Week 3 Reflection

I spent more time learning about remote functions and remote events, as this has been the trickiest part of learning lua for me. I also spent time working on modulescripts, which I will explain below.

In the prior reflection, I touched on remote functions and remote events, as well as the distinction between server scripts and local scripts. However, it raises a question — if everything in a local script is relative and separate from a server script, how do you communicate between the two scripts?

For example, if a player clicked a button locally, it wouldn't immediately notify the server, since the client handles that input without passing it along. Therefore, you couldn't really change attributes within the server just by a localscript itself. **This is why remotes are important.** "Remotes," otherwise known as remote events and remote functions are the gateway between the client and the server. I see them as either a one-way, or two-way connection, depending on what type of remote you're using.

RemoteEvents

Remote events, which are held in ReplicatedStorage, are either fired via the client and caught by a server script, or vice versa. You're able to pass in arguments as well — it makes things a lot easier.

RemoteFunctions

Remote functions are the exact same, but instead, they can provide two-way communication between the client and server.

An analogy

I understand the distinction between the two to be similar to TCP vs UDP. UDP (remote events), is more for one-time communication or broadcasting. TCP (remote functions),

are more useful if you need a response, and can represent an ongoing communication between the client and server.

ModuleScripts

ModuleScripts are important and rarely understood in ROBLOX lua programming. The first reason why they are rarely understood is that they have their own syntax.

Module scripts are essentially a module or table of functions that you can call. It's an object, essentially.

You begin by creating and defining the modulescript, which looks like:

```
-- Tables store multiple values in one variable
local MyFunctions = {}

-- Add a few functions to the table
function MyFunctions.foo()
  print("Foo!")
end

function MyFunctions.bar()
  print("Bar!")
end

-- ModuleScripts must return exactly one value
return MyFunctions
```

(via ROBLOX documentation)

Then, you can import the module and call its associated function:

```
-- The require function is provided a ModuleScript, then runs
-- the code, waiting until it returns a singular value.
local MyFunctions = require(script.Parent.MyFunctions)

-- These are some dummy functions defined in another code sample
MyFunctions.foo()
MyFunctions.bar()
```

This is one of the primary ways that LUA in ROBLOX allows for object oriented programming without any other frameworks that you have to import. It's different from other scripts, which don't necessarily speak to one another. Modulescripts are typically used when a function has to be called repeatedly, since the overhead on the server is typically very low compared to other options, like a remote event.

One of the primary things that I've been struggling with this quarter has been organizing my code in a manner that allows for me to fix any bugs or errors as I build upwards. I need to find a way to build a strong foundation for a codebase. I'll look into that next week.

Game Design

Throughout the past few weeks, I've learned that in order to have an enjoyable and successful game, it's important to have the tasks required to win be obvious to anyone who plays. As such, I also added a "tutorial" course and added signs for specific levels where new assets or parts are introduced. The hope is that the player will read these signs and gain an understanding of the game. Retention is an important statistic in whether the algorithm favors your game, so it's important to prevent players from playing and then leaving after twenty seconds because they don't know what to do. The average age of the ROBLOX player is also typically low.