Game Programming

Szymon Szydlik | 6C5

Trape 2023

Tutrice: Edyta Zenatello

Haus: Schoenfels

Table of Contents

1. Introduction

1.1. My game

The first idea I had was making a game where the player could travel between a variety of planets. But I thought that concept was a little bit boring, that’s why I decided to make a game where you can make your own planet. The goal is to make the best-looking planet you can!

1.2. Lua

A lot of different applications employ the lightweight, high-performance programming language Lua. A group of academics from the Pontifical Catholic University of Rio de Janeiro in Brazil developed and applied it for the first time in 1993.

One of Lua's important characteristics is its compact size and minimalist design, which makes it simple to incorporate in other programs. Its straightforward syntax makes it simple to understand and use for both seasoned programmers and newbies. Additionally, Lua is quite portable, supporting a variety of operating systems such as Windows, Linux, and macOS.

With a strong and adaptable C API that enables integration with other systems and languages, Lua is also very expandable. Due to this, it is a well-liked option for game creation and is used in several well-known game engines, including Unity, Unreal Engine and ROBLOX Studio. Numerous additional applications, including scientific computing, network programming, and web development also employ Lua.

The performance of Lua is outstanding. A virtual machine interprets the bytecode created by the compilation of Lua code. This enables effective execution, with a speed that is frequently on par with that of compiled languages. Lua is a strong choice for embedded systems and other resource-constrained situations due to its minimal memory footprint and memory-efficient design.

In general, Lua is a robust and adaptable programming language that is suitable for a variety of applications. It is frequently used in game development, scientific computing, network programming, and web development due to its compact size, extensibility, and outstanding speed.

1.3. ROBLOX Studio

A software development tool for the online gaming platform ROBLOX is called ROBLOX Studio. It was created by the ROBLOX business. With the help of a drag-and-drop user interface and a basic programming language, users can make their own video games and virtual worlds. Users of ROBLOX Studio can design environments, make their own 3D models, and utilize Lua to program game mechanisms.

The accessibility of ROBLOX Studio to users of all ability levels is one of its key advantages. With the help of the drag-and-drop interface and straightforward scripting language, even non-programmers may make their own video games and virtual worlds. The software is also available to a wide variety of people for free download and use.

Additionally, ROBLOX Studio has a large and active community of developers and creators who share tutorials, tips, and resources to help others learn and create with the software. This community helps new users to get started, and experienced users to improve their skills and take their creations to the next level. This is a good game engine and I personally like using it.

1.4. Visual Studio Code

Microsoft created Visual Studio Coding (VS Code), a well-liked and potent code editor. It is available for Windows, Linux, and macOS and is open source. With a large selection of extensions and themes available to improve its functionality and appearance, VS Code is extremely customizable.

Debugging tools are also provided by VS Code, making it simple to locate and correct issues in your code. Several languages, including JavaScript, TypeScript, JavaScript, Python and Lua, which is the language I used for my game, are supported by the editor's built-in debugger. Additionally, it offers a live sharing feature that facilitates collaboration by enabling numerous developers to collaborate in real-time on the same codebase.

In addition, VS Code has a wide range of extensions that can be added to enhance its functionality. These extensions can provide additional languages, themes, debugging tools, and more. Many of these extensions are open source and community-developed, which means that they are constantly being updated and improved.

1.5. Blender

A free and open-source 3D modeling, animation, and rendering program is called Blender. The Blender Foundation created it, and it is widely used in the video game, film, and architectural visualization sectors.

One of Blender's standout features is its robust 3D modeling capabilities, which let users easily create and edit intricate 3D models. Numerous modeling capabilities, including as sculpting and retopology are included in the software. It also offers a significant collection of tools for designing textures and materials, enabling the creation of models that are incredibly realistic and detailed.

Overall, Blender is a powerful and versatile 3D modeling and animation software that is widely used in a variety of industries. Its combination of

blender®

advanced modeling, animation, and rendering tools, as well as its support for multiple renderers and the active community make it an excellent choice for a wide range of projects.

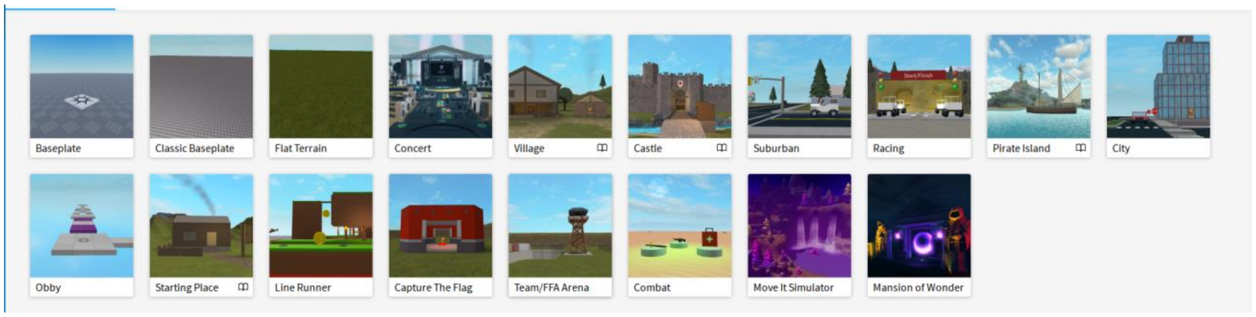
2. The making

2.1. The first steps

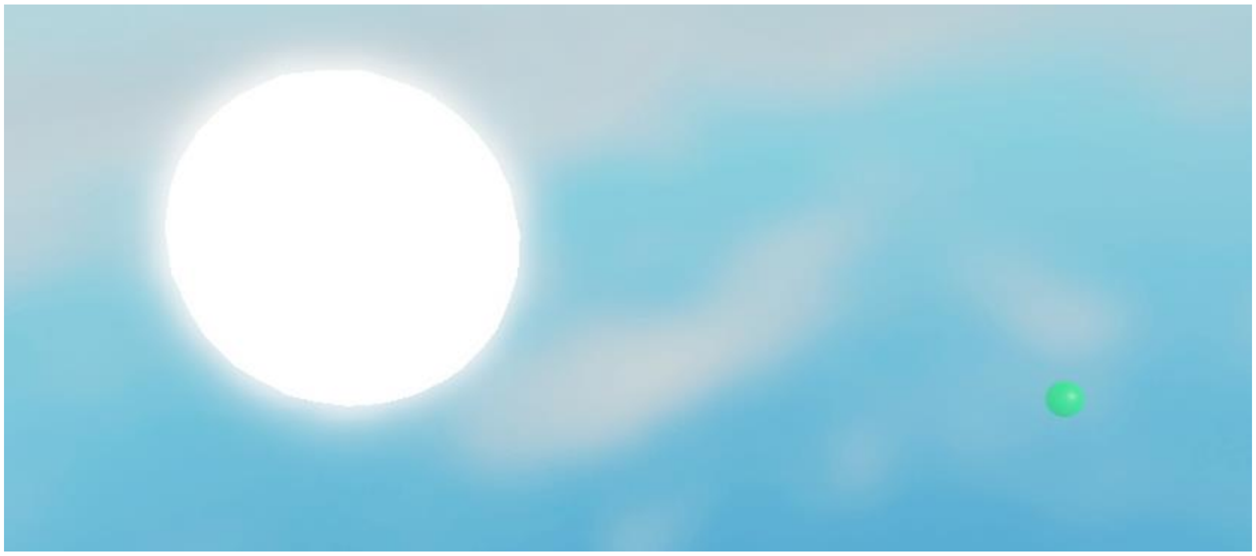
I wanted the game to have a lot of features a player can use and be as user friendly as possible. Here is a list of all the functions I decided to add to my game:

- **Physics** (Planet rotation etc.)
- **A ring generator tool** (Allows the player to generate rings for their planet.)
- **A resize tool** (Allows the player to resize their planet to their liking.)
- **Different materials/colors** (I wanted to add as many different materials/colors as I can, so the player has a good amount to choose from.)
- **A custom camera** (ROBLOX Studio has one built-in, but I wanted to make my own.)

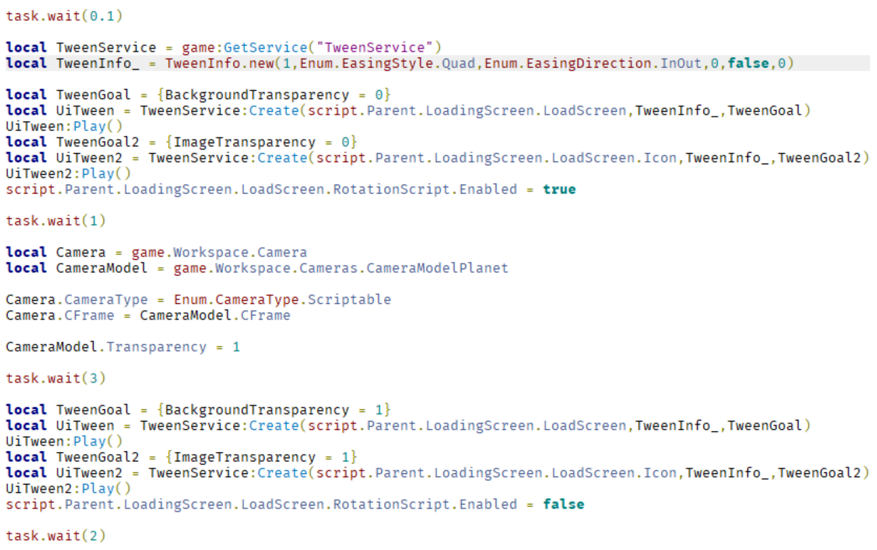
After planning, I could start my game. First, I opened a new project inside ROBLOX Studio.



ROBLOX Studio has a variety of templates to choose from, but I chose the „Baseplate“ Template, as it only has 2 starter objects that I can delete later. The next step was cleaning the workspace and deleting objects I don’t need. After had done that, I was left with a blank void of nothingness. I then decided to add 2 different objects – an object that will work as the camera for the player. You can see it in the picture: the green part is the camera, and the white glowing part is the planet.

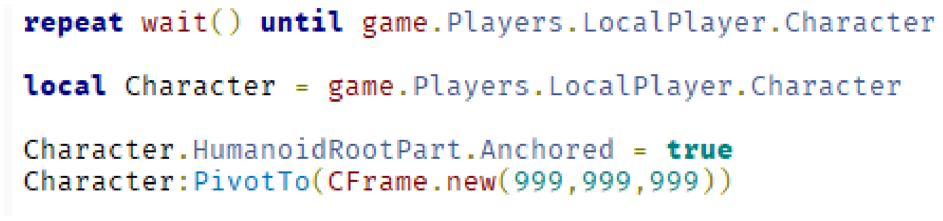


I then made a simple loading screen and a script that positions the player’s usual camera to the green ball. This way the player will not be able to move it.



```
task.wait(0.1)

local TweenService = game:GetService("TweenService")
local TweenInfo_ = TweenInfo.new(1,Enum.EasingStyle.Quad,Enum.EasingDirection.InOut,0,false,0)

local TweenGoal = {BackgroundTransparency = 0}
local UiTween = TweenService:Create(script.Parent.LoadingScreen.LoadScreen,TweenInfo_,TweenGoal)
UiTween:Play()
local TweenGoal2 = {ImageTransparency = 0}
local UiTween2 = TweenService:Create(script.Parent.LoadingScreen.LoadScreen.Icon,TweenInfo_,TweenGoal2)
UiTween2:Play()
script.Parent.LoadingScreen.LoadScreen.RotationScript.Enabled = true

task.wait(1)

local Camera = game.Workspace.Camera
local CameraModel = game.Workspace.Cameras.CameraModelPlanet

Camera.CameraType = Enum.CameraType.Scriptable
Camera.CFrame = CameraModel.CFrame

CameraModel.Transparency = 1

task.wait(3)

local TweenGoal = {BackgroundTransparency = 1}
local UiTween = TweenService:Create(script.Parent.LoadingScreen.LoadScreen,TweenInfo_,TweenGoal)
UiTween:Play()
local TweenGoal2 = {ImageTransparency = 1}
local UiTween2 = TweenService:Create(script.Parent.LoadingScreen.LoadScreen.Icon,TweenInfo_,TweenGoal2)
UiTween2:Play()
script.Parent.LoadingScreen.LoadScreen.RotationScript.Enabled = false

task.wait(2)
```

Then, I made a script that anchors the player's character in one place, which disables its movements.



```
repeat wait() until game.Players.LocalPlayer.Character

local Character = game.Players.LocalPlayer.Character

Character.HumanoidRootPart.Anchored = true
Character:PivotTo(CFrame.new(999,999,999))
```

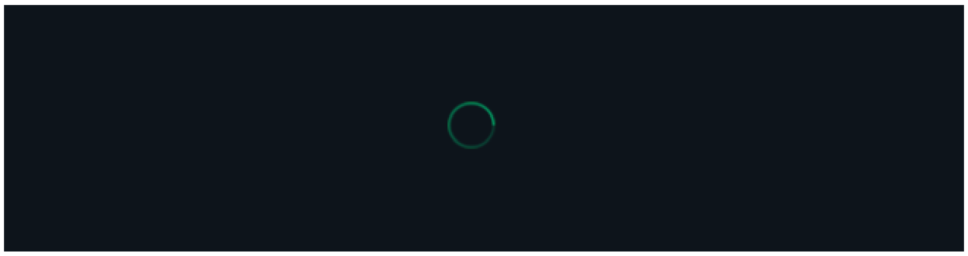
This code also moves the character to a position where it's not in the way of the planet. Now that I had done the setup, I could continue doing more important things.

2.2. User interface (UI)

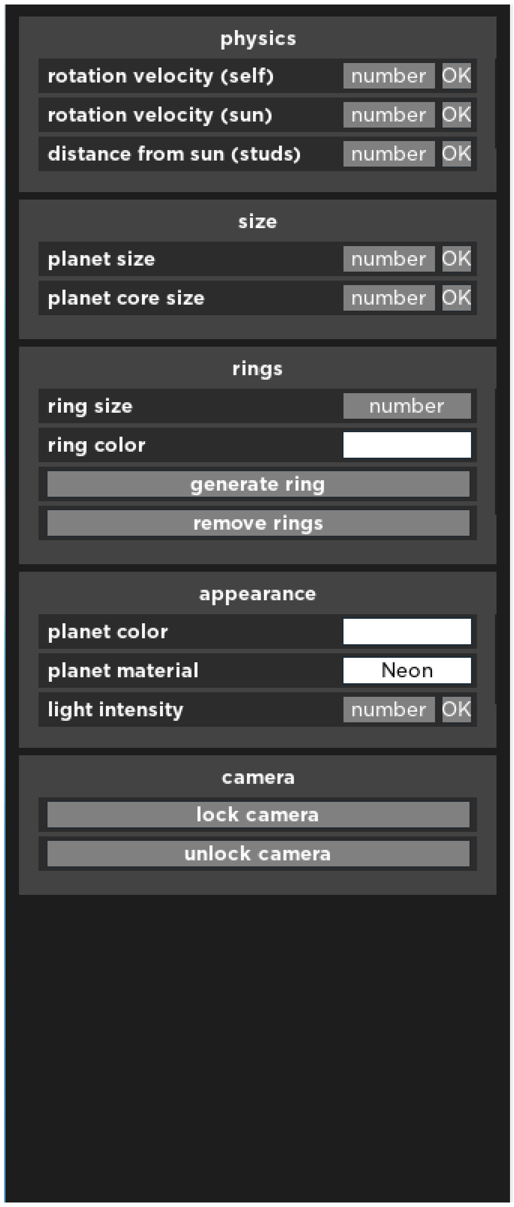
(2.2.1.) What is a user interface (UI)?

A user interface (UI for short) is the point of interaction between a user and a digital device or software application. For example, to interact with your phone you tap on the screen of it. But for a computer, you use the mouse and keyboard objects to interact with the computer. If you are on google and want to search for something, you use your keyboard to type it. That is an example of interacting with a device.

First, I added a loading icon to the loading screen I made in chapter 2.1.



Then, I started making a panel with buttons which the player can edit their planet with. I wanted to go for a minimalistic UI style. For the colors I decided I would go with grey and white. First, I made the frame for the UI. Then, in the frame I added a "UIListLayout" element which will automatically position all content in the frame. I then added 5 new ones to the frame.



There are 5 Categories: "Physics", "Size", "Rings", "Appearance" and "Camera". I will now explain what each one is for. The physics category has 3 input boxes that assign a value once it has been submitted. For example, if I set the "rotation velocity (self)" to 6, it will fire a remote to the server and the server will assign that value to the players planet.

(2.2.2.) What is a remote?

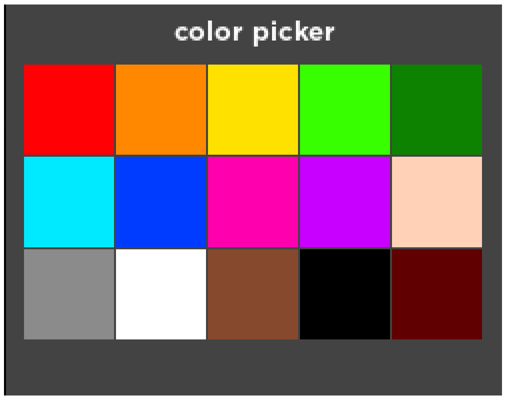
Imagine you (the "client") are playing a game. The game is multiplayer, which means that there are other people playing with you. It is a building game, and you decide to place a block. When you place the block, a remote fires the server to make your action visible to every player on the server. If the game doesn't do that, the action you have performed is only visible client-sided (only visible to you). In Lua you fire a remote like this:

```
local Arguments = {
	[1] = "Hello";
	[2] = 999;
	[3] = "Ok!";
}
game.ReplicatedStorage.Remote:FireServer(Arguments)
```

In the Arguments table there are 3 values. The first value that will be passed is "Hello", because it is marked as first in the table. You can also do it like this:

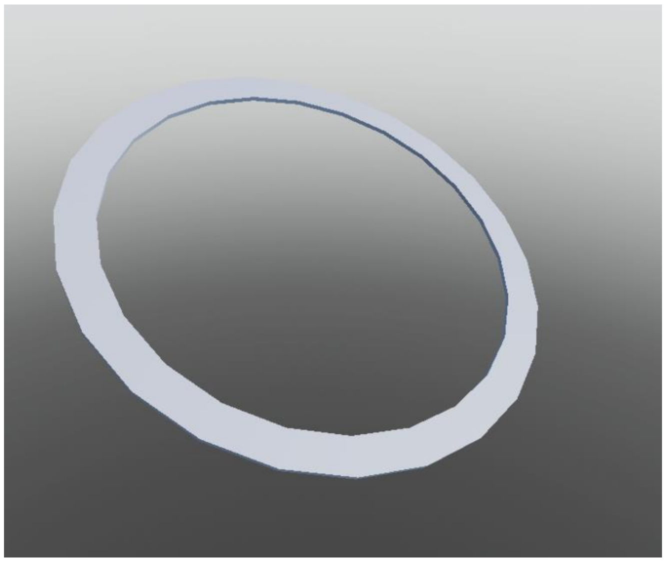
```
game.ReplicatedStorage.Remote:FireServer("Hello",999,"Ok!")
```

The rings category generates a ring for the player’s planet. You can edit the ring’s size and color. The appearance section applies a color, light or material to the player’s planet. Lastly, the camera category basically just unlocks the player’s camera or locks it in one place. I also made a color picker UI to make the options that require colors work.



2.3. Modeling

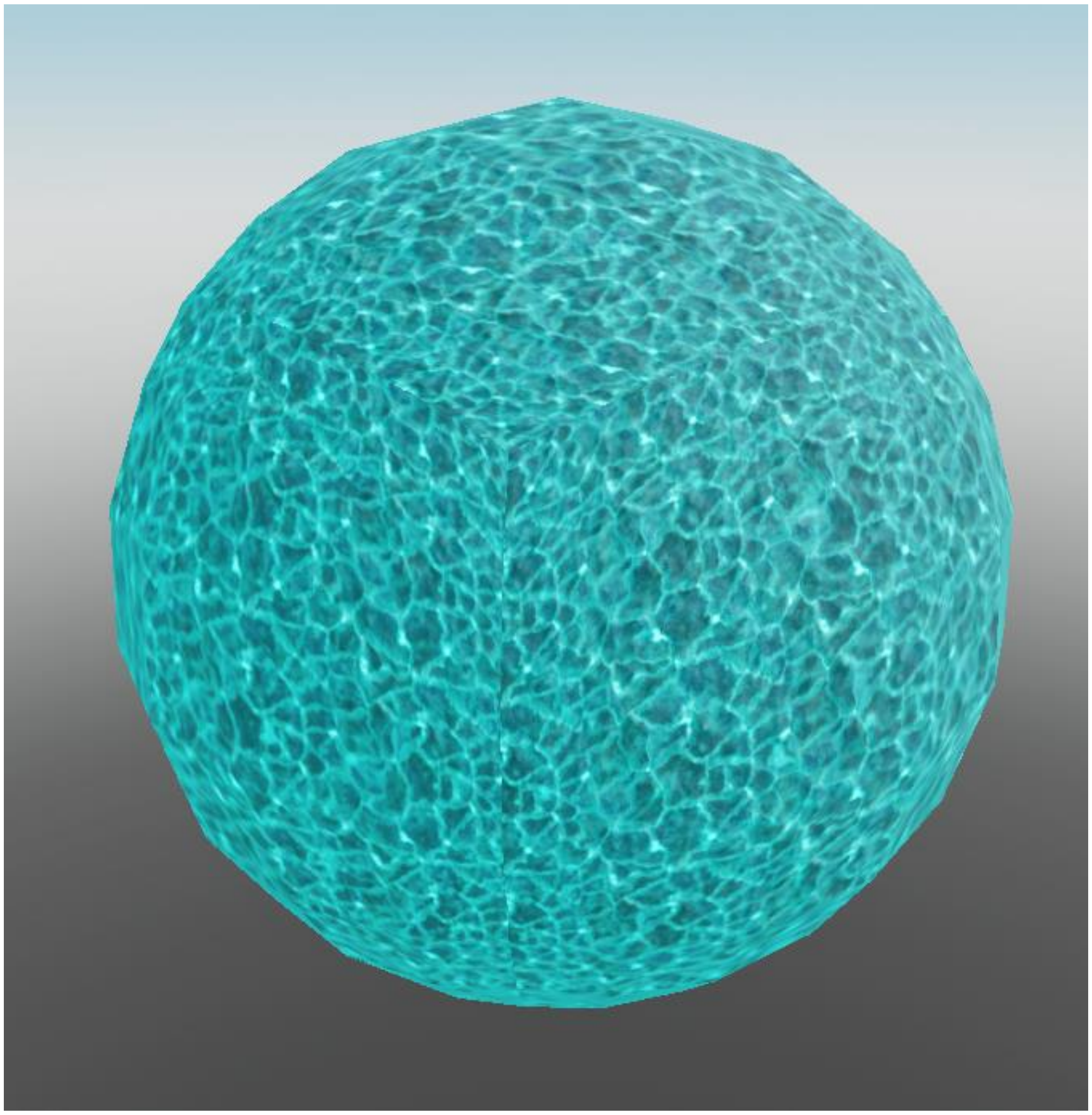
My game didn’t have a lot of things to model, but a lot things to code. I first started modeling a ring (for the ring generator).



This is how it turned out. I wanted the edges to be a bit smoother, but I thought this design wasn’t that bad. I designed the ring so it can scale on every axis (X, Y, Z) and that the ring can use different colors.

Then I started making some textures for the planet. ROBLOX Studio provides free textures, but I also made my own. Here is a texture I made:

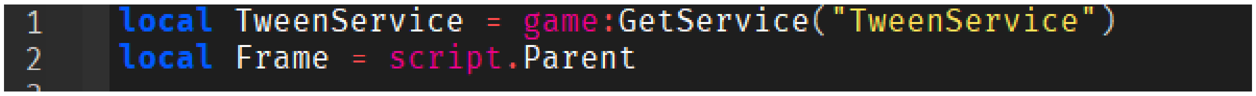
I called it “Blue Lava”



I also modeled some other objects for my game like spheres and other shapes.

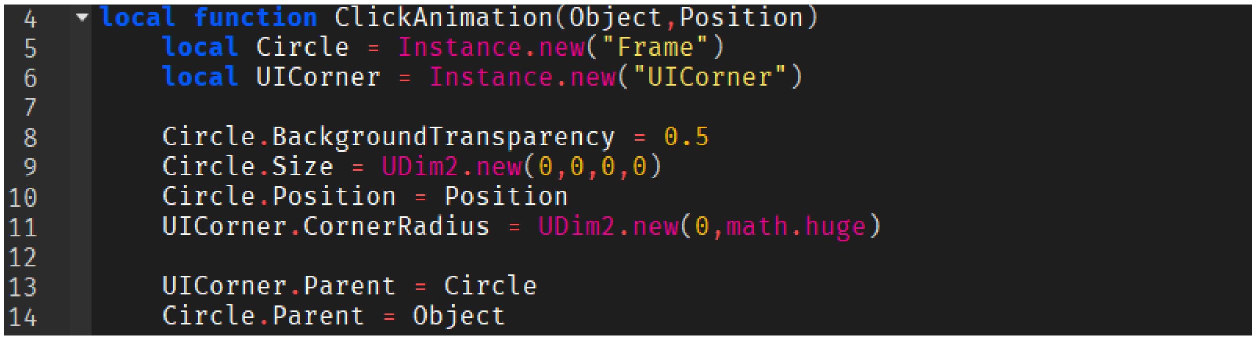
2.4. Coding

Now that I had everything done, I could start coding the game. I will be explaining what every code does in detail. First, I decided to make the UI function properly. I wanted the UI to be animated and I also wanted the UI to have sounds. First, I wrote a script that added click animations to the UI. This makes it when a player clicks a button it has a small effect. (The effect is a “Ripple” effect):



```lua
local TweenService = game:GetService("TweenService")
local Frame = script.Parent
```

On the first 2 lines of the code, I made 2 variables. “Frame” locates the UI frame. “TweenService” calls a tweening service provided by ROBLOX.

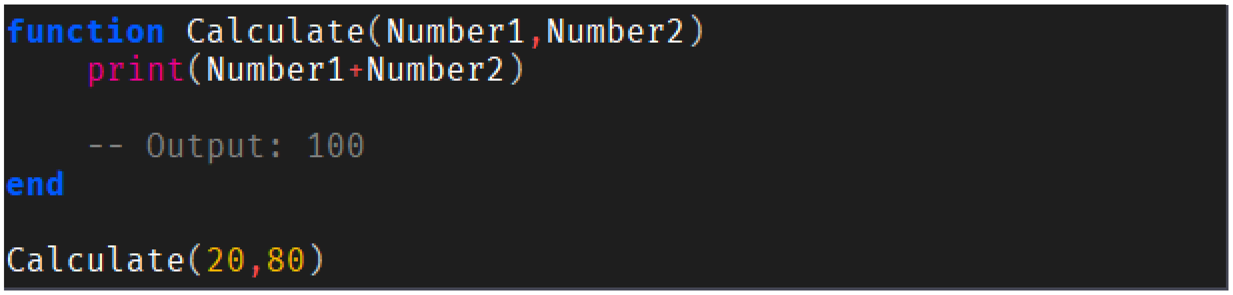


```lua
local function ClickAnimation(Object,Position)
    local Circle = Instance.new("Frame")
    local UICorner = Instance.new("UICorner")

    Circle.BackgroundTransparency = 0.5
    Circle.Size = UDim2.new(0,0,0,0)
    Circle.Position = Position
    UICorner.CornerRadius = UDim2.new(0,math.huge)

    UICorner.Parent = Circle
    Circle.Parent = Object
```

Here, I made a function called “ClickAnimation”.

(2.4.1) What are “Functions”?

You will see “Functions” in pretty much every programming language. In Lua, a function is a block of code that can be called at any point. Functions are first-class values which means they can be assigned to variables, passed as arguments to other functions and returned as results from functions. Here is an example of a function.

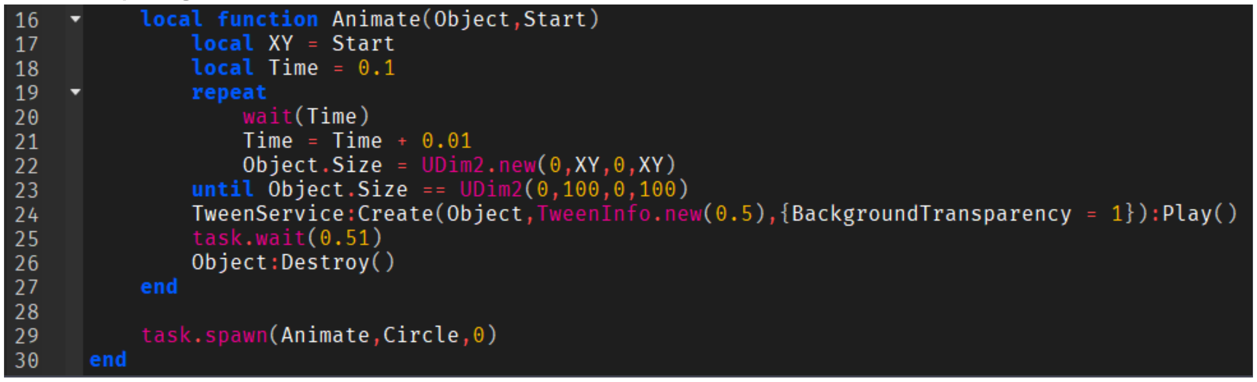


```lua
function Calculate(Number1,Number2)
    print(Number1+Number2)

    -- Output: 100
end

Calculate(20,80)
```

This function will calculate two numbers. In this case, 20 and 80. Once this function is called it will print out the result (100).

In this function, the circle that will be used for the animation is being created with “Instance.new”

Next, I programmed a function that handles the animation for the button.



```lua
    local function Animate(Object,Start)
        local XY = Start
        local Time = 0.1
        repeat
            wait(Time)
            Time = Time + 0.01
            Object.Size = UDim2.new(0,XY,0,XY)
        until Object.Size == UDim2(0,100,0,100)
        TweenService:Create(Object,TweenInfo.new(0.5),{BackgroundTransparency = 1}):Play()
        task.wait(0.51)
        Object:Destroy()
    end

    task.spawn(Animate,Circle,0)
end
```

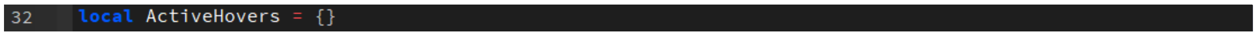
This function makes the created circle bigger over time and makes it fade away after. I also called this function with “task.spawn”. Here’s why:

(2.4.2) What is “task.spawn”?

Task.spawn allows you to create a new coroutine. Calling a function with task.spawn acts like making a new script run that function. The reason why I used task.spawn here is because when a player had clicked a button a few times in a row, the “Animate” function would have not finished animating the button in time, because it would have been called with every click. That’s why when this function is called with task.spawn, it creates a new coroutine. As I said before, a coroutine acts like a new script. The function executes and no matter what, it will not stop until it is finished. (I will explain coroutines more in detail on the next page)

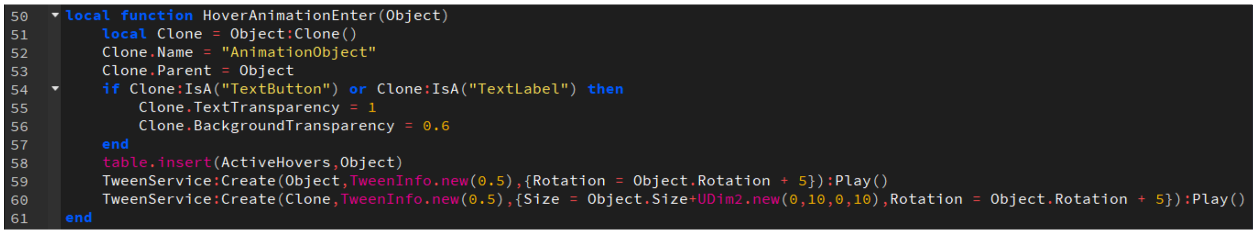
I also made my own tweening style. ROBLOX provides pre-made tweening styles, but I always like my own. I did use their pre-set tweening style on the fade-out animation.

Next, I made a script that adds hovering animations. When you hover over certain UI objects, it will play an animation.



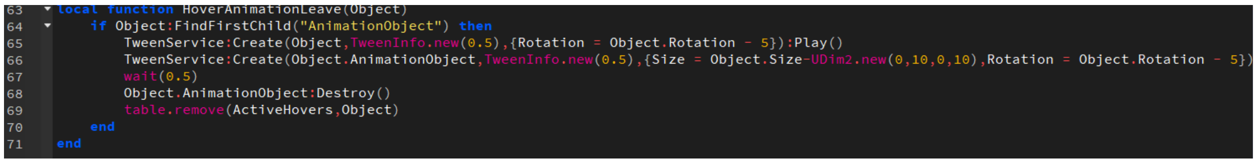
```lua
local ActiveHovers = {}
```

First, I made a table variable which will keep track of all objects that are currently hovered on.



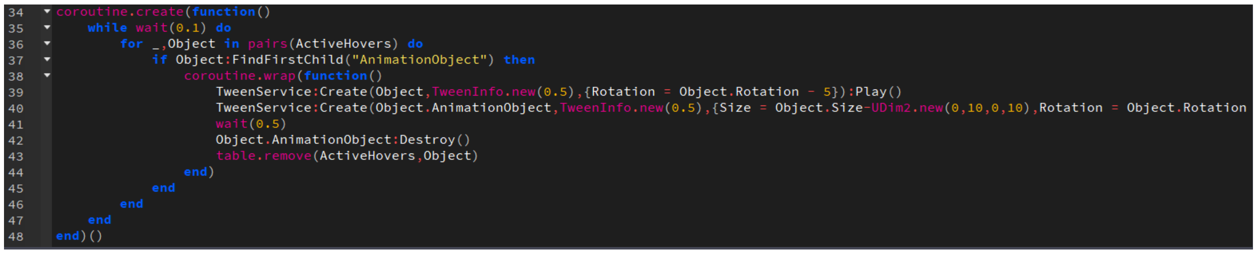
```lua
local function HoverAnimationEnter(Object)
    local Clone = Object:Clone()
    Clone.Name = "AnimationObject"
    Clone.Parent = Object
    if Clone:IsA("TextButton") or Clone:IsA("TextLabel") then
        Clone.TextTransparency = 1
        Clone.BackgroundTransparency = 0.6
    end
    table.insert(ActiveHovers,Object)
    TweenService:Create(Object,TweenInfo.new(0.5),{Rotation = Object.Rotation + 5}):Play()
    TweenService:Create(Clone,TweenInfo.new(0.5),{Size = Object.Size+UDim2.new(0,10,0,10),Rotation = Object.Rotation + 5}):Play()
end
```

Next, I made a function which will be called when a player hovers over a UI object. Here, a replica/clone of the object is made. Then it plays an animation that rotates and scales the replica.



```lua
local function HoverAnimationLeave(Object)
    if Object:FindFirstChild("AnimationObject") then
        TweenService:Create(Object,TweenInfo.new(0.5),{Rotation = Object.Rotation - 5}):Play()
        TweenService:Create(Object.AnimationObject,TweenInfo.new(0.5),{Size = Object.Size-UDim2.new(0,10,0,10),Rotation = Object.Rotation - 5})
        wait(0.5)
        Object.AnimationObject:Destroy()
        table.remove(ActiveHovers,Object)
    end
end
```

This function will be called one a player hovers off a UI object. This basically destroys the hover effect and resets the button’s rotation.



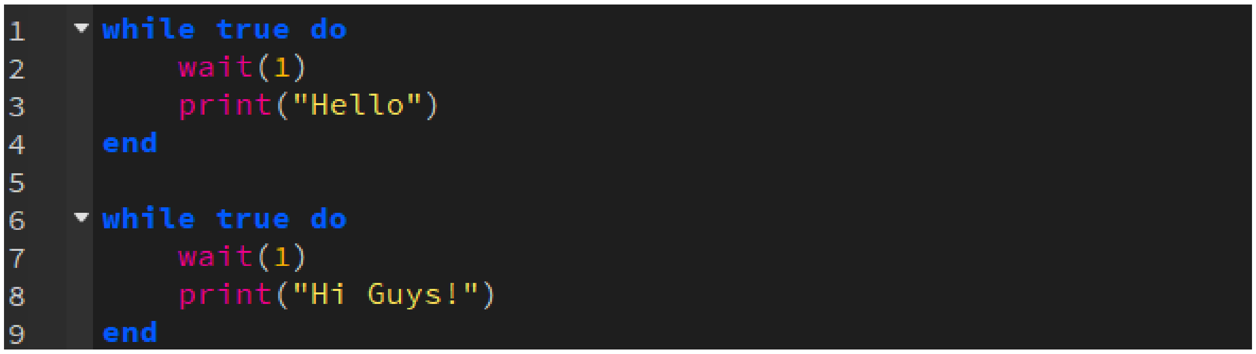
```lua
coroutine.create(function()
    while wait(0.1) do
        for _,Object in pairs(ActiveHovers) do
            if Object:FindFirstChild("AnimationObject") then
                coroutine.wrap(function()
                    TweenService:Create(Object,TweenInfo.new(0.5),{Rotation = Object.Rotation - 5}):Play()
                    TweenService:Create(Object.AnimationObject,TweenInfo.new(0.5),{Size = Object.Size-UDim2.new(0,10,0,10),Rotation = Object.Rotation
                    wait(0.5)
                    Object.AnimationObject:Destroy()
                    table.remove(ActiveHovers,Object)
                end)
            end
        end
    end
end)()
```

Here we see a coroutine. This coroutine has a loop in it. The reason why I wrote this is because when a player hovers over UI objects too fast, it doesn’t have time to call the “HoverAnimationLeave” function. This loop checks if a object still has the hovering effect, even if the player is not hovering on it anymore. If the loop detects something like that, it will create a new coroutine that removes the hovering effect from the object.

(2.4.3) What is a coroutine?

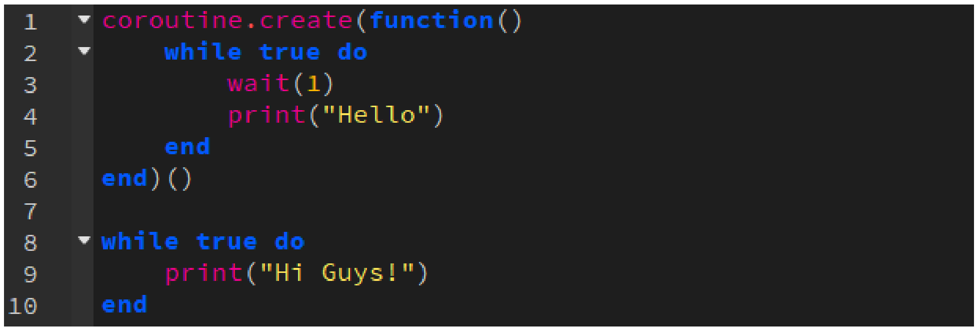
Coroutines are a programming concept that allows for multitasking, where a program or function can temporarily pause its execution, and then resume from where it left off later.

A coroutine is a special type of function that can be paused at any point during its execution, and then resumed later from where it left off. This makes it possible to write programs that can perform multiple tasks at once, without having to resort to complex threading or process management. For example, if you want to have two loops in a program, you can't just do it like this:



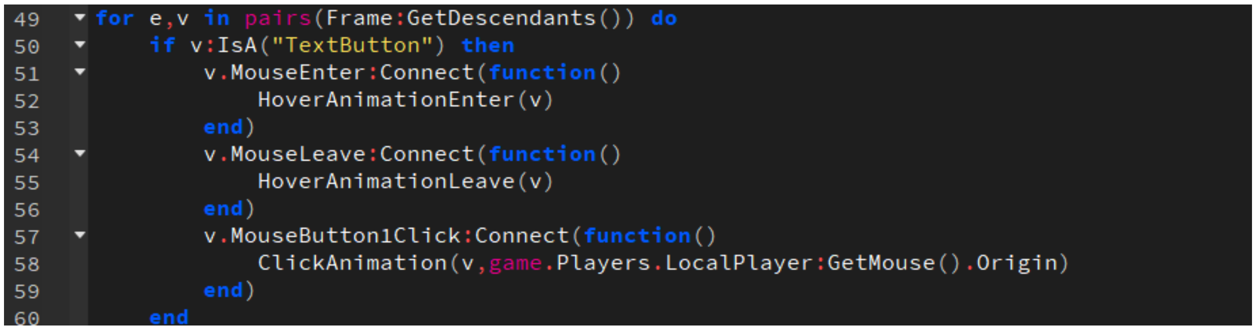
```lua
while true do
    wait(1)
    print("Hello")
end

while true do
    wait(1)
    print("Hi Guys!")
end
```

You would have to use a coroutine to execute both loops at a time. As I explained earlier, a coroutine is like a new script.



```lua
coroutine.create(function()
    while true do
        wait(1)
        print("Hello")
    end
end)()

while true do
    print("Hi Guys!")
end
```

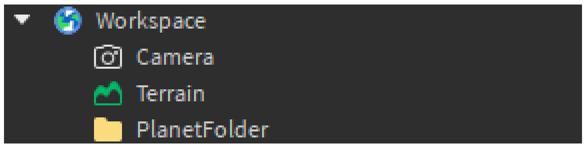
Now, I made a script that fires the server with the player's values when the player clicks the button.



```lua
for e,v in pairs(Frame:GetDescendants()) do
    if v:IsA("TextButton") then
        v.MouseEnter:Connect(function()
            HoverAnimationEnter(v)
        end)
        v.MouseLeave:Connect(function()
            HoverAnimationLeave(v)
        end)
        v.MouseButton1Click:Connect(function()
            ClickAnimation(v,game.Players.LocalPlayer:GetMouse().Origin)
        end)
    end
```

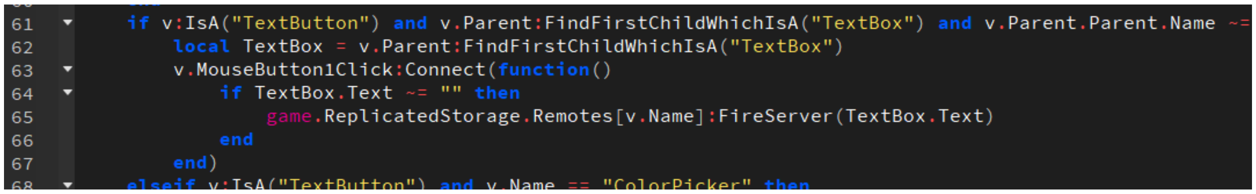
First, I made a code that loops through every child in the UI frame. I used the function "GetDescendants" to do this. In ROBLOX, every object (child) has a parent. Next, I wrote a block of code that calls the hover/click functions when a button is being hovered/clicked on.

(2.4.4) What are parents and children (in Lua)?



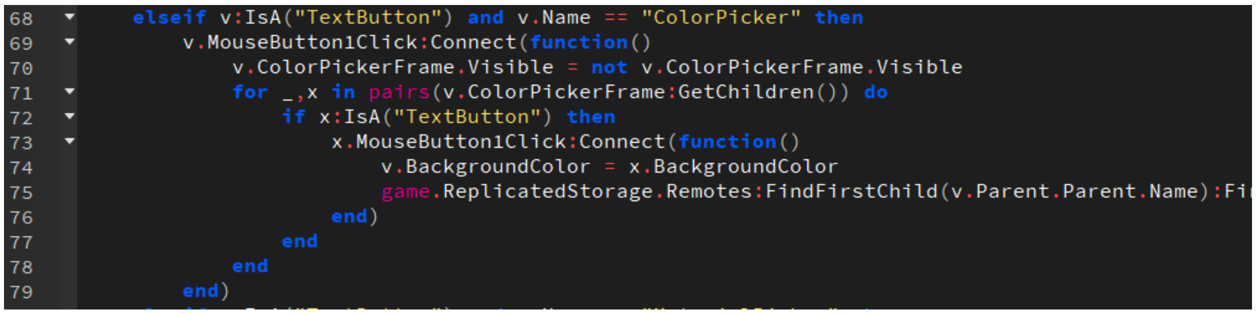
```
Workspace
    Camera
    Terrain
    PlanetFolder
```

Here, "Camera", "Terrain" and "PlanetFolder" are the children of "Workspace". So workspace is the parent of all these objects.



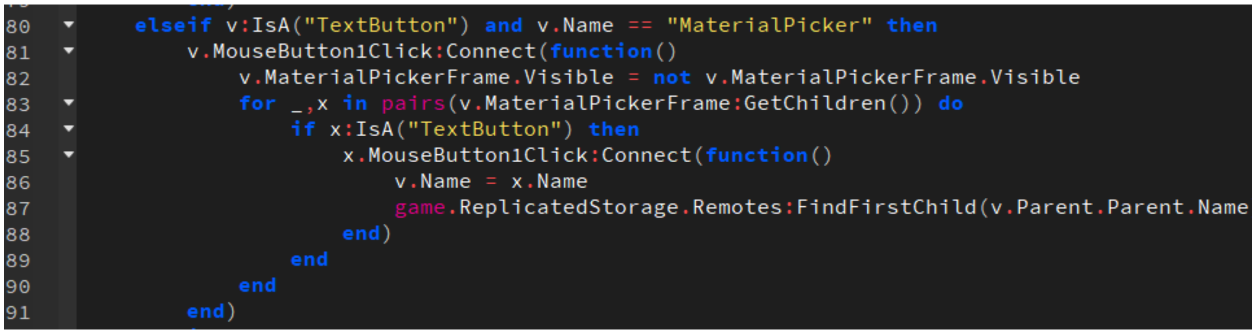
```lua
    if v:IsA("TextButton") and v.Parent:FindFirstChildWhichIsA("TextBox") and v.Parent.Parent.Name ~=
        local TextBox = v.Parent:FindFirstChildWhichIsA("TextBox")
        v.MouseButton1Click:Connect(function()
            if TextBox.Text ~= "" then
                game.ReplicatedStorage.Remotes[v.Name]:FireServer(TextBox.Text)
            end
        end)
    elseif v:IsA("TextButton") and v.Name == "ColorPicker" then
```

Here, it checks if the child is a button and if the button includes a text box. Then, once clicked, it will fire the server to apply the values of the button to the player's planet. It also checks if the child's name is not "RingGenerator", "ColorPicker" or "MaterialPicker". This was cut out in the screenshot.



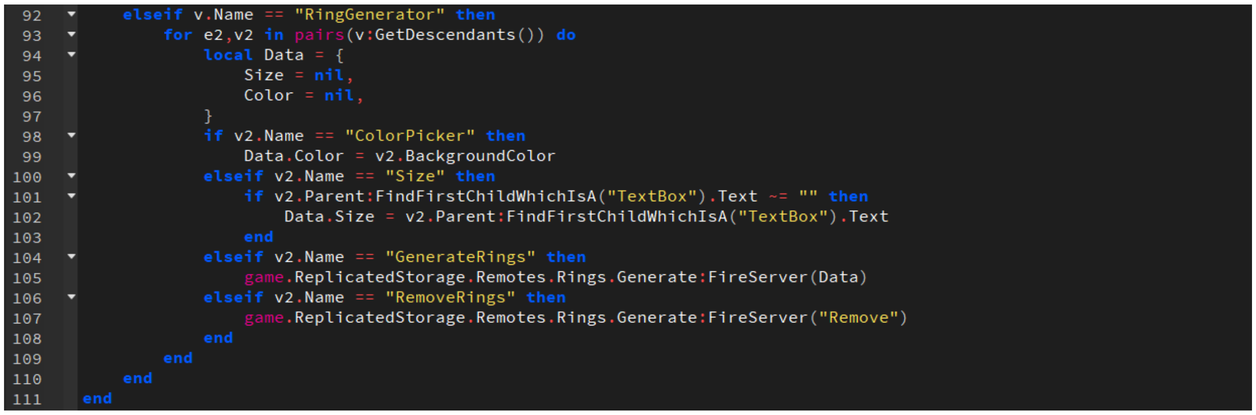
```lua
    elseif v:IsA("TextButton") and v.Name == "ColorPicker" then
        v.MouseButton1Click:Connect(function()
            v.ColorPickerFrame.Visible = not v.ColorPickerFrame.Visible
            for _,x in pairs(v.ColorPickerFrame:GetChildren()) do
                if x:IsA("TextButton") then
                    x.MouseButton1Click:Connect(function()
                        v.BackgroundColor = x.BackgroundColor
                        game.ReplicatedStorage.Remotes:FindFirstChild(v.Parent.Parent.Name):Fi
                    end)
                end
            end
        end)
```

Here, it checks if the button's name is "ColorPicker". If it's the case, it will run a function where it loops though all the colors of the color picker, and when the player clicks on one of these colors, it fires the server and applies the color to whatever category the player has chosen.



```lua
    elseif v:IsA("TextButton") and v.Name == "MaterialPicker" then
        v.MouseButton1Click:Connect(function()
            v.MaterialPickerFrame.Visible = not v.MaterialPickerFrame.Visible
            for _,x in pairs(v.MaterialPickerFrame:GetChildren()) do
                if x:IsA("TextButton") then
                    x.MouseButton1Click:Connect(function()
                        v.Name = x.Name
                        game.ReplicatedStorage.Remotes:FindFirstChild(v.Parent.Parent.Name
                    end)
                end
            end
        end)
```

This code functions the same as the color picker code, but the player chooses materials for the planet instead of colors.



```lua
    elseif v.Name == "RingGenerator" then
        for e2,v2 in pairs(v:GetDescendants()) do
            local Data = {
                Size = nil,
                Color = nil,
            }
            if v2.Name == "ColorPicker" then
                Data.Color = v2.BackgroundColor
            elseif v2.Name == "Size" then
                if v2.Parent:FindFirstChildWhichIsA("TextBox").Text ~= "" then
                    Data.Size = v2.Parent:FindFirstChildWhichIsA("TextBox").Text
                end
            elseif v2.Name == "GenerateRings" then
                game.ReplicatedStorage.Remotes.Rings.Generate:FireServer(Data)
            elseif v2.Name == "RemoveRings" then
                game.ReplicatedStorage.Remotes.Rings.Generate:FireServer("Remove")
            end
        end
    end
end
```

And here is the last part of the code, which handles the ring generator buttons.

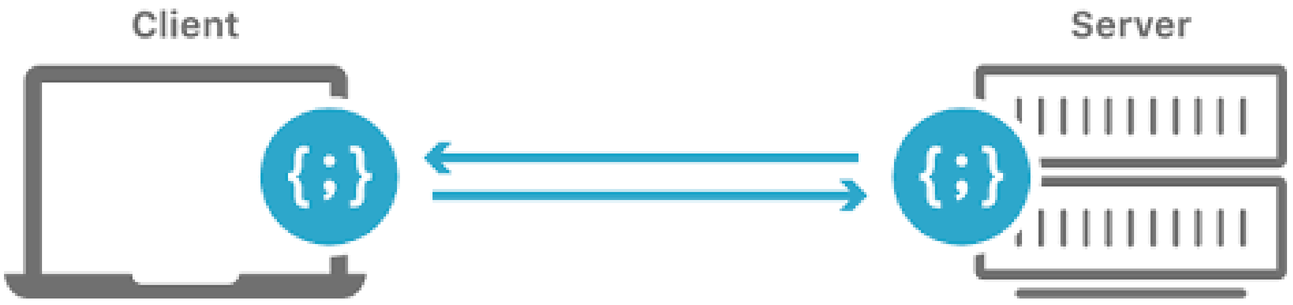
Now, it was time to program the server-side part of my game.

(2.4.5) Server-side and client-side, what’s the difference?

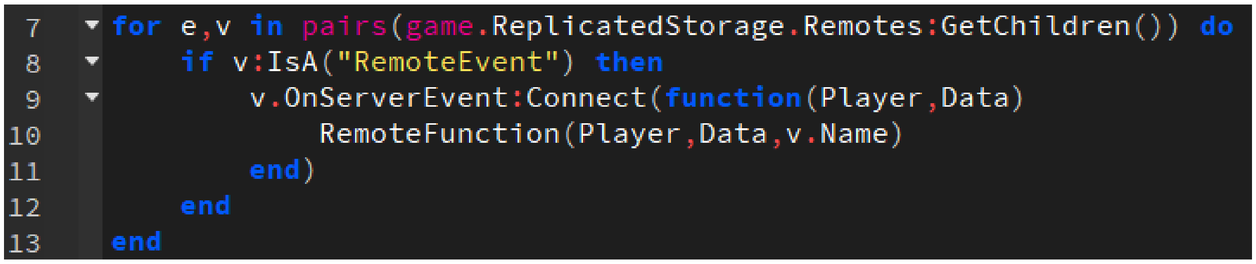
Server-side code is responsible for managing the game state, handling game logic, and communicating with other game servers or external services.

Client-side code, on the other hand, is responsible for rendering the game and providing the UI. Client-side code includes UI, sound effects, animations, and player input. Client-side code runs on the player's device and communicates with the server to receive updates about the game state.

If you made a game that relies solely on the client-side code, the whole game would only be visible to the client. You can also say “frontend” and “backend” for server-side and client-side.

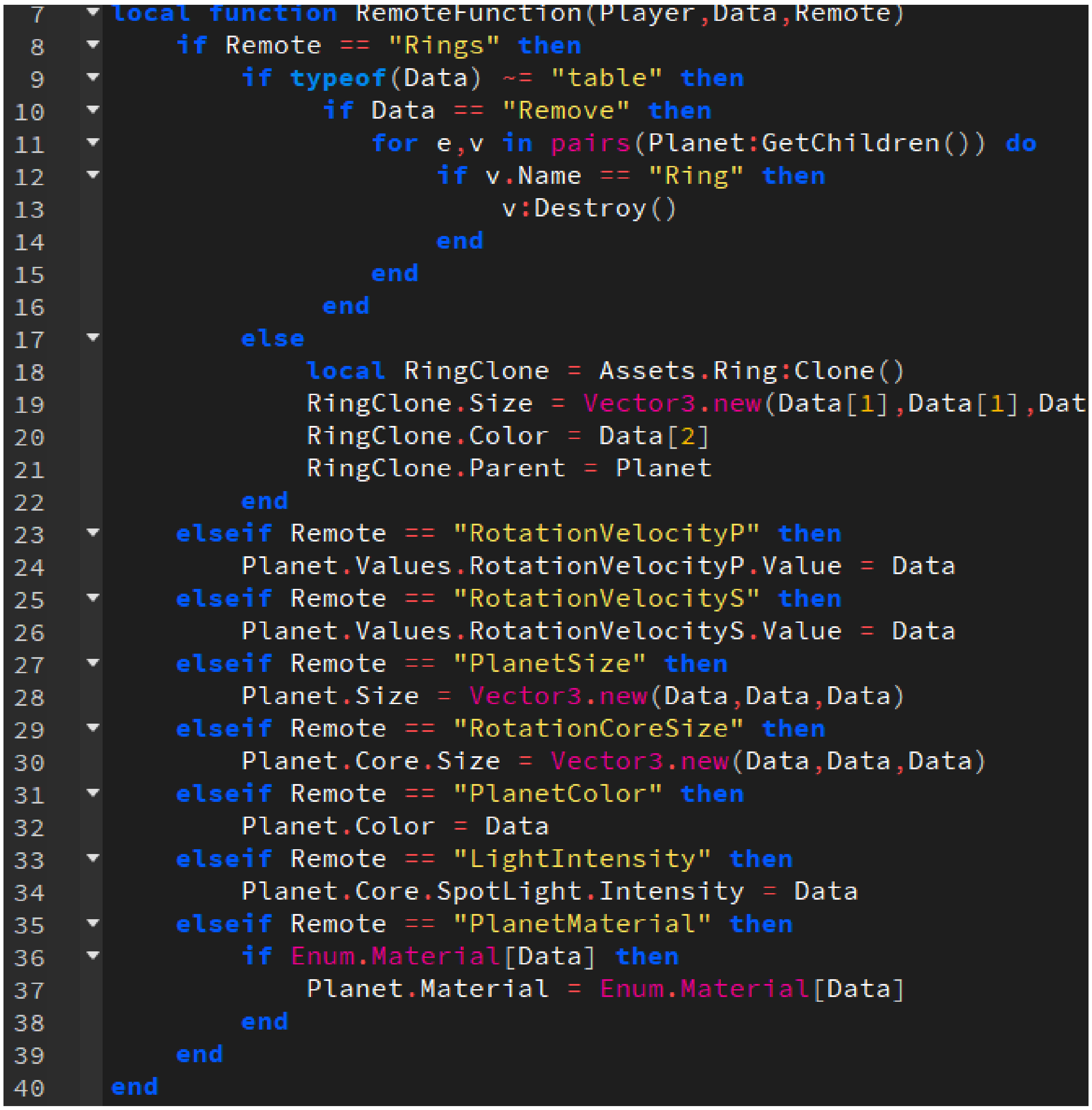


First I made a script that receives events from remotes that the client/player fired when clicking a button on the UI. For this, I made a brand-new script.



```
for e,v in pairs(game.ReplicatedStorage.Remotes:GetChildren()) do
    if v:IsA("RemoteEvent") then
        v.OnServerEvent:Connect(function(Player,Data)
            RemoteFunction(Player,Data,v.Name)
        end)
    end
end
```

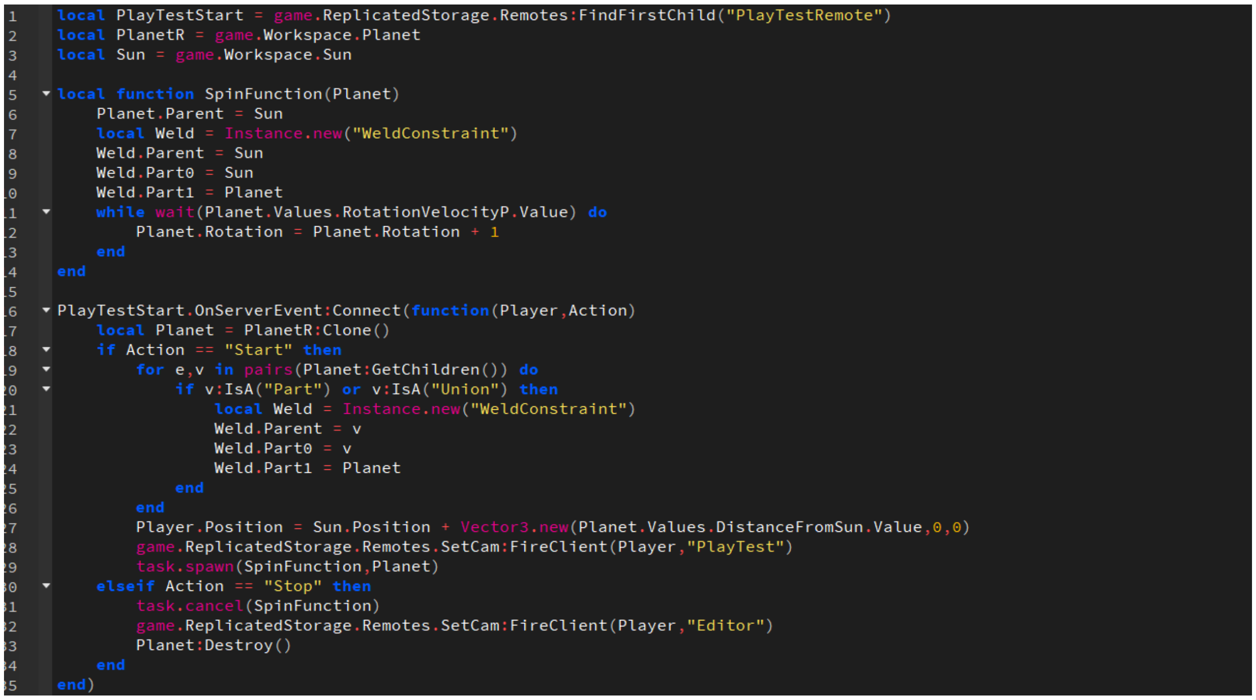
Now I made a function that applies the data that the client sent to the planet.



```lua
local function RemoteFunction(Player,Data,Remote)
    if Remote == "Rings" then
        if typeof(Data) ~= "table" then
            if Data == "Remove" then
                for e,v in pairs(Planet:GetChildren()) do
                    if v.Name == "Ring" then
                        v:Destroy()
                    end
                end
            end
        else
            local RingClone = Assets.Ring:Clone()
            RingClone.Size = Vector3.new(Data[1],Data[1],Dat
            RingClone.Color = Data[2]
            RingClone.Parent = Planet
        end
    elseif Remote == "RotationVelocityP" then
        Planet.Values.RotationVelocityP.Value = Data
    elseif Remote == "RotationVelocityS" then
        Planet.Values.RotationVelocityS.Value = Data
    elseif Remote == "PlanetSize" then
        Planet.Size = Vector3.new(Data,Data,Data)
    elseif Remote == "RotationCoreSize" then
        Planet.Core.Size = Vector3.new(Data,Data,Data)
    elseif Remote == "PlanetColor" then
        Planet.Color = Data
    elseif Remote == "LightIntensity" then
        Planet.Core.SpotLight.Intensity = Data
    elseif Remote == "PlanetMaterial" then
        if Enum.Material[Data] then
            Planet.Material = Enum.Material[Data]
        end
    end
end
```

Here it checks if the data exists and applies the requested data to the planet.

Finally, the last thing to add was the “play test”. In the play test mode, you could see your planet rotating around the sun.



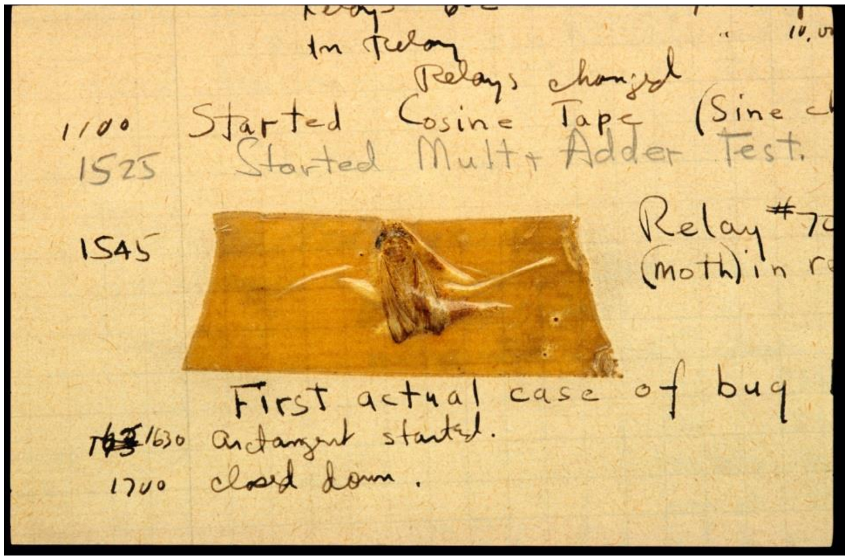
```lua
local PlayTestStart = game.ReplicatedStorage.Remotes:FindFirstChild("PlayTestRemote")
local PlanetR = game.Workspace.Planet
local Sun = game.Workspace.Sun

local function SpinFunction(Planet)
    Planet.Parent = Sun
    local Weld = Instance.new("WeldConstraint")
    Weld.Parent = Sun
    Weld.Part0 = Sun
    Weld.Part1 = Planet
    while wait(Planet.Values.RotationVelocityP.Value) do
        Planet.Rotation = Planet.Rotation + 1
    end
end

PlayTestStart.OnServerEvent:Connect(function(Player,Action)
    local Planet = PlanetR:Clone()
    if Action == "Start" then
        for e,v in pairs(Planet:GetChildren()) do
            if v:IsA("Part") or v:IsA("Union") then
                local Weld = Instance.new("WeldConstraint")
                Weld.Parent = v
                Weld.Part0 = v
                Weld.Part1 = Planet
            end
        end
        Player.Position = Sun.Position + Vector3.new(Planet.Values.DistanceFromSun.Value,0,0)
        game.ReplicatedStorage.Remotes.SetCam:FireClient(Player,"PlayTest")
        task.spawn(SpinFunction,Planet)
    elseif Action == "Stop" then
        task.cancel(SpinFunction)
        game.ReplicatedStorage.Remotes.SetCam:FireClient(Player,"Editor")
        Planet:Destroy()
    end
end)
```

Here I made a script that receives data if the player wants to start the play test mode. Then it sets up the planet and sun and makes it rotate with the player’s customized values.

2.5 – Bug fixes

<u>(2.5.1) What are “bugs”?</u>

A bug is a flaw or glitch in a system. Programmers often call errors in their code bugs. But why do we call errors “bugs”? The term "bug" was first used in the context of computing by Grace Hopper in 1947, when she found a moth trapped in a relay in the Harvard Mark II computer. The moth caused a problem in the computer's operation, and Hopper removed it and taped it to a logbook, noting that they had "debugged" the system.



in relay

Relays changed

1100 Started Cosine Tape (Sine c

1525 Started Mult+ Adder Test.

1545 Relay #70 (moth) in r

First actual case of bug

~~1630~~ 1630 Arctangent started.

1700 closed down.

Here we can see the moth that is taped to the logbook. Since then, an error has been called a bug by programmers.

The first bug I encountered was when the player generated a ring after the play test, the ring would not be in place. This was because the planet had changed positions after the play test. An easy fix to this problem was simply adding this line to the code where it generates the ring:

```
RingClone.Position = Planet.Position
```

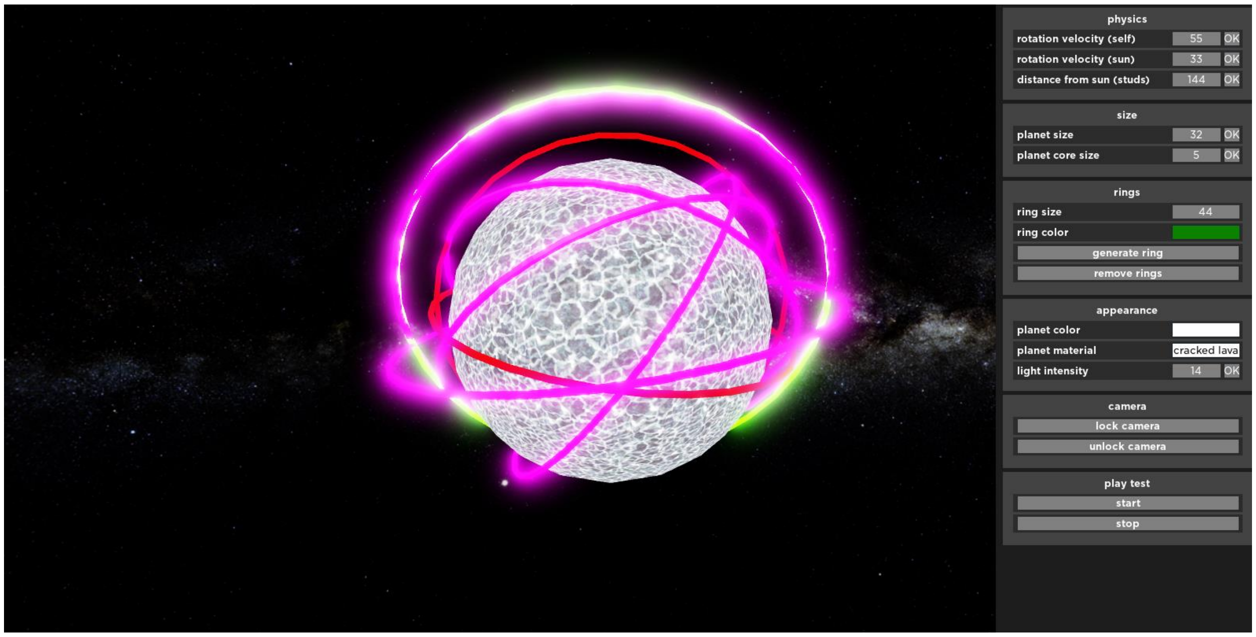
The next bug I encountered was in the play test mode. The planet wouldn't rotate around the sun. It would only spin around itself. This was because I had forgotten to add a function that makes the planet rotate around the sun.

```
        Planet.Rotation = Planet.Rotation + 1
    end
end

local function SpinFunction2(Planet,Sun)
    while wait(Planet.Values.RotationVelocirtS.Value) do
        Sun.Rotation = Sun.Rotation + 1
    end
end
```

I made a new function in the play test script that makes the planet rotate around the sun. The rest of the game was bug-free 😊

3. The end

3.1. Conclusion


physics

rotation velocity (self) 55

rotation velocity (sun) 33

distance from sun (studs) 144

size

planet size 32

planet core size 5

rings

ring size 44

ring color 🟩

appearance

planet color ⬜

planet material cracked lava

light intensity 14

There we go! The game is finally done. I had a lot of fun making this TraPe!

I learnt a lot about the programs I have used and about LUA. But I realized that ROBLOX studio is not the best game engine I could have used. It's good for beginners, but I could have used something more advanced like unreal engine or CryEngine.

3.2. Sources

https://www.lua.org/docs.html

https://create.roblox.com/docs

https://stackoverflow.com/

https://google.com/

https://wikipedia.com/

https://tutorialspoint.com/

Download links:

VS Code: https://vscode.dev | https://code.visualstudio.com/

ROBLOX Studio: https://roblox.com/create

Blender: https://www.blender.org/download/

Lua: https://www.lua.org/download.html

Download my game here:
https://mediafire.com/download/9GGh2VzhOX0rha