game Design

For our game, we decided to go with a client-server architecture, as it is one of the most popular architectures used in multiplayer games and offers a simple structure that provides centralized control and scalability for future use if we decide to expand on the game at a later time.

As mentioned previously, our game was made through Godot and was made to follow a simple 2D pixelated platformer game design. The game currently only contains one level and is devoid of any objectives or enemies for the players. The multiplayer hosting and joining server is its primary engagement, followed by a simple level design to allow players to interact in the game and challenge each other with the map.

Our process when making the game started off by first making a simple 2D player with animated movements including left, right, jump, double jump, and idle. All of which were achieved through using free art asset sprites. Following the player, we then moved on to building a level, which was also made through free online art assets and the TileMap tool in Godot to help us incorporate the physics to each of the environment pixels. After both player and level design was complete we tested to ensure that the physics and collisions of the two worked. Our final phase of the process then encompassed implementing all the code scripts. To make the game character playable by the user we included a script that controlled the physics process and properly handled the inputs (W, A, S, D, and Space) for movement with their corresponding animations. In the script, we also made sure to include the proper checks to ensure the player is controlled by their assigned user. The second script we wrote was for the level map, which made the multiplayer functionality including the server creation, player connection, and disconnection. Both scripts were written in GDScript which is Godot's built-in language that is syntactically similar to Python.

IV. Evaluation

Throughout the process of making the game, we performed several functionality tests to ensure that the main components of our game like controls, environment interactions, and the multiplayer functionally worked successfully. These tests for most of our development went by smoothly, but we faced the most challenges when we initially incorporated the multiplayer functionality. Our first attempt was to make our game a Local Area Network multiplayer that automatically searched for nearby devices hosting on an IP address. This attempt proved to be challenging as most of the components and built-in functions to make this feature happen were outdated, and were featured in Godot's previous version, Godot 3. In turn, to accommodate this challenge for our second attempt, we simplified our design and scripts to use updated Godot 4 functions to focus on setting up a server-client architecture on a local machine. This second attempt utilized Godots Multiplayer Spawner and Synchronizer nodes to synchronize all properties from the multiplayer authority, better known as the main player, to the other remote peers/players. Utilizing these nodes made this second attempt successful, but we wanted to attempt to improve synchronization by replacing the Synchronizer node with Remote procedure calls (RPC). Transitioning to RPC would've provided more precise control over what actions were being synchronized within the multiplayer game, but upon

implementation, we ran into more challenges regarding lag compensation. Transferring data through RPC can be reliable or unreliable, a reliable transfer mode is slow but has guaranteed delivery to its peers and unreliable is much faster but doesn't guarantee delivery. For our use, specifically for transferring peer player positions, we needed to convert to using an unreliable transfer mode in the RPC function that updated player movement. However, implementing this specific function resulted in a setback as it introduced latency issues, causing any joining users to experience delays in receiving their player movement or any other peer players. As a response to this delay, we reverted to our previous successful attempt to ensure smoother gameplay and connections.

V. Summary and Future Work

In conclusion, this research project emphasizes the use of distributed systems with the practical domain of multiplayer game development using Godot. By integrating client-server architecture with our 2D games, we've established a foundation of creating a scalable and responsive multiplayer gaming experience. Within online multiplayer gaming, the architecture supporting these experiences must evolve to ensure seamless interactions among thousands of players. Our venture into creating a distributed system for a 2D game not only underscores the technical components involved but also highlights the transformative impact on the overall gameplay. The reason distributed systems play such a crucial role in the gaming industry lies in their ability to optimize resource utilization, enhance reliability, and provide a more inclusive gaming experience. Distributing the computational load among multiple nodes prevents a single point of failure and reduces the risk of server crashes. It also facilitates geographical distribution, allowing players from different parts of the world to connect and interact seamlessly. Our main takeaways include learning about real-time synchronization, managing flow of information between the server and clients to maintain a consistent game state. We learned about techniques such as interpolation and extrapolation to mitigate the impact of latency and ensure smooth gameplay. We gained an understanding of mitigating latency and techniques such as client-side prediction and lag compensation to provide players with a responsive gaming experience. We attempted to achieve data consistency across distributed nodes as data is shared and synchronized. We focused on ensuring that all players would observe a synchronized view of the game world. Finally, we plan to further take into consideration the security in our client-server model and make sure the server maintains control over the game and prevents potential exploits or cheating by clients.

VI. References and Citations

[1] AyeshwerySamarakoon, "Architectures in Distributed System," *Medium*, Oct. 28, 2023. https://medium.com/@ayeshwery/architectures-in-distributed-system-b2ace2fca6bb

[2] A. Bharambe, J. Pang, and S. Seshan, "A Distributed Architecture for Interactive Multiplayer Games," 2005. Available: https://www.cs.cmu.edu/~ashu/papers/cmu-cs-05-112.pdf

[3] S. Rad, "Designing a Distributed System for an Online Multiplayer Game — Architecture (Part 3)," *Medium*, Apr. 02, 2022. https://theredrad.medium.com/designing-a-distributed-system-for-an-online-multiplayer-game-architecture-part-3-f9483ebbe5ac

[4]"High-level multiplayer," *Godot Engine documentation*. https://docs.godotengine.org/en/stable/tutorials/networking/high_level_multiplayer.html