Week 2 Reflection

In Week 1, I learned the general syntax of the LUA language. Now, it's time for me to understand how code is organized within the ROBLOX environment.

The main understanding I have currently about how ROBLOX games work in general is:

- Client connects to server
- Client has its own set of functions and input handlers, which are then broadcasted to the server when necessary
- Every client connects to the same game server
- Game server handles things that are replicated amongst all clients, such as player health, emulating player movement on other players' screens, etc
- Client device handles things that are specific to the client (obviously), such as relative interfaces, mouse clicks, etc.

There are multiple sections within the ROBLOX environment in order to organize the code and objects. The first is the **workspace,** where all of the physical parts that players see in games are organized. You can have folders within the workspace. What I immediately figured out was that **unless it's in the workspace, players won't see it.** Scripts can be kept inside of the workspace as well, but that's not the best way to organize your code.

Now that we've established that the workspace is generally for physical attributes, let's take a look at where scripts are held:

1. ServerScriptService

- This is where server-side scripts are executed, typically immediately once the game instance has begun. You can add a function to wait for a certain condition before firing, but the default is instantaneous execution.

2. ReplicatedStorage

- Replicated storage is where remoteFunctions and remoteEvents are used (something which I will touch on later on). It's basically seen as the "dresser" of the ROBLOX environment. You throw everything in there, and if you need it, you reference it from there.

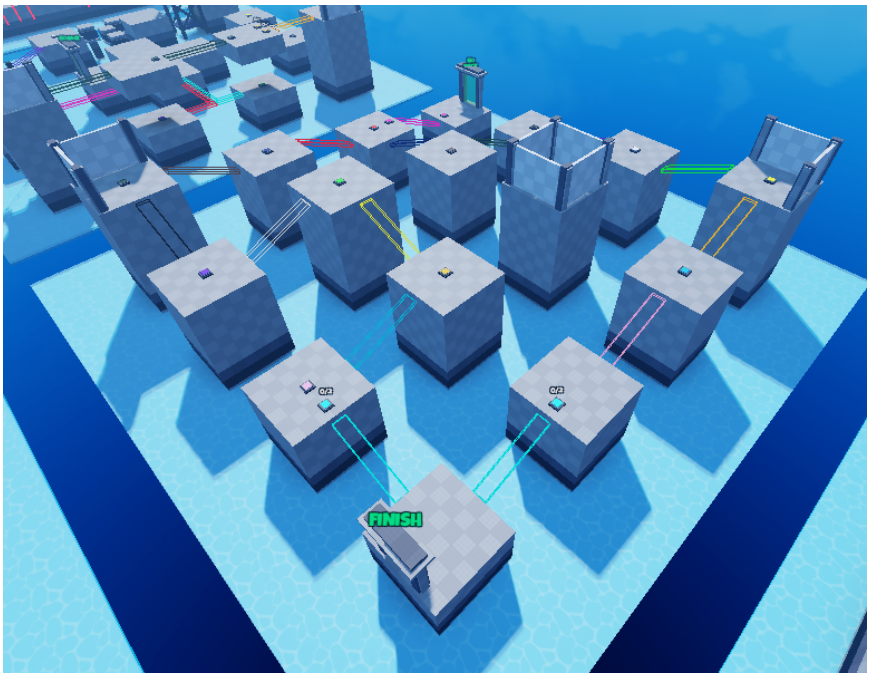
3. StarterPlayerScripts

- As the name suggests, starterPlayerScripts are strictly for the LocalPlayer — whereas the prior two organizational structures are for the server

It's also important for me to touch on the distinctions between the local player and the server. For what the player actually sees, such as a graphical interface, all of those things have to be manipulated in a localscript. Otherwise, everyone would see the same thing. Essentially, if you want your functions to be **relative** to what the player sees and does, then you have to use it locally. If you wanted to manipulate something that everyone sees, however, you have to use a server script.

During this week, I also worked on some of my level-building skills, which is also important within ROBLOX game development. There is a lot that I can cover, but the basis of it is:

- Buildings consist of parts, which you can change the attributes to. Such as: color, size, transparency, surface-image, texture, etc
- Parts can have graphical text labels associated with them, which are displayed to every user within the server session



The game that I’m working on this quarter is a two-player obstacle course. The example buildings are below. You spawn in to the game, and you and a friend can stand on buttons to activate them, so you can cross the bridges. If you fall into the water, you lose and can try again. The point of the game is to read the “Finish” portal and win!

By working on the modeling **and** scripting, I got a taste of what game development is truly like on both sides of the ball. Building was a tedious process that got faster as the initial assets were made, but having to label every part something different and group assets by color was incredibly tedious and I imagine there’s a better way to do it. Nonetheless, I learned the ins-and-outs of ROBLOX studio by doing this, which was a plus.