REPUBLIC OF TURKEY

YILDIZ TECHNICAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

SERVER AUTHORITATIVE STRATEGY GAME

19011023 — Gazi Muhammet Canbolat

SENIOR PROJECT

Advisor

Assist. Prof. Dr. Oğuz ALTUN

Haziran, 2023

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor, Assist. Prof. Dr. Oğuz ALTUN, for his unwavering support during the project acceptance and throughout the process.

Gazi Muhammet Canbolat

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

OS	Operating System
GB	Gigabyte
MB	Megabyte
RAM	Random Access Memory
HDD	Hard Disk Drive
GPU	Graphics Proccessing Unit

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

SERVER AUTHORITATIVE STRATEGY GAME

Gazi Muhammet Canbolat

Department of Computer Engineering
Senior Project

Advisor: Assist. Prof. Dr. Oğuz ALTUN

This project is a multiplayer turn-based strategy game. The project's goal is to make sure gamers can play the game safely. Docker Desktop was used on the server side to build a server infrastructure. The server infrastructure was built using the game server engine Nakama. The Unity game engine was used to create the client application. The project has security features that enable users to correctly authenticate themselves and limit game access to authorized users. Additionally, by verifying the validity and proper processing of the data that participants send, unfair advantages among players are avoided.

Each player in the game has the option to place their colored-marked soldiers on the squares. Each player places a soldier on the board one at a time to begin the game, trying to spread their soldiers to other players' soldiers and touch them. A soldier must touch another one in any direction in order to spread. Depending on the capacity of the square in which it is located on the board, each soldier creates more soldiers. The players' objectives are to quickly deploy their armies throughout the board and remove their rivals' soldiers from it. The game ends with all soldiers turning into one player's colors.

Keywords: Multiplayer, turn-based, strategy game, identity verification, authorized players, game server engine, Docker Desktop, Unity game engine, Nakama, soldiers, squares, spreading

ÖZET

SERVER AUTHORITATIVE STRATEJİ OYUNU

Gazi Muhammet Canbolat

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Dr. Ögr. Üyesi Oğuz ALTUN

Bu proje, çok oyunculu bir sıra tabanlı strateji oyunudur. Projenin amacı, oyuncuların güvenli bir şekilde oyunun keyfini çıkarmalarını sağlamaktır. Bu doğrultuda, sunucu tarafında Docker Desktop kullanılarak bir sunucu altyapısı oluşturulmuştur. Sunucu altyapısı için Nakama adlı bir oyun sunucu motoru kullanılmıştır. İstemci uygulaması, Unity oyun motoru kullanılarak tasarlanmıştır. Proje, oyuncuların doğru bir şekilde kimlik doğrulaması yapmasını ve sadece yetkilendirilmiş oyuncuların oyuna katılmasını sağlayacak güvenlik önlemleri içermektedir. Ayrıca, oyuncuların gönderdiği verilerin doğrulanması ve doğru işlenmesi sağlanarak, oyuncular arasında haksız avantaj elde etmeleri engellenmektedir.

Oyunda, her oyuncu kendi renkleriyle işaretlediği askerleri, karelerin üzerine yerleştirme imkanı vardır. Oyun başladığında, her oyuncu sırayla kendi askerlerini tahtaya yerleştirir ve amaçları, kendi askerlerini diğer oyuncunun askerleriyle temas ettirmek ve askerlerin diğer askerlere doğru yayılmasını sağlamaktır. Bir askerin yayılabilmesi için, herhangi bir yönde bir adet diğer askerle temas etmesi gerekir. Her bir asker, tahtada bulunduğu karenin kapasitesine bağlı olarak daha fazla asker üretir. Oyuncuların hedefi, kendi askerlerini mümkün olduğunca hızlı bir şekilde yayarak rakip askerlerini tahtadan silmektir. Oyun, tüm askerlerin bir oyuncunun renklerine dönüşmesiyle sonlanır.

Anahtar Kelimeler: Çok oyunculu, sıra tabanlı, strateji oyunu, güvenlik, oyun sunucu motoru, Docker Desktop, Unity oyun motoru, nakama, askerler, kareler, yayılma

1 Introduction

The goal of this project is to create a server-authoritative, tile-based, turn-based, and strategic game. The created game resembles "Chain Reaction," but it seeks to provide players a fresh experience by having a different theme and gameplay. By placing more of our own soldiers on tiles, we will try to overwhelm the opposing forces in this game. Players in this game will have the ability to dynamically change their tactics in real-time due to the effects of each move made on the board.

By using recent game development technologies like the server-authoritative framework, this project also wants to deliver a fair gaming environment. This framework has the benefit of preventing injustice brought on by variations in player hardware or connection quality, allowing for level playing field competition.

In conclusion, the goal of this project is to create a strategy game that provides a fun and fair gaming environment. Modern game development elements and technology will be used with this goal in mind to give players a fresh experience.

2 Preliminary Examination

This game offers a highly immersive experience for strategy game players. By placing more of our own soldiers on squares, we may defeat the opposing soldiers in the game. Each player can place a number of their soldiers on various board squares throughout each turn in this turn-based game. How swiftly and effectively our soldiers move will depend on their positioning.

Controlling our soldiers while attacking enemy soldiers in the appropriate manner is the main goal of the game. Our soldiers can capture the squares they are on as well as adjacent squares as they travel across the board, spawning more soldiers there. By increasing in number and occupying more squares, our forces will strive to conquer the enemy's territory in this manner.

The game's tile-based design is one of its key characteristics. Players can deploy their soldiers and opponent soldiers on the square-shaped board. Additionally, players can immediately alter their strategy due to the effects of each move on the board.

The server-authoritative structure of the game is an additional interesting feature. By removing variations in hardware and connection speeds across players, this structure guarantees an equitable gaming environment. All actions in the game are carried out via the server. This ensures that the game is played fairly and stops players from getting undue advantages.

In conclusion, the created game provides a thrilling experience for fans of strategy games. The turn-based gameplay, server-authoritative structure, and tile-based organization allow players to have fun while employing their strategies in an equitable setting.

The general game rules:

Each participant in the game can deploy their own colored soldiers on the squares. Each square can hold only one soldier. Each player takes turns putting their men on the board as the game begins. The players' objective is to assist the spread of soldiers toward other soldiers by bringing their own soldiers into contact with the other player's soldiers.

A soldier must come into contact with another soldier in any direction for it to spread. The soldier who was contacted might similarly come into contact with another soldier. This causes a chain reaction as soldiers move around the board.

Depending on how much space each soldier occupies in the square, they each spawn more soldiers. For instance, three troops on a square will result in the creation of three other soldiers of the same hue. As a result, armies might grow over time and cover more ground on the board.

The players' goals are to quickly distribute their own soldiers over the board and remove all of their rivals' soldiers from the board. When one player's colors are represented by every army on the board, the game is over.

3
Feasibility

3.1 Labor and Time Feasibility

Week	Tasks Completed
1	Research on turn-based strategy games for study purposes.
2	Research on server-authoritative structure.
3	Determining the game that aligns with the suitable technologies to be used.
4	Researching the server infrastructure to be used in the game.
5	Creating the game's server infrastructure using the determined server infrastructure.
6	Designing the game.
7	Implementing the necessary mechanics of the game.
8	Designing the required interfaces of the game.
9	Rectifying any deficiencies and conducting a general quality check.

Table 3.1 Work-Time Schedule

The labor and time feasibility of the project is as shown in the table 3.1.

3.2 Technical Feasibility

3.2.1 Software Feasibility

The game was created using the Unity 3D game engine and the Nakama server architecture. Popular game engine Unity 3D offers high-performance graphics and tools to game makers. On the other side, Nakama is a server architecture that works with Unity 3D and is made for real-time multiplayer games. This architecture is built to scale easily and deliver great performance.

3.2.2 Hardware Feasibility

The game's hardware requirements are kept to a minimum. The game is made to function on a typical computer. Two gigabytes of RAM, a dual-core processor, and a graphics card that supports DirectX 11 are the absolute minimum requirements for the game to operate. As a result, it is anticipated that a large audience would be able to play the game without experiencing any hardware problems.

3.3 Legal Feasibility

The purpose of the project is to provide a new experience to players who enjoy military strategy games. The application does not collect personal information and will not cause any harm to users. The software applications we employ are authorized and freely usable. The project is therefore seen as being compliant with the law. The usage of Unity and Nakama applications is also not subject to any legal restrictions. Both programs have been licensed throughout the development process and are frequently used by developers. The official websites of both programs contain comprehensive information regarding the licenses and terms of usage. As a result, it is anticipated that using these applications within the parameters of the project will not provide any legal challenges.

3.4 Economic Feasibility

The student versions of the Unity and Nakama platforms were utilized during the project's development, so no licensing fees were incurred. There were no costs associated with server hosting because Docker Desktop was used as the local server infrastructure.

4
System Analysis

4.1 General Objectives and Goals of the Project

By making contact with enemy soldiers and rendering them ineffective, players are able to spread their own soldiers across the playing field. The objective of the game is to quickly change every soldier to your own color in order to win.

4.2 User Needs and Requirements

Multiple players can play the game simultaneously. To play the game, users must have an internet connection. The game is designed to function on the Windows operating system.

4.3 Hardware and Software Requirements

The Unity 3D game engine is used to create the game.[1]

The Nakama server infrastructure is used to build the game's server side. Docker Desktop is used to run the server side.

4.4 System Performance Metrics

Game performance will be influenced by hardware requirements like the server-side processor and memory. As more players join the game at once, the server-side performance of the game may suffer accordingly. The hardware and internet connection of the device have a direct impact on the client-side performance of the game.

4.5 Analysis of Existing Systems

Although there are games that are similar to this one already, the overall objectives and gameplay mechanics are different. The server infrastructure used to build the game's server side is known as Nakama. Through its server authoritative structure, this server infrastructure offers significant benefits in terms of security.

4.6 Security Requirements

A server authoritative structure is used in the game's design on the server side. This enables the server to implement security measures against player actions and permits the validation of data sent by players.

To play the game, player authentication is necessary. As a result, cheating and unauthorized access are prevented and only authorized players are allowed to join the game.

Passwords and other private data are securely kept and shielded from unauthorized access. There are safeguards in place to protect these delicate details.

All player communication is encrypted for secure transmission. This makes sure that any messages sent between players are secure and can't be seen by unauthorized parties.

5 System Design

5.1 Software Design

5.1.1 Programming Languages

For data sharing and the development of functionality across several components, the project uses the programming languages C# and TypeScript. While TypeScript is recommended for server-side programming, the Unity game engine uses C#.

5.1.2 Database Management

PostgreSQL is used as the database management system to store user information and game data. PostgreSQL is a powerful and feature-rich relational database management system.

5.1.3 Server Infrastructure

The project uses the game server infrastructure provided by Nakama, a game server engine. Nakama was created especially for multiplayer gaming. Using Docker Desktop, the server infrastructure is set up and customized to meet the project's needs.

5.1.4 Development Environments

The Visual Studio IDE and the Unity game engine are both used in the project's development. While Visual Studio is a potent IDE for C# development, Unity offers a complete development environment for creating video games.

5.2 Input-Output Design

5.2.1 User Inputs

Using the client application built on Unity, players engage with the game by giving inputs like clicking on squares on the game board. These inputs go through client-side processing before being submitted to the server.

5.2.2 Server-side Processing

Based on the rules of the game, the server processes the inputs that have been received and takes the proper action. For instance, it uses database access to update the game state, disperse soldiers throughout the clicked area, and update the soldier count.

5.2.3 Game State Updates

The game state is updated as a result of the server-side processing, and the modified game state is communicated to all players. Players see the modified state in the client application and proceed in accordance with it.

5.3 System Requirements and Design Features

The game is only compatible with the Windows operating system. The game's minimal system requirements should be comparable to those of a game of a similar genre, like Chain Reaction Game.. You can see the system requirements for the Chain Reaction game in Figure 5.1.

5.4 Server Infrastructure and Architecture

The game servers are designed to run on localhost using Docker Desktop. An open-source game server called Nakama is used to build the server architecture.

5.5 Client Application Design

The Unity game engine was used in the development of our client application for the game. Players' inputs are delivered directly to the server from the client side. The server's permission system checks and, if necessary, modifies players' behavior.

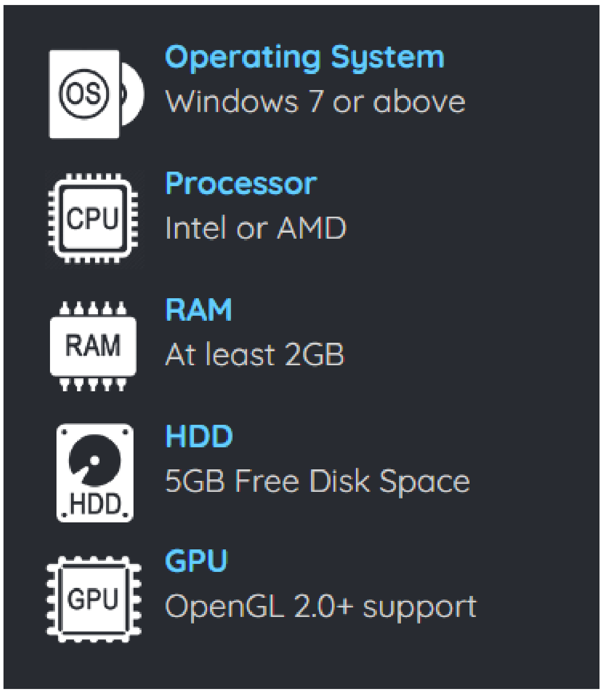
Operating System
Windows 7 or above
Processor
Intel or AMD
RAM
At least 2GB
HDD
5GB Free Disk Space
GPU
OpenGL 2.0+ support

Figure 5.1 System Requirements of Chain Reaction Game.[2]

5.6 Game Engine and Algorithm Design

For our game, Unity was used as the game engine. The physics engine and autonomous navigation mechanism integrated into Unity were used to develop the game's algorithm. The algorithm running on the server processes the player activities, and the results are broadcast to all players.[3]

5.7 User Interface Design

The game's user interface was created to improve players' in-game interaction. The user interface design tools of Unity were used to construct the interface.

5.8 Data Integration and System Integration

The game's data is handled on the server and is kept in a Postgresql database. The Nakama server's features are interwoven with how some game components are managed.

6 Application

The game in the project is a multiplayer turn-based strategy game. By causing your own soldiers to interact with the soldiers of other players, you can takeover the control of the board and win the game. Players can put their colored tiles' worth of marked soldiers on the board. Each square has a maximum capacity, and when that capacity is reached, the soldiers move to other squares.

Each player takes turns putting their men on the board at the beginning of the game. Each player moves their soldiers and places new soldiers as it is their turn once the game order has been decided. When a soldier comes into contact with another soldier, it can spread in any direction. Depending on how much space each soldier occupies in the square, they each spawn more soldiers.

The game has authentication and authorisation security features. The game can only be joined by participants who have correctly entered their login information and are approved. The game's server infrastructure makes use of the Nakama game server engine. Docker Desktop is used to build the server.

6.1 Basic Mechanics

Soldier movement: Players can maneuver their soldiers throughout the playing field. If a soldier comes into contact with at least one other soldier, it can expand to neighboring squares.

Production of soldiers: The number of soldiers that each soldier may create depends on the size of the square that it is located on the board.

Soldier capture: When two players' soldiers come into contact, one player's men are taken prisoner, and that player now has possession of the square.

The game's rules are straightforward and uncomplicated. When every soldier on the board is of a single player's color, the game is over.

6.2 Different Tiles From Game

Here is the different types of tiles from the game in figure 6.1:

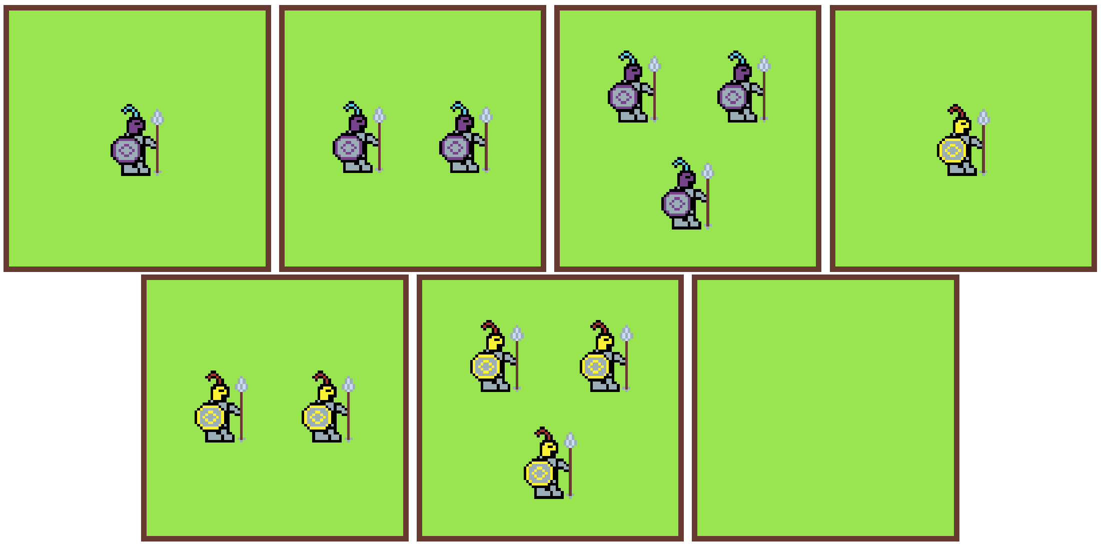

Figure 6.1 Tiles From Game

6.3 Example Game Board

Here is an example game board in figure 6.2 screen shot while playing:

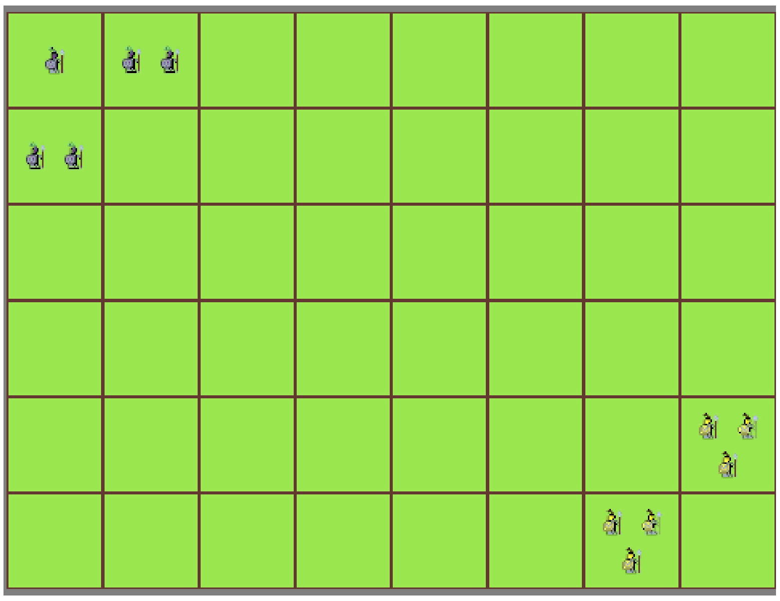

Figure 6.2 Example Game Board

7
Performance Analysis

7.1 Server Configuration

On the local host, the server is operating on Docker Desktop, which offers the best performance. Following a server authoritative approach, the server handles all of the game logic and processing. This guarantees accuracy and discourages fraud. A tick rate of 10 times per second is used to update the game state and transmit it to clients, ensuring coordinated gameplay.

7.2 Data Transmission

In order to facilitate quick data transfer, the game's information is sent to clients in JSON format. Clients receive only the information that is required, limiting the quantity of data provided and lowering performance overhead.

7.3 Nakama Integration

Nakama makes communication easy and effective thanks to its special features for both server and client sides. Nakama's built-in capabilities aid in maintaining a steady and dependable connection, which improves performance overall.

The technology provides smooth gameplay, effective data transmission, and a dependable connection by putting these performance standards in place, giving users an improved experience.

8
Experimental Results

8.1 Used Applications and Systems

8.1.1 Unity Game Engine

- The graphical and user interface design of the game were created using the Unity game engine.
- Unity is a platform with several features that makes game production easier.

8.1.2 C# Programming Language

- The basic mechanics and gameplay of the game have been developed by using the C# programming language.
- For the creation of video games, C# is a well-liked programming language that works well with Unity.

8.1.3 Nakama Server Engine

- On the server side of the game, the Nakama server engine has been used.
- Nakama is a server infrastructure that guarantees security while streamlining server-side operations in multiplayer video games.

8.2 Server Authoritative Structure

Nakama supports both relayed multiplayer and the server-authoritative multiplayer model, providing you the choice and flexibility to select the strategy that works best for your game.

All gameplay data exchanges in server-authoritative multiplayer are verified and disseminated by the server. For Nakama to enforce the game's rules (such as how many players may join, whether matches can be joined in progress, etc.), you must create custom server runtime code.

It is a design choice depending on the desired gameplay; there are no strong determining variables that require the relayed or authoritative method over the other. Authoritative multiplayer is better suited for games that rely on a central state that is controlled by the game server, games that have more players per match, games where you don't want to trust the game clients and instead want stricter control over the rules of the game to reduce cheating, etc.[4]

8.3 Match Handler

Nakama has a match handler which helps us to handle multi player matches. There are 7 functions required in any match handler. These functions are called only by Nakama, they cannot be called directly by clients or other runtime code.

- Match Init: Invoked when a match is created as a result of the match create function and sets up the initial state of a match. This will be called once at match start.
- Match Join Attempt: Executed when a user attempts to join the match using the client's match join operation. This includes any re-join request from a client after a lost connection is resumed, where client will need to explicitly re-join upon reconnection. Match join attempt can be used to prevent more players from joining after a match has started or disallow the user for any other game specific reason.
- Match Join: Executed when one or more users have successfully completed the match join process after their MatchJoinAttempt() returns true. When their presences are sent to this function the users are ready to receive match data messages and can be targets for the dispatcher's BroadcastMessage() function.
- Match Leave: Executed when one or more users have left the match for any reason including connection loss.
- Match Loop: Executed on an interval based on the tick rate returned by MatchInit(). Each tick the match loop is run which can process messages received from clients and apply changes to the match state before the next tick. It can also dispatch messages to one or more connected match participants. To send messages back to the participants in the match you can keep track of them

in the game state and use the dispatcher object to send messages to subsets of the users or all of them.

- Match Terminate: Called when the server begins a graceful shutdown process. Will not be called if graceful shutdown is disabled. The match should attempt to complete any processing before the given number of seconds elapses, and optionally send a message to clients to inform them the server is shutting down. When the grace period expires the match will be forcefully closed if it is still running, clients will be disconnected, and the server will shut down.

- Match Signal: Called when the match handler receives a runtime signal. Match signals allow the match handler to be sent a reservation signal to mark a user ID or session ID into the match state ahead of their join attempt and eventual join flow. This is useful to apply reservations to a matchmaking system with Nakama's matchmaker or match listings APIs.

[4]

8.4 Sending Match State

Nakama has real-time networking to send and receive match state as players move and interact with the game world.

During the match, each client sends match state to the server to be relayed to the other clients.

Match state contains an op code that lets the receiver know what data is being received so they can deserialize it and update their view of the game.[5]

9 Results

The project's goals have been met in this study, according to the findings. The project's major objective is to provide users with the ability to play an interactive multiplayer strategy game and to guarantee its security.

The game's graphics and user interface were created using the Unity game engine during the software design phase. C# programming was used to create the game's gameplay and fundamental principles. Users can do things like manage armies in the virtual world, apply spreading techniques on grids, and engage in rivalry with other players.

Server authoritative structure was used as security precautions. In this setup, the server handles all game-related tasks and acts as the gatekeeper to thwart player attempts at cheating. The server verifies and keeps track of every player action to guarantee fair gameplay.

The game's server engine was Nakama. A technology called Nakama makes server-side operations in multiplayer games easier. Nakama was used to carry out tasks including situating all soldiers, dispersing player actions over grids, and synchronizing the game state among all players. This seeks to protect the game's security and stop efforts at cheating.

The use of a local server running on Docker Desktop produced good performance results. The operations on the server side were completed fast and effectively. Since only relevant information was exchanged, there was no performance impact when game information was sent to clients in JSON format.

In conclusion, the project's goals have been met. By comprehending the game's fundamental principles, users can play it successfully. The server authoritative structure maintains the game's security and fosters an environment that is conducive to fair play. Nakama is used on the server side to guarantee safe and seamless communication between players. The needed performance has been attained through the use of a local server and effective program architecture.

References

[1] J. K. Haas, "A history of the unity game engine," 2014.

[2] *System requirements of the chain reaction game*. [Online]. Available: https://www.emulatorpc.com/chain-reaction/.

[3] *Right game engine choice, unity*. [Online]. Available: https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you.

[4] *Nakama authoritative multiplayer*. [Online]. Available: https://heroiclabs.com/docs/nakama/concepts/multiplayer/authoritative/#receive-data-messages.

[5] *Nakama unity client guide sending match state.* [Online]. Available: https://heroiclabs.com/docs/nakama/client-libraries/unity/#sending-match-state.

Curriculum Vitae

FIRST MEMBER

Name-Surname: Gazi Muhammet Canbolat
Birthdate and Place of Birth: 04.05.2000, İstanbul
E-mail: muhammet.canbolat@std.yildiz.edu.tr
Phone: 0538 078 48 02
Practical Training: Tübitak Bilgem Yazılım Geliştirme Birimi Web Yazılım Geliştirme

Project System Informations

System and Software: Windows OS, DirectX 11
Required RAM: 2GB
Required Disk: 256MB