Multi-Rimecraft Documentation

By Ayrton Muniz

2/28/22

Contents

[Introduction 2](#_Toc96981357)

[Compilation 2](#_Toc96981358)

[How to Run 2](#_Toc96981359)

[Controls 3](#_Toc96981360)

[Game Features 3](#_Toc96981361)

[Network Protocol Implementation 3](#_Toc96981362)

[How Data is Sent 3](#_Toc96981363)

[Abstractions Used 4](#_Toc96981364)

# Introduction

Multi-Rimecraft is an inspiration based off the original Rimecraft game created by me. It differs in that there is a separate executable (RimecraftServer.exe) that hosts a server and is required for the original (Rimecraft.exe) to run. Meaning the Rimecraft.exe, the client program, needs to join a server in order for the game to be playable beyond the main menu.

A game in Rimecraft is simply an infinite world of voxels that are editable via left/right click. Where left click shoots a projectile that bounces depending on how long you hold and right click places a block if you are holding any. When a voxel is destroyed an item will drop which can be picked up by a client program and stored in the client’s inventory. This inventory can be opened using Q and any held item in the inventory can be dropped with G. The player controller itself is controlled by WASD (move), Shift/Control (sprint/crouch), and space bar (jump).

# Compilation

To compile the client code you will need to use Unity version [2020.3.26f1](https://unity3d.com/unity/whats-new/2020.3.26) and build the project to windows (control + B). To compile the server code, open RimecraftServer.sln using visual studio and go under Build. Under Build there is publish and target win-x86, netcoreapp3.1. In additional settings deploy as self-contained if the goal is to deploy the server.exe program.

I don’t really recommend building the code from scratch as doing so will take long to just open the Unity project. Cloning the repository is probably the easiest way to do so as this is a large game. This is why I have also submitted the build for the game.

# How to Run

Rimecraft and RimecraftServer is only tested to run on Windows 10 x86. To run the client, go under Build and run “Rimecraft.exe.” To run the server manually go under Build, then Server, and run RimecraftServer.exe. If there is any difficulties with getting a connection to the server, add RimecraftServer.exe as a rule to bypass the windows defender firewall and run it as administrator.

When the client program is opened there is two important buttons, Play and Settings. To begin, adjust the settings to have a low render distance to prevent any lag issues (such as a view distance of 2 or 3). After doing this, hit Play and there should be a bunch of input fields with a few buttons. This is the server list. The server list does *not* contain all available servers in Multi-Rimecraft. This is intended, as the server list is modeled after Minecraft.

To add a known server IP, input an IP under the IP address field on the bottom right. Also add its corresponding port. Once done hit the “add” button to add the server to the list. The client program should then test to see if it can connect to the given IP and Port through opening a TCP connection and closing it. If it succeeds it will be green. *Warning:* If its green despite it not actually making a connection the client program will crash. To prevent this be sure to ping or verify you have properly hosted the server.

To automatically host a server, input a seed (like 0) and simply hit Host and the server program will open. This will be a console program. Once a server program is ran on some machine and the client has added the server IP, the client can hit the corresponding IP in the list, which is a button, to join the server. If a client has successfully joined the server but no blocks are appearing after a minute, then leave and rejoin until it does. This will occur occasionally.

# Controls

Player Controller:

* WASD: Move
* Shift: Sprint
* Ctrl: Crouch (keeps from falling off ledges)
* Space Bar: Jump

Inventory:

* Q: Open/Close inventory
* Mouse Wheel: Scroll through hotbar (bottom row of inventory)
* 1-0: Select corresponding slot in hotbar
* Mouse click: Pick up item
* G: Drop held item

Misc:

* Esc: Disconnect from server
* R: Teleport to (0, 10, 0)
* Right click: Place block that is held in hotbar
* Left Click Held: Charge staff to get more bounces for projectiles
* Left click Release: Fire projectiles that break blocks

# Game Features

* Editing Voxels
* Firing Projectiles that bounce
* Inventory management and pickable items
* Terrain Generation
* Infinite Terrain
* Server List
* Multiplayer

*Note: If at this point you want to quit, hit Escape and then hit Escape again once in the menu. This should close the application. You can also hit the Quit button in the main menu*

# Network Protocol Implementation

The library used is the built-in C# library System.Net.Sockets. In this project UDP was used for player movement/rotation while everything else TCP is used.

## How Data is Sent

When the server starts it will create P# (Player Number) number of Clients each with a unique id (e.g. 2). When the Player connects, they will be sent an assigned ID and then respond with that ID back to the server. The Player will also send its view distance for the server to use. The server will then send this ID to all other clients so that they can spawn “dummy players” in their worlds. These dummy players will match the position and rotation for the corresponding client IDs. When a client is assigned an ID, the server will also give it the IDs, positions, and rotations of all other clients. The client will take this information and spawn the corresponding dummy players and one actual player (with information like main camera and so on).

Once in the game, the client will constantly send updates on its position and rotation to the server and the server will forward this information to all other clients. Other information that is forwarded is the spawning of projectiles. What this means is each client will spawn their own projectiles, but only the source client’s projectiles will break any blocks. Another thing that is forwarded is if the client picks up or drops any blocks. This is so that other clients know when to drop items or remove items in their own worlds. An item simply is an uuid, an amount, and a block ID (which is a ushort).

But what about the blocks? When the client joins the server, the server will see that they have never seen any chunks before. So it will generate a 2Nx2Nx2N chunks (where N is the view distance) and then send this data to the client. The server will always manage if it needs to send chunks to each client. All the client needs to do is listen for any new chunks and decide if it needs to create the mesh for that chunk and render it. For block manipulation, all the server does is forward the changes done by clients to all other clients. For example, if I break a block in my world the server will tell all other worlds to break a block in the same position in their worlds. The server is the primary storage of blocks/chunks.

To better visualize how this is all done, its best to look at Diagram 1 from Progress report 2.

## Abstractions Used

When it comes to details on how all of this data management is sent, packets are identified through using an integer. This integer identifies what type of packet it is (e.g. packet containing chunk information). Packets are listened to by a thread pool that handles each specific type of data.

There are specific functions for converting the byte information to the corresponding data and vice versa (e.g. integer to byte array of 4). On the server side there will be ServerSend.cs that manages what functions that send data, ServerHandler that manages what functions receive data, and Server.cs that manages opening TCP/UDP connections and the packet labels. The client side its similar but replace “server” with “client.” Packet.cs manages how to write/read bytes that are sent by the protocols.

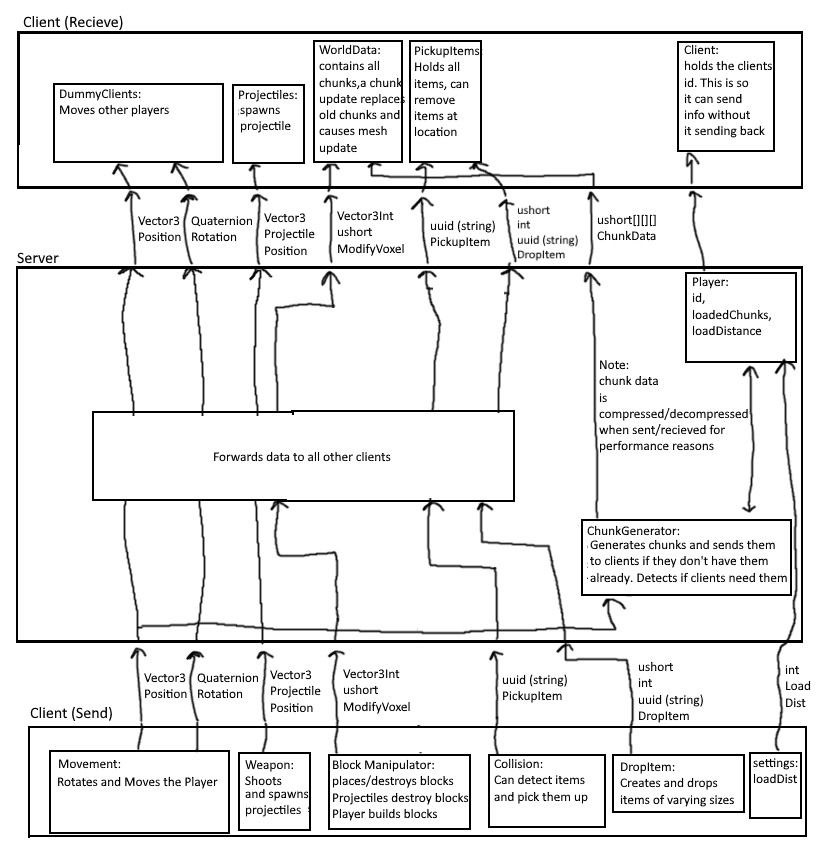


Diagram 1: Network Protocol Implementation