ROBLOX SUPPLEMENTAL

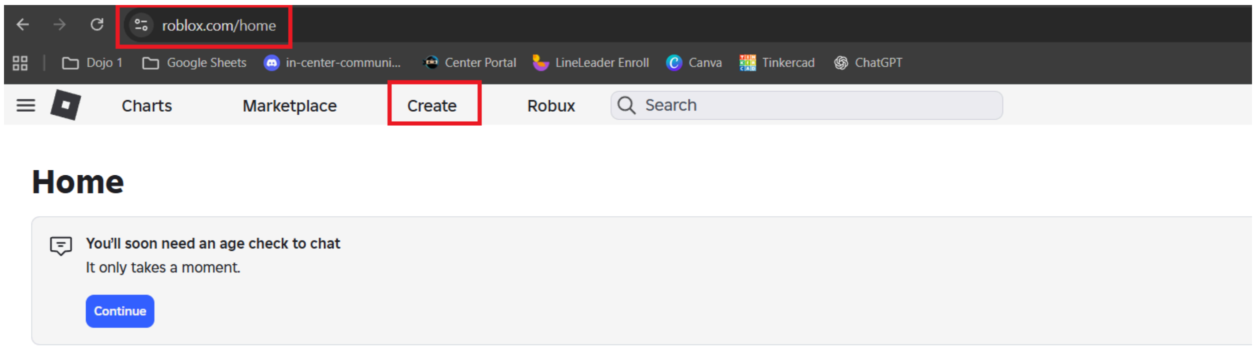
Roblox Video Game Development

Programming Language: Lua scripting

Tool: Roblox Studio

Chapter 0. Opening Roblox Studio

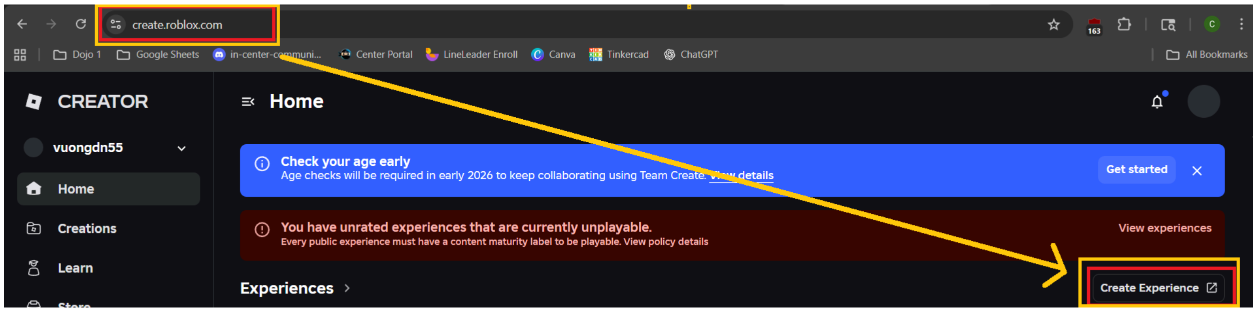
Task: Sign into your account by visiting https://www.roblox.com/. Next, click on "**Create**" that is found at the top.



Home

You'll soon need an age check to chat
It only takes a moment.

Task: Then, click on "**Create Experience**" that is found on the right side.

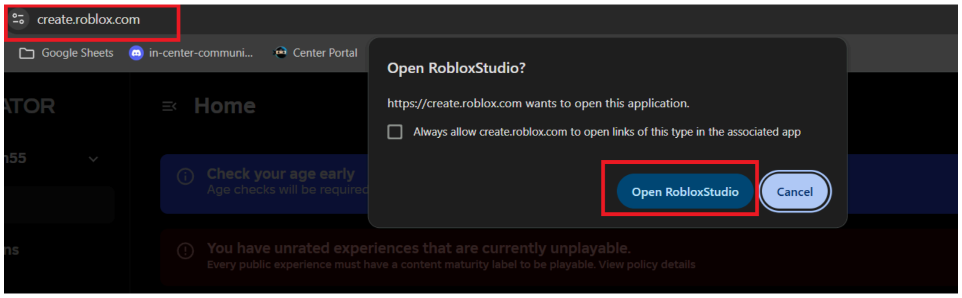


Home

Check your age early
Age checks will be required in early 2026 to keep collaborating using Team Create. View details

You have unrated experiences that are currently unplayable.
Every public experience must have a content maturity label to be playable. View policy details

An alert will appear. **Task:** Click on "**Open Roblox Studio**". **Give Roblox Studio time to load**.

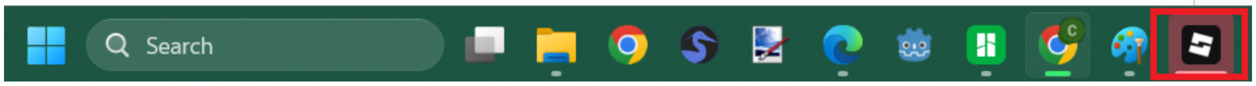


Open RobloxStudio?

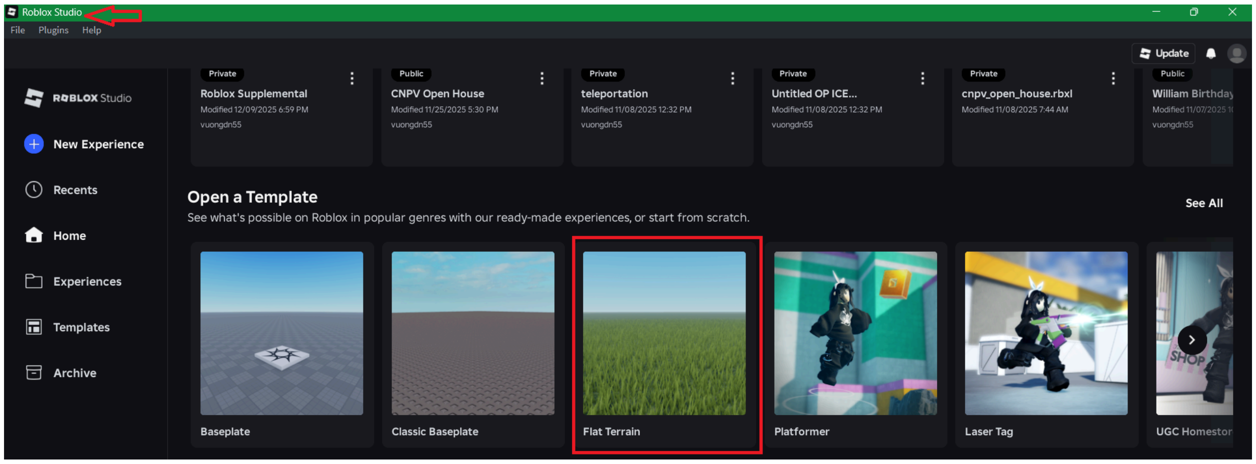
https://create.roblox.com wants to open this application.

☐ Always allow create.roblox.com to open links of this type in the associated app

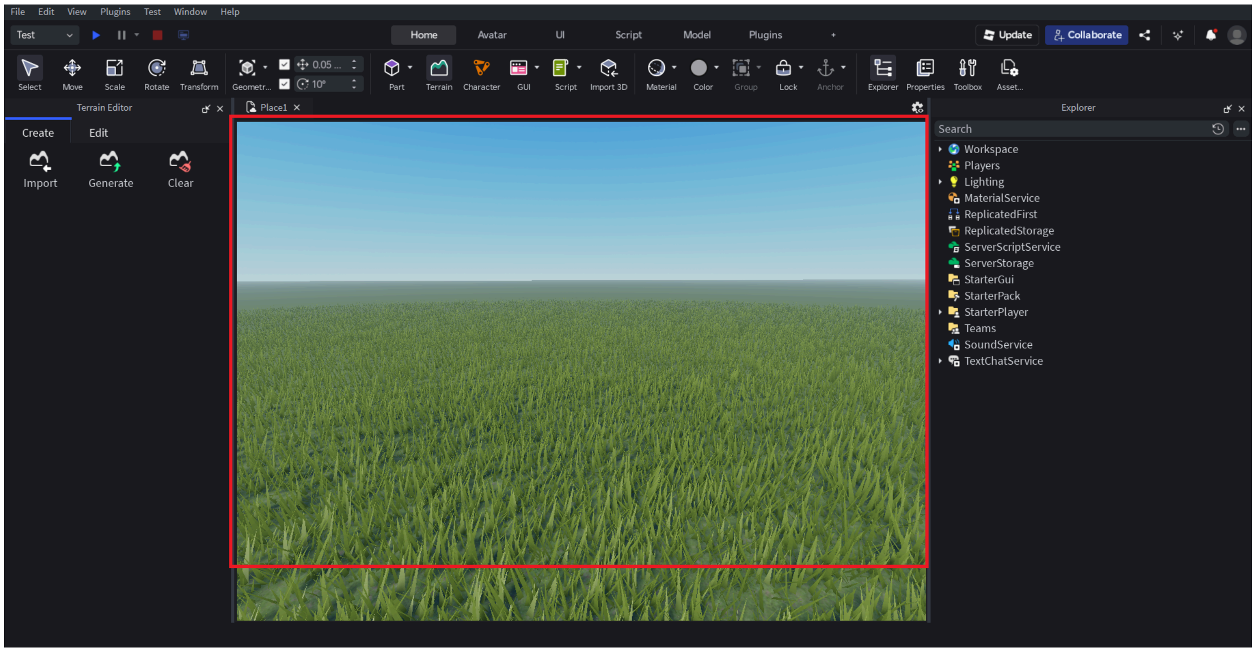
Once fully loaded, the Roblox Studio icon will appear on the tool bar at the bottom (see red square). **Task:** Click on this Roblox Studio icon.



Task: Once in Roblox Studio, scroll down until "**Flat Terrain**" is found (see red square). Click on this "**Flat Terrain**".



The “**Flat Terrain**” world will be loaded and a grassy area will be shown. See below.

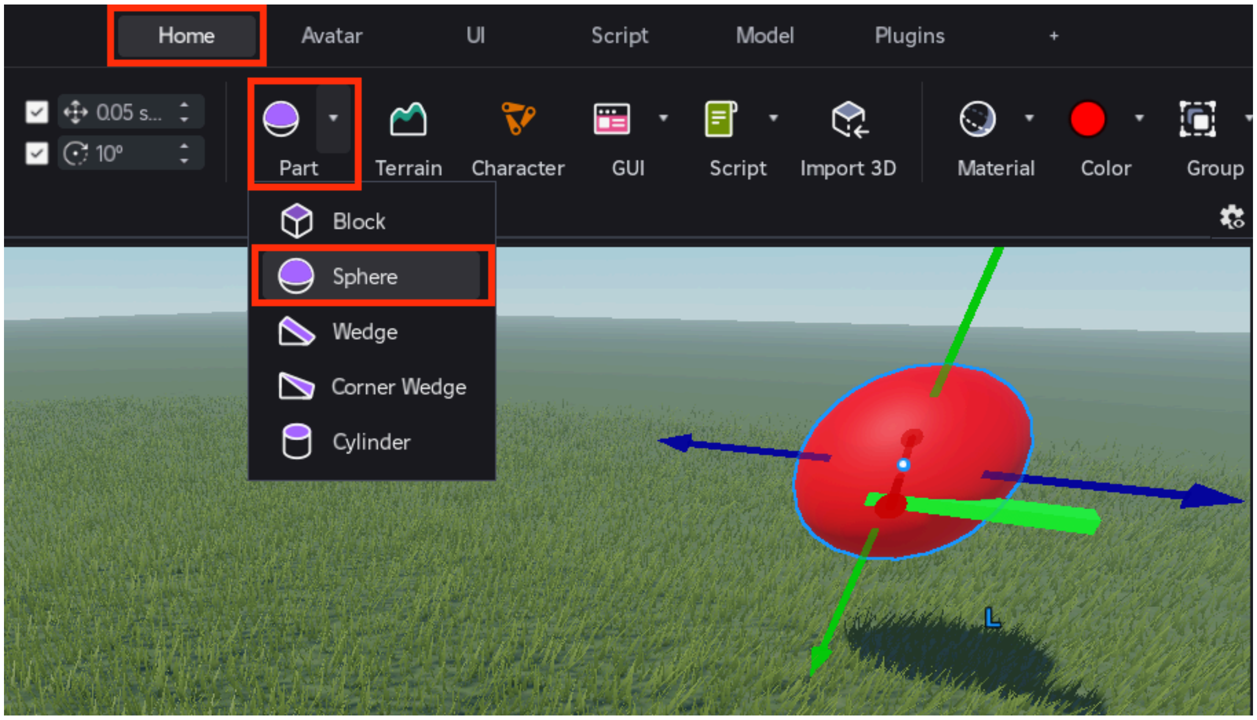


Chapter 1. Camera Keyboard Controls

```
a, Left Direction Key  - move camera left
d, Right Direction Key - move camera right
w, Up Arrow Key        - move camera forward
s, Down Arrow Key      - move camera backwards
q - move the camera down
e - move the camera up
i - zoom in
o - zoom out
```

Chapter 2. Insert A Part.

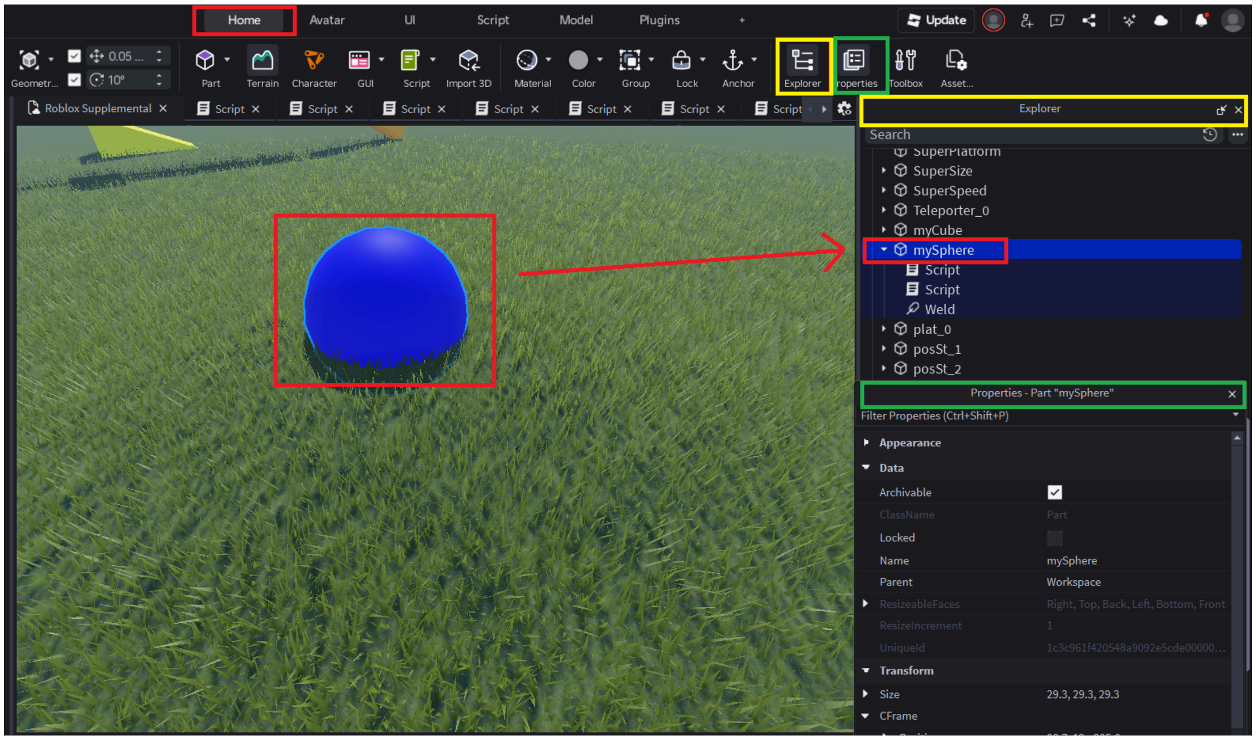
Task: Insert a sphere into your game



Chapter 3. Explore, Properties, and Rename A Part

Task: 3.1. Put the mouse cursor over to the sphere and click the left button of the mouse. **This will make the sphere the active item.**

Task: Press F2 and then change its name to "**mySphere**".



- SuperPlatform
- SuperSize
- SuperSpeed
- Teleporter_0
- myCube
- mySphere
  - Script
  - Script
  - Weld
- plat_0
- posSt_1
- posSt_2

Appearance	
Data	
Archivable	✓
ClassName	Part
Locked	
Name	mySphere
Parent	Workspace
ResizeableFaces	Right, Top, Back, Left, Bottom, Front
ResizeIncrement	1
UniqueId	1c3c961f420548a9092e5cde00000...
Transform	
Size	29.3, 29.3, 29.3
CFrame	

Chapter 4. Authoring - Clipboard

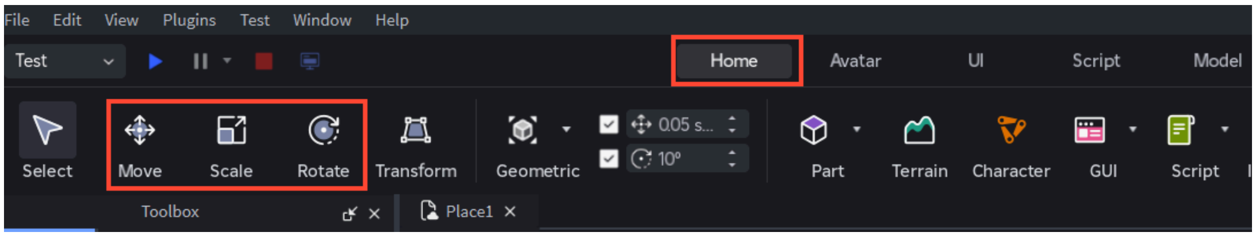
Make sure that the sphere is the active item.

Task: Cut - hold down the “ctrl” button on the keyboard with one hand. Next, with the other hand, press “x” button on the keyboard just once.

Task: Copy - create a copy or clone of the selected object. Hold down the “ctrl” button on the keyboard with one hand. Next, with the other hand, press the “c” button on the keyboard just once.

Task: Past - press the “v” button on the keyboard just once.

Chapter 5. Ribbon Panel called "Home" (look up to the top of the navigation)



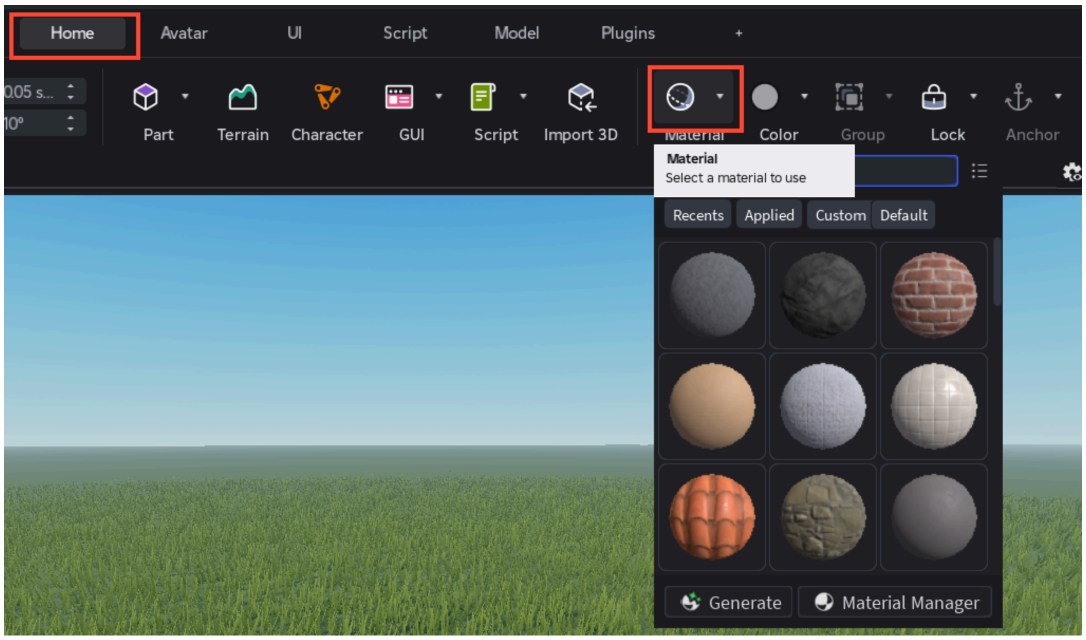
5.1. Object Manipulation

Move - position an object

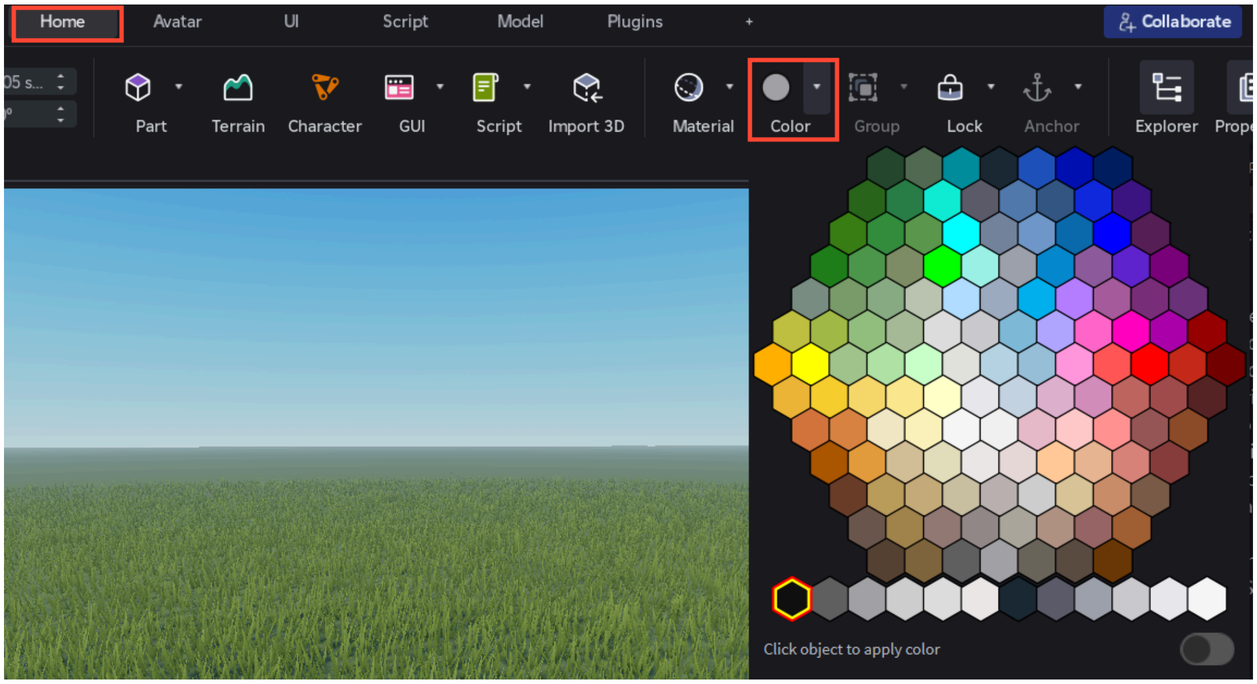
Scale - change the object's size

Rotate - rotate on an axis

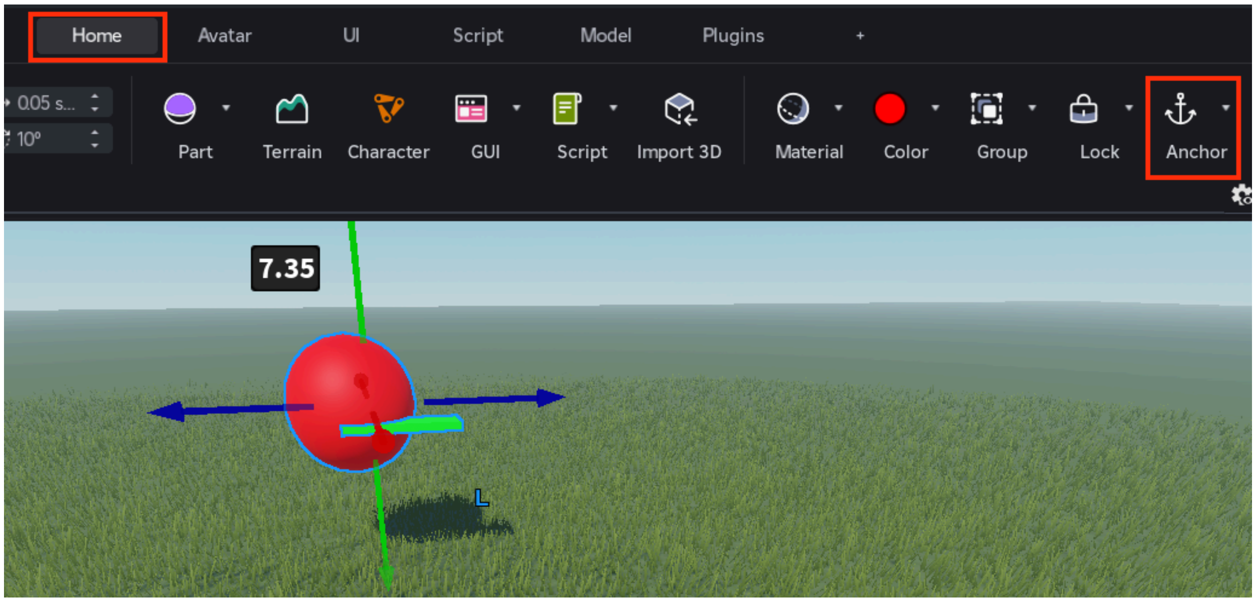
Task: 5.2. Material. Change the material composition of the active object



Task: 5.3. Color. Change the color of the active object



Task: 5.4. Anchor. Place the active object permanently in a location - this allows a part to float in the air

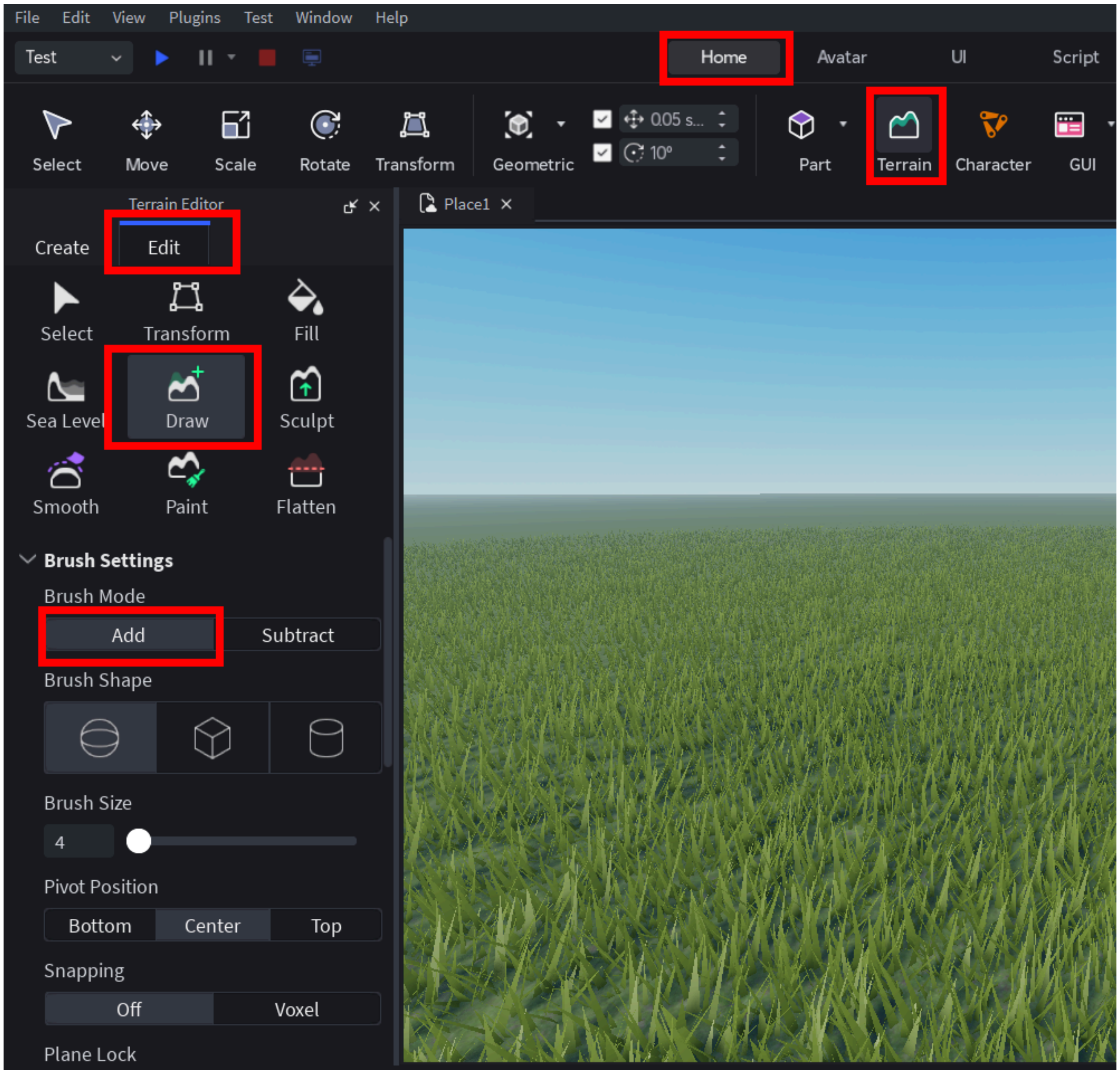


Chapter 6. Terrain Editor.

Task: This tool is used to create your own Roblox world. Look at the image and click on all of the areas that are boxed in red.

Task: Move the mouse pointer to the grassy area. Click and hold down the left button of the mouse to apply the biome.

Task: Click and hold down the right button of the mouse to rotate the camera



6.3. Add will insert a biome at location of the mouse click

6.4. Subtract will remove the biome at the location of the mouse click

6.5. Paint. This allows us to paint snow on top of the mountain

6.6. Smooth a. level a depression, hill or mountain. For example, if we have a depression, smoothing it brings the depression up OR, if we have a hill, smoothing it brings it down This end result of using this feature is that the location of the mouse click becomes a flat plain

6.7. Generate Select one or multiple biomes and add them into the game

Chapter 7. Composition & Creation

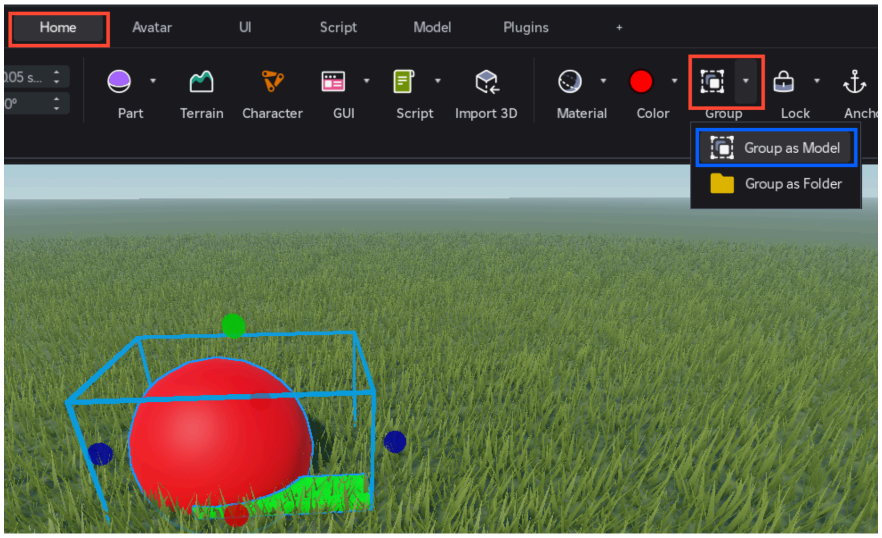
Composition means to combine or merge together different entities. For us, we can compose objects by using group and ungroup controls.

Task: Insert a cube into the video game. Move the cube so that half of it is within the sphere while the other half is sticking out of the sphere.

Task: To combine different objects together, click and hold down the left button of the mouse. Next, move the mouse cursor and you will see a red box. Move the mouse cursor so that the red box includes all of the parts that are to be merged together.

All entities that are to be combined will have their outline lit up

Task: hold down the “ctrl” button and then click on the “g” key of the keyboard. This will combine all of the entities together into one new object.



3. to **ungroup**, hold down the “ctrl” button and then click on “u” to ungroup

Composition is important because it allows us to create new parts by using the basic, primitive parts (wedge, sphere, cube, and cylinder).

By combining parts, we are now able to create custom components to make our game unique

Task: **Challenge** - Show To A Sensei
a. create the Batcave, Avengers Fortress, or anything you want by grouping parts together

Task: **Super Challenge** - Show To A Sensei
a. create table and couch using composition of parts
b. group the parts together so that the entire table moves when it is being dragged by the mouse cursor

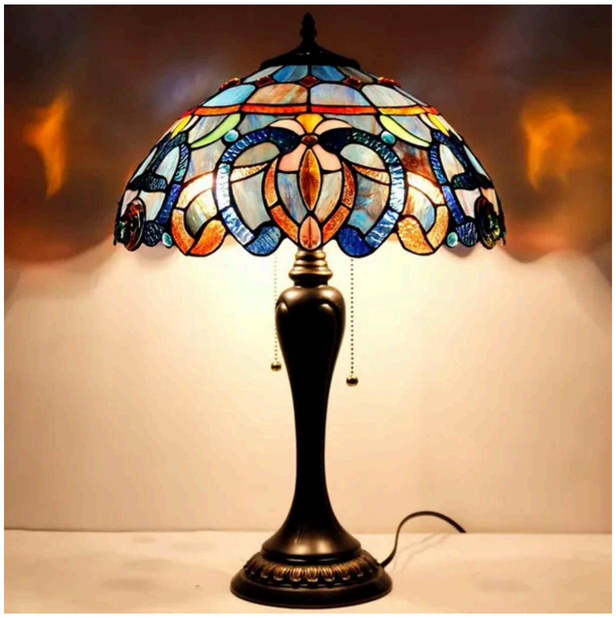
7.1. Group via Ctrl G. Composition of parts into a group

7.2. Ungroup via Ctrl U

Break up the group composition and return to individual parts

Task: **7.3. Challenge.** Make an arm chair

Task: **7.4. Super Challenge -** Make the Tiffany’s Lamp below by using composition. Try as best as you can to recreate the above in Roblox Studio.

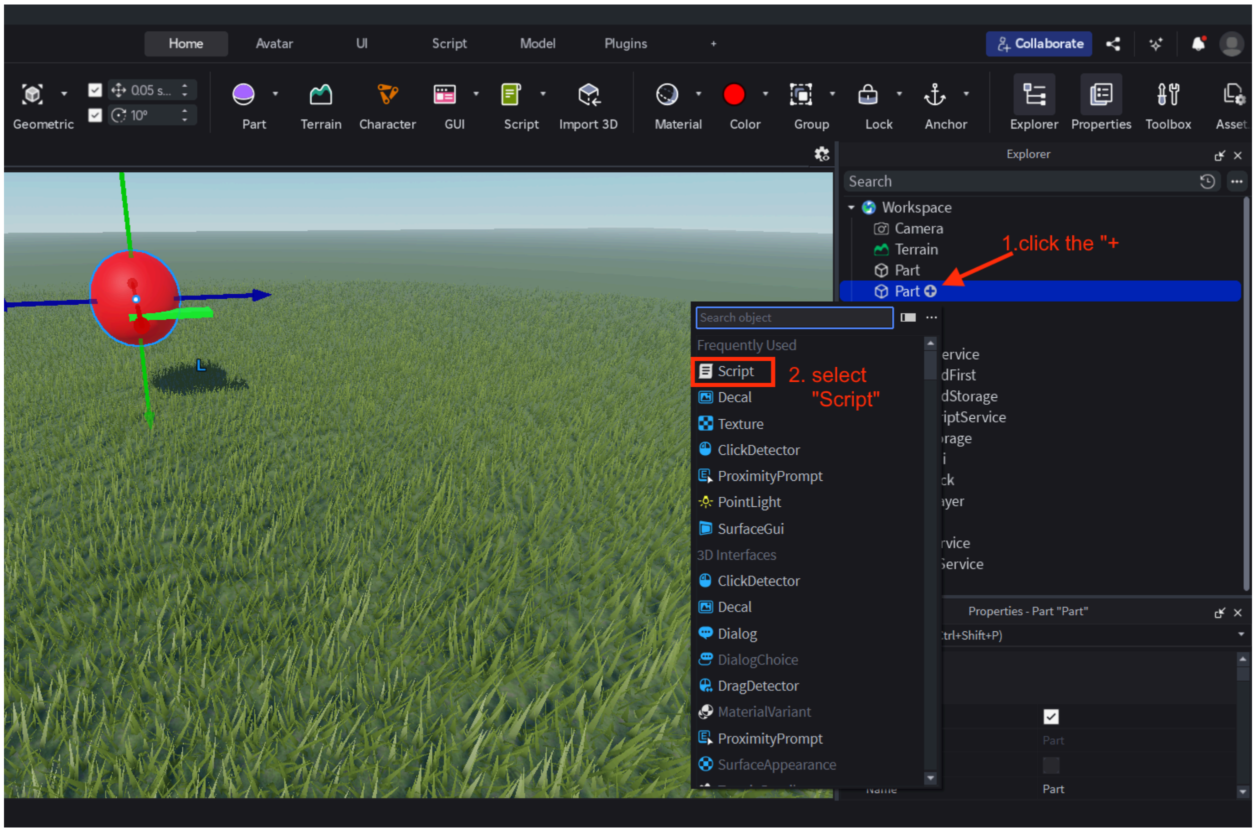


Chapter 8. Introduction to Programming

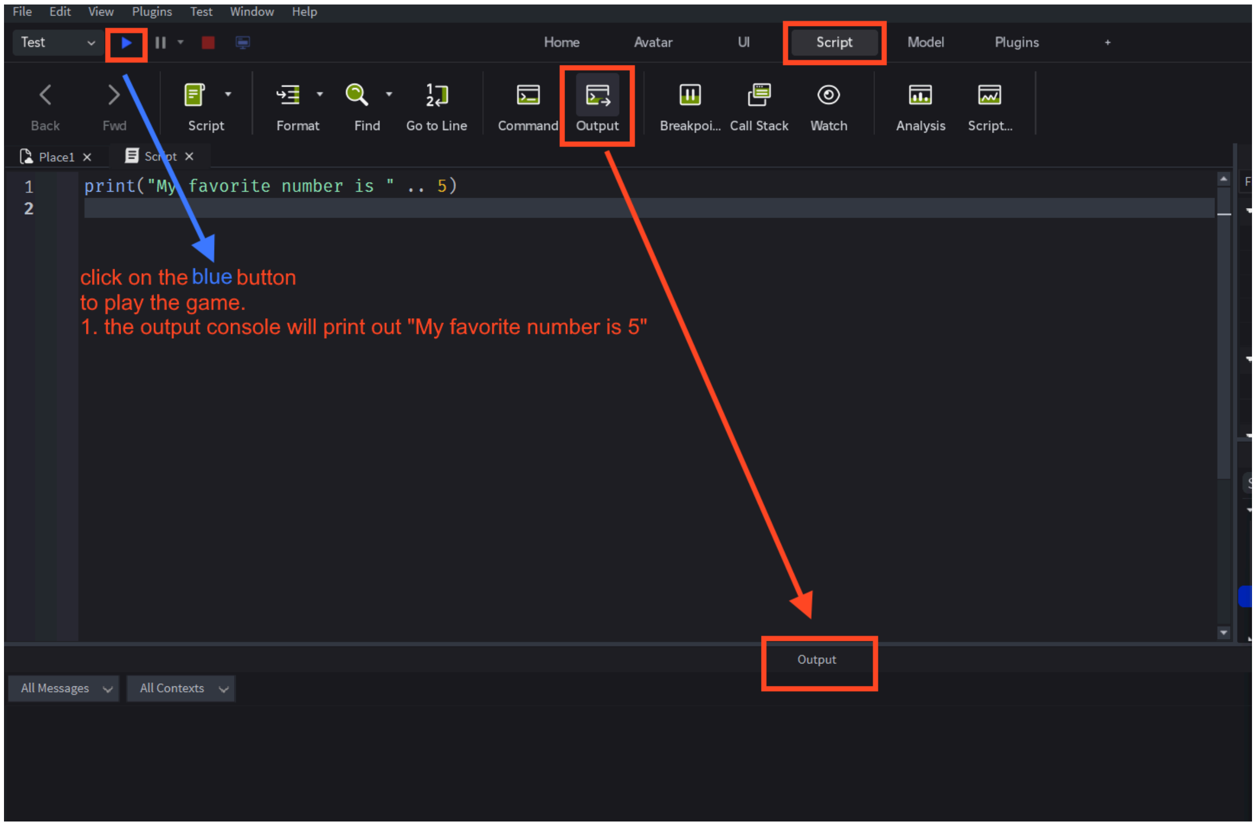
What is programming? Programming is writing characters in a structured and organized way that makes sense to the computer. The structure and organization creates an instruction and a computer is a machine that execute instructions

Task: 8.1. Add A Script To Your Game

8.1.1. find the sphere that is named "`mySphere`" and left click on the mouse to select your part. To the right of the part, you will find a plus symbol (+). Click on this plus and type in "Script"
8.1.1.1. **do not select "LocalScript", that is something else**



Task: **8.2. Show The Output Console.** The output console will allow us to see the result of our **`print()`** statement.



```
print("My favorite number is " .. 5)
```

click on the blue button
to play the game.
1. the output console will print out "My favorite number is 5"

8.3. Print Statement

The `print()` statement will output any custom message that we would like to be shown onto the output console. The `print()` statement is helpful because it can be used to help us debug.

The print statement allows us to print any custom message to the Output console. Task: Write the print statement below into the script.

```
print( "My favorite number is " .. 5 )
```

The **..** from above is called the concatenation operator. It is used to combine two things together.

Chapter 9. Basic Roblox Programming

A game is composed of actions and these actions are used to manipulate property values. The actions are implemented via functions and property values are implemented as storage elements.

9.1. Local Variables

We declare a local variable by using the reserve word "local" followed by the name of the variable. Next, we assign data to the variable by using the assignment operator, which is the single equal sign

NOTE: the word "var" is not needed
NOTE: Roblox does not need a delimiter to end a line of code.

```
local myPart = 0
```

9.2. Function Definition and Function Call

9.2.1. A function definition in the Roblox programming language requires an "end". This tells the computer that the function definition has ended.

9.2.2. In most programming languages, we group together lines of code by using curly braces { }. However, in Roblox, we do not need to use curly braces. Instead, the computer will search for the reserve word "end". We start a function definition by using

```
local function sayHello()
      print( "my name is Vuong, my favorite color is " .. 5 )
end
```

The beginning is the "`local function sayHello()`".
The reserve word "`end`" signals the end

Lines of code in between the beginning and end will be grouped together by the computer.

Task: **Challenge 1** Inside your function definition, write the code to print your name.

9.2.1. Function Call To Use The Function

In order to run the function definition, we now have to call its name. **After the "end" statement of your function definition**, call the name of your function by writing the name of the function definition and then open and closing parenthesis.

```
sayHello()
```

9.2.2. Full script

It should be noted that the function call should be written after the function definition.
Task: Erase the code from step 8.3 and write the code below instead. Play the game by pressing F5, look at the output console and confirm the message is correct.

```
- function definition
local function sayHello()
     print( "my name is Vuong, my favorite color is " .. 5 )
end

- function call
sayHello()
```

Chapter 10. Passing Data To Function Definition

We can make our print statement unique by passing data to the function definition. To pass data to the function definition, we need to put data between the open and closed parenthesis.

In the example below, the parameter is "`Code Ninjas PV`". This data is put into a variable called `nameVar`. It is important to give data a name because doing so means that the function definition can access the data by referencing the name of the variable that holds the data. The function definition then accesses the parameter in the body of the function definition. See below.

```
-- function definition
local function sayHello( nameVar )
     print( "Hi, my name is " .. nameVar )
end

-- function call
sayHello( "Code Ninjas PV" )
```

Task: Challenge

Write the code above but change the data that is passed to the function definition. We do not want to pass "`Code Ninjas PV`" to the function definition. Instead, replace it with your name.

Press F5 and check that the code does print your name in the output console.

Chapter 11. Looping

What if we want to execute a block of code forever? In this case, we use an infinite loop. The `while` loop can be used to ensure a block of code continues to run. This can be used to create a time delay. An infinite loop can be created by writing the following code

```
while true do

end
```

Chapter 12. Delay

What if we want to create a delay? Then we use the `wait()` function.

*** **NOTE:** always put a number inside the open and closed parenthesis.
*** The time unit for a `wait()` function is seconds.

For example, `wait(1)` means to wait 1 second before proceeding to the line of code after the `wait(1)`

For us, we will create a 1 second delay using `wait(1)`

Chapter 13. Random Number Generator

There are times where we want to pick a random number to serve as our delay. To prevent repeating numbers, we must create a seed. We can do this by writing

```
math.randomseed(tick())
```

Once our seed is set, we then use math.random(100) to select a single random number between 1 and 100

```
math.random(100)
```

Task: Challenge

Finish the function definition below that will print out a random number every 2 seconds.
Fill in the code for steps 0, 1, 3, and 4

When the game is played, the output console will show a random number every 2 seconds.

```
print("Hello world!")

-- step 0. start by create a seed generator

-- function definition
local function loopyRandom(tDelay)

     -- step 1. create a forever loop. All steps
     --         below will go INSIDE the forever loop

          -- step 2. create a delay
          wait(tDelay)


          -- step 3. create a variable named ranNum. Next, use
          --         math.random(100) to load a random number
          --         into ranNum


          -- step 4. print the value of the variable ranNum


end

-- function call
loopyRandom(2)
```

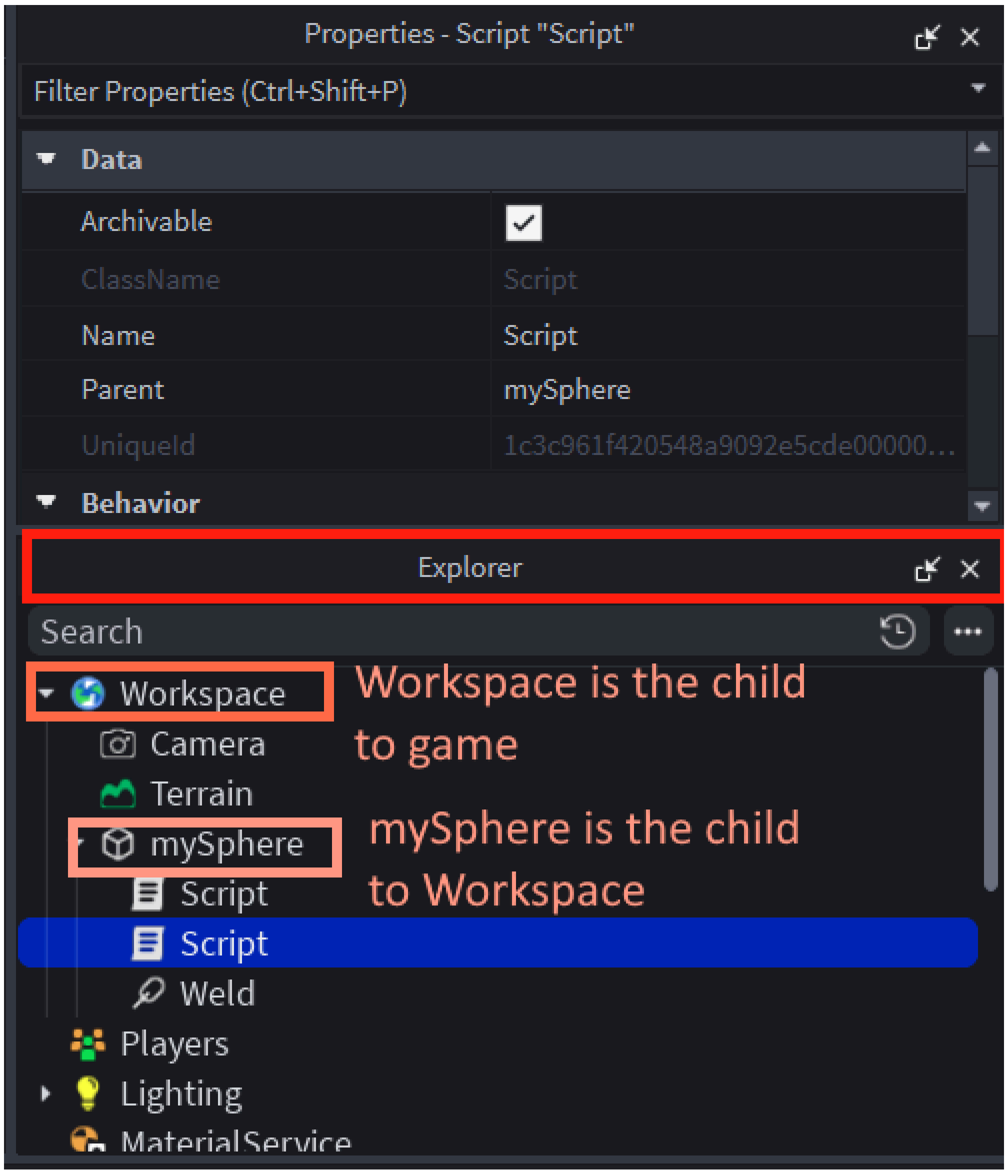
Chapter 14. Hierarchy

A Roblox game is based on a hierarchy and this means that the entities within a Roblox game are organized as a tree structure.

The root (or the top) node is the game itself and then comes the Workspace. We can use this tree structure to our advantage because it can be used to access an entity within the game.

For example, we can write the code below to refer to mySphere:

```
game.Workspace.mySphere;
```


Properties - Script "Script"
Filter Properties (Ctrl+Shift+P)
Data
Behavior
Explorer
Search
Workspace
Camera
Terrain
mySphere
Script
Script
Weld
Players
Lighting
MaterialService
Workspace is the child to game
mySphere is the child to Workspace

Archivable	✓
ClassName	Script
Name	Script
Parent	mySphere
UniqueId	1c3c961f420548a9092e5cde00000...

Chapter 15 Event programming

In the previous challenge, the function definition "`loopyRandom()`" will run when the Roblox game starts.

This is important because there are situations where we want a function to run when the game starts.

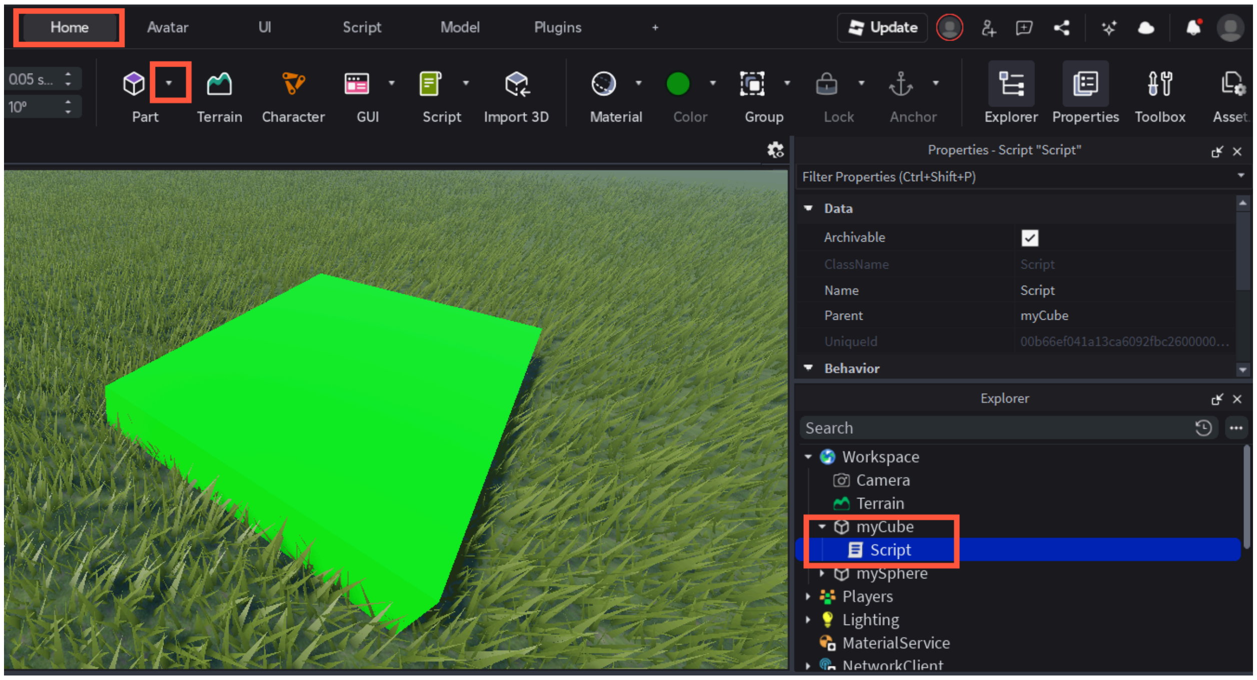
However, what if we want to execute a function ONLY when something happens? This "something happens" is called an event and events are **VERY IMPORTANT** to game development.

An event indicates that something has happened. We need an alert to signal to get our attention. Once we receive the alert, then we can act.

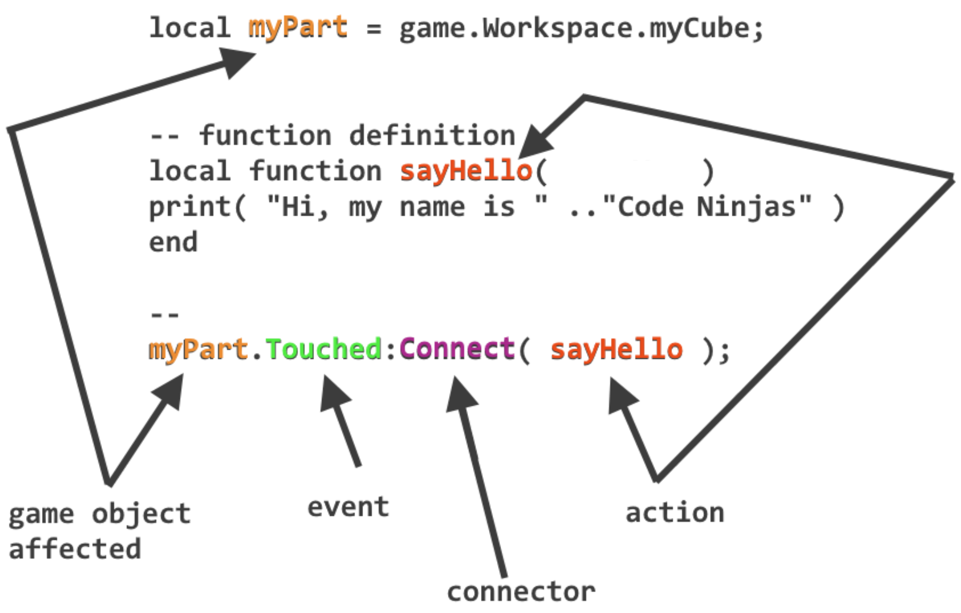
One of the most basic event is called **Touched**. When we touch a part, like a wedge, we want to execute a function definition (an action). How do we do this? In Roblox, we have to connect a function to an event.

How can we connect a function to an event? We use the function `Connect()` - notice the open and closed parenthesis after "`Connect`".

Task: Insert a cube into your video game. Anchor the part. Name this part “**`myCube`**” and attach a script to it. Next, write the code below. Once done, press F5 to play the game.



Archivable	✓
ClassName	Script
Name	Script
Parent	myCube
UniqueId	00b66ef041a13ca6092fbc2600000…



```
local myPart = game.Workspace.myCube;


-- function definition
local function sayHello(          )
print( "Hi, my name is " .."Code Ninjas" )
end

--
myPart.Touched:Connect( sayHello );
```

Chapter 16. Object Property

A property is an adjective, which is used to describe a game object.

In our example, we have a part that was renamed "**`myCube`**". Click on "**`myCube`**" and search for the "Properties" panel. You will see that these are the properties that describe myCube. **Refer back to Chapter 3 to help open the "Properties" panel.**

These properties of the object are storage elements. This means that we can read the content and also write data into them.

For example, our part called "**`myCube`**" can have a different color. We use our hierarchy and then reference the property name.

**** NOTE: it is important to recognize that the property name must be spelled exactly, including lower or uppercase. Else, the computer will be confused ****

16.1. BrickColor Property

Task: Insert a script into "**`myCube`**". Write the code below into your script for "**`myCube`**":

```
game.Workspace.myCube.BrickColor = BrickColor.new( 154 );
```

Notice that `BrickColor` is capital B and C. The code above creates a new `BrickColor` and gives it the value of 154. We then take this new color and assign it into our existing "`myCube`".

Go to the website https://create.roblox.com/docs/reference/engine/datatypes/BrickColor and you will see all of the possible numbers and the colors that are associated with the numbers. Notice that number 154 is dark red

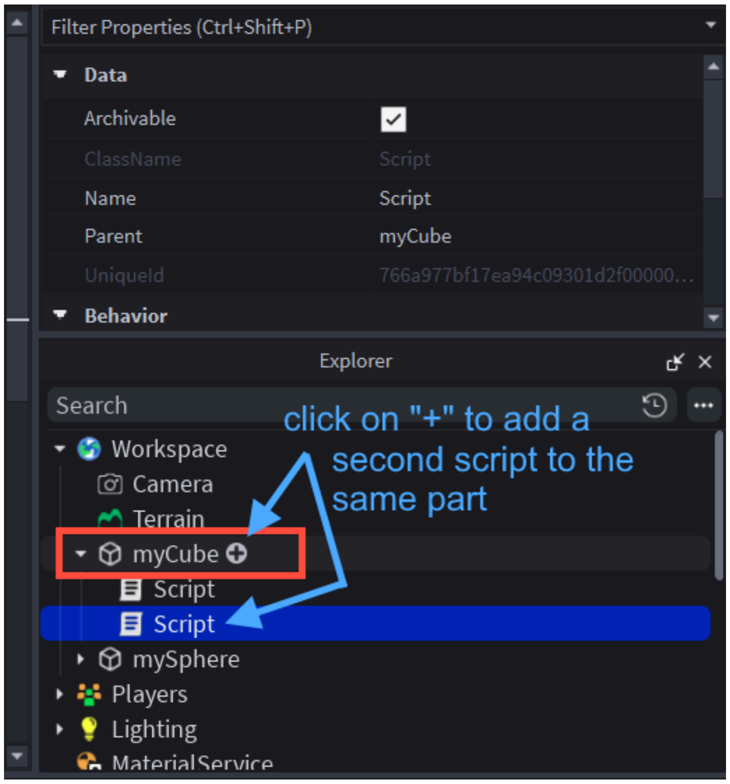
Task: Press F5 to play the game and you will see that the cube will change colors right away.

Chapter 17. Animation

We can create animation by having the `BrickColor` change values. In order to see the change in color, we need to create a delay.

In Roblox, a delay can be created using `wait(1)`. We can also create flashing animation by using a `while true` loop.

Task: Add a second script to the same part that is named "`myCube`"



The **second script** should have the following code

```
local myPart = game.Workspace.myCube;


-- function definition
local function change_color()
	while true do
		myPart.BrickColor = BrickColor.new( 145 )
		wait(1)
		myPart.BrickColor = BrickColor.new( 125 )
		wait(1)
	end
end

--
myPart.Touched:Connect( change_color );
```

Explanation

The part that is named "`myCube`" has two scripts.

The **first script** has a touch event and, upon your avatar touching the cube, will print out "Hi, my name is Code Ninjas".

The **second script** also has a touch event and, upon your avatar touching the cube, will run a while loop that will change the `BrickColor`.

Chapter 18. The "script.Parent" short cut, Super Speed and Super Jump

The Shortcut of `script.Parent`

In all of the code above, we have used the tree structure to refer to a game object. There is a short cut that can be used and the short cut is "`script.Parent`".

When we write "`game.Workspace.mySphere`", we are starting at the root node (the top) and going downwards. The shortcut of "`script.Parent`" is going in the opposite direction. This means that we are starting at the script level and going upwards. **A script is within the part** and this is composition where one entity is within another entity. The script is the child and the part is the parent.

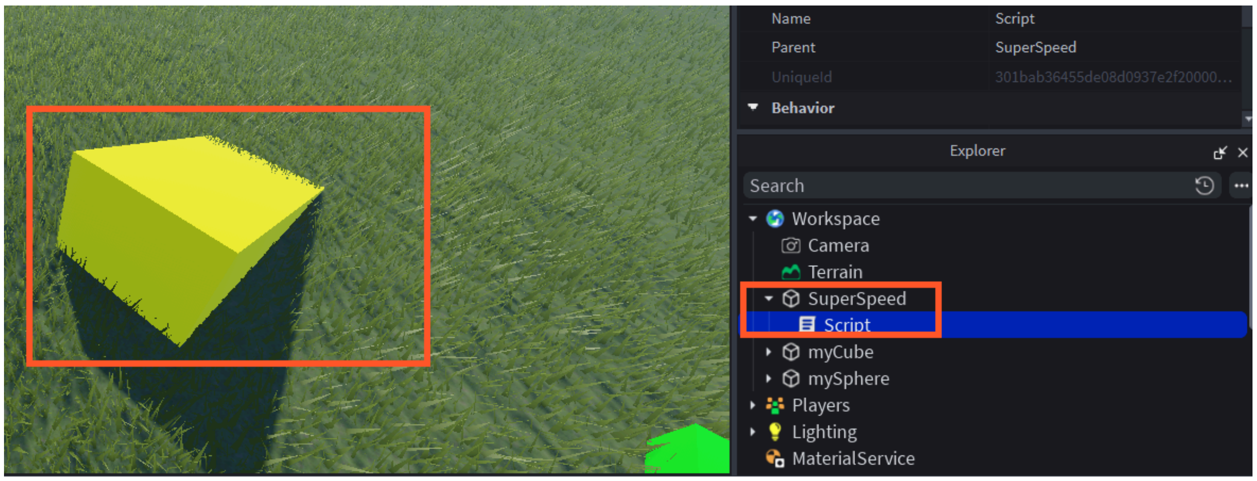
Assume that we have a script open and type in "`script.Parent`". This means that we are referring to the higher level, which is the parent. Because the script is within a part, this means that the part of `mySphere` is the parent.

So far, all properties belong to the part. **We can also change property values of an avatar**, such as the video game character that we are controlling when the video game is in Play mode.

WalkSpeed, JumpHeight, & JumpPower - Property Values

One property of the avatar is `WalkSpeed`, which is an attribute that describes how fast the avatar is traveling.

Task: Insert a wedge into your video game, make it yellow, name it "`SuperSpeed`", and insert a script into it. See below.



Name	Script
Parent	SuperSpeed
UniqueId	301bab36455de08d0937e2f20000...

Task: Double click the script of `SuperSpeed` and type in the following code.

```
print("Hello world!")

local myPart = script.Parent

-- function definition
local function onPartTouch(otherPart)
     local partParent = otherPart.Parent
     local humanoid = partParent:FindFirstChildWhichIsA("Humanoid")
     if ( humanoid ) then
          -- Set player's WalkSpeed to 250
          humanoid.WalkSpeed = 250
     end
end
-- event listener

myPart.Touched:Connect(onPartTouch)
```

Explanation

```
local function onPartTouch(otherPart)
```

a. the code above is the heading of function definition. The function name is "`onPartTouch`". The input to the function definition, which is named "`otherPart`", is placed there by Roblox. The "`otherPart`" identifies the component of the avatar (leg, arm, and etc) that touched the "SuperSpeed" part.

```
local humanoid = partParent:FindFirstChildWhichIsA("Humanoid")
```

a. remember that Roblox is based on a hierarchy. When a part touches "`SuperSpeed`", we want to find out whether or not the part is our Avatar game character. The code above uses the hierarchy to find the location of the part.

```
if ( humanoid ) then

end
```

a. Once we find the entity that touched the "`SuperSpeed`" part, we have to check if it is our Avatar game character. It is possible that another part, like a sphere, was rolled into the "`SuperSpeed`" part.

The "if" statement performs this check. If our Avatar did indeed touch "`SuperSpeed`", ONLY then do we manipulate the "Health" property of the Avatar.

Task: Challenge 1

Add another wedge part into your video game, make the color **orange**, and name it “`SuperJump`”.

Next, add a script into “`SuperJump`”. Next, write the code so that when your avatar touches the “`SuperJump`” part, load the number 500 into the attribute of “`JumpHeight”` and “`JumpPower”`.

Task: Challenge 2

Add another cube part into your video game, make the color red, and name it “**`HazardPack`**”.

Next, add a script into **`HazardPack`**. Next, write the code so that when your avatar touches the **`HazardPack`** part, load the number 0 into the attribute of `Health`.

Chapter 19. Checkerboard Obby

An obby is an obstacle course. There are many types of obbies but most require the main character to avoid a hazard by going around a hazard or jumping onto a safe platform.

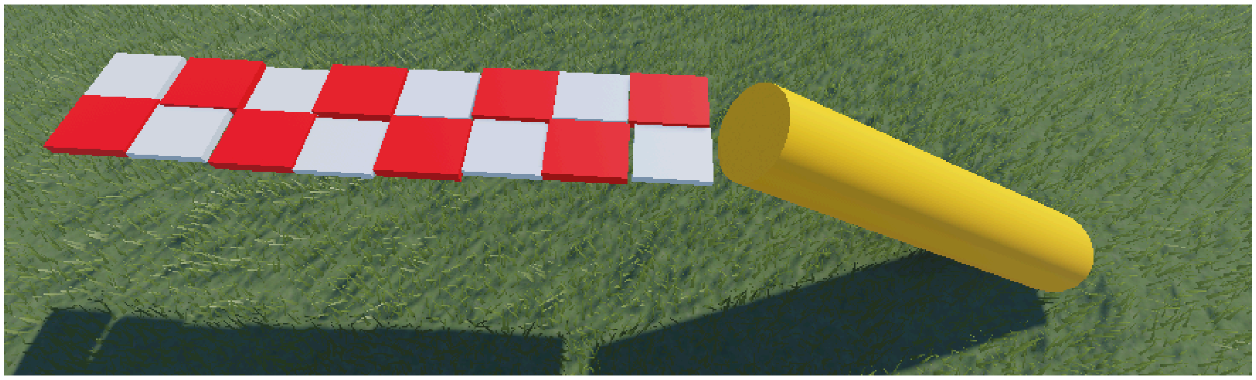
One of the most common types of obby is the checker board obby. See the image below for a checkerboard obby. **Your goal is to recreate the image that you see below.**

The red blocks are the "`HazardPack`", which was created as part of Challenge 2 from Chapter 18. If the avatar touches any of the red blocks, then the avatar will have its health go down to 0 and will respawn.

The white blocks are safe blocks and have no effect.

The orange cylinder is simply a ramp that is used to get up to the checkerboard obby.

Remember to "anchor" the checkerboard obby and the orange cylinder so that they float in the air.



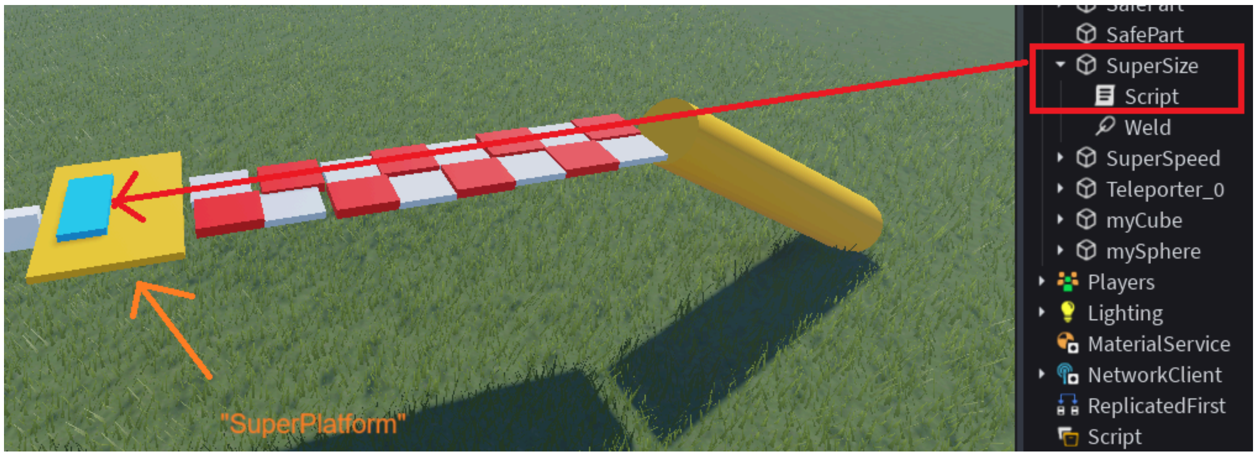
After completing the checkerboard obby, the game player should be rewarded with a super power. We will make the avatar grow in size. **The next super power will be Super Size.**

Chapter 20. Super Size Me

The avatar is not a single part but is composed of many parts that are combined together. We can modify each part individually.

Task: Start by adding a cube into the game and name it "`SuperPlatform`". Make the color as yellow and raise it into the air. Position it next to the checkerboard obby and remember to anchor it so that it stays in the air. **See on the next page.**

Task: Add a cube into your video game, make it a blue color, and name it “`SuperSize`”. Next, add a script into the part and **write the code that is found below.**



```
local part    = script.Parent
local count   = 0

part.Touched:Connect(function(hit)
      local Humanoid = hit.Parent:FindFirstChild("Humanoid")
      if Humanoid and count == 0 then

            part.Position = Vector3.new( 100, 200, 300 )

            local HS = Humanoid.HeadScale
            local BDS = Humanoid.BodyDepthScale
            local BWS = Humanoid.BodyWidthScale
            local BHS = Humanoid.BodyHeightScale

            HS.Value = HS.Value * 10
            BDS.Value = BDS.Value * 10
            BWS.Value = BWS.Value * 10
            BHS.Value = BHS.Value * 10

            count = count + 1
      end

      if Humanoid then
            Humanoid.WalkSpeed   = 100
            Humanoid.JumpPower   = 200
            Humanoid.JumpHeight  = 200
      end
end)
```

After writing the code, play the game and you will see that the avatar will grow in size. The "`* 10`" means to scale up each body party by an amount of 10. This means that each body part will increase in size by 10.

If the goal is to shrink the avatar, then change the "*** 10**" to "*** .10**"

Danger Bridge

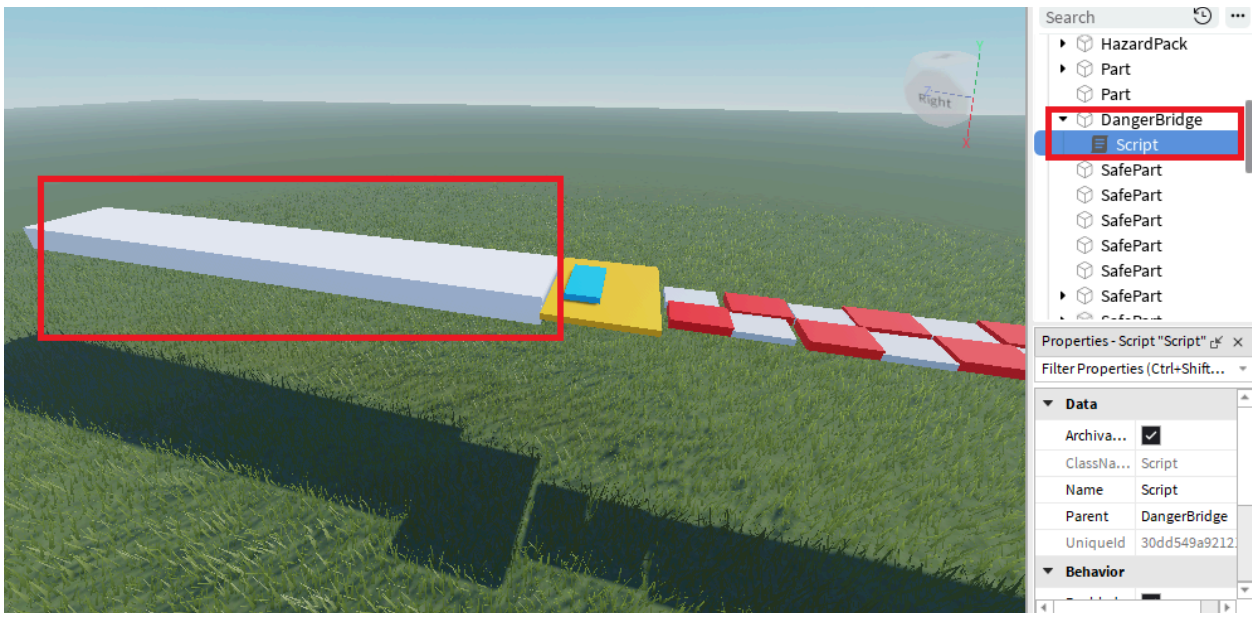
We will create a bridge that has the following attributes

1. Changes `BrickColor` every 1 second
2. When the `BrickColor` is either 24 or 50, the health of the avatar is set to 0

Task: Start off by inserting a cube into the game, make the cube long, make the initial color as white, and give the cube a name of "`DangerBridge`".

Task: First, place the "`DangerBridge`" floating into the air and next to the `SuperSize` block. Remember to anchor the "`DangerBridge`".

Task: Next, insert a script into the "`DangerBridge`".



Archiva...	✓
ClassNa...	Script
Name	Script
Parent	DangerBridge
UniqueId	30dd549a9212

Task: Write the following code into the script.

```
print("Hello world!")

local cnt = 0;
local myPart = script.Parent

while true do
	myPart.BrickColor = BrickColor.new(cnt)
	wait(1)
	cnt = cnt + 1

	if cnt > 55 then
		cnt = 0
	end

	print( "cnt: " .. cnt )

end
```

The code above uses a variable named `local cnt` to keep track of the current color of the `BrickColor` attribute.

Every 1 second, the `cnt` value is increased by 1. This means that the count becomes the color number.

There are situations where we will want to make a decision based on the `BrickColor` being of a specific number. Is it possible to detect the current `BrickColor` when the game is in play? There is a way to detect the current number of the `BrickColor` and we will use `BrickColor. Number` to help us.

Task: Challenge 1

We want some colors of the `DangerRamp` to be a hazard while others are safe. When the avatar is running on the `DangerRamp` and the color is 24, then the avatar's health will be set to 0.

Add another script to the same `DangerRamp` part. **In this second script**, do the following

1. Write a function definition that will set the avatar's health to 0 ONLY if the `BrickColor.Number` is 24. **Read the hint below.**

   Hint

1. When the avatar is running on the DangerRamp, this is considered a touch event. **The avatar must continue to be running for it to be seen as a touch event.** When the avatar stops running, this stationary state will not trigger a touch event even if the avatar is standing on the part
2. The "if" statement to check the `BrickColor.Number` comes first
3. When the `BrickColor.Number` is equal to 24, inside the "if" statement is the code that will set the avatar's health to 0
4. Then, enhance the "if" statement to be "`BrickColor.Number` is equal to 24" **or** "`BrickColor.Number` is equal to 50". In Lua scripting, the word "**`or`**" is used instead of the pipe "**||**"

Roblox Supplemental 1

Chapter 1. Introduction to Worm Hole

What is a worm hole? According to **https://en.wikipedia.org/wiki/Wormhole**, a wormhole is "a tunnel with two ends, each at separate points in spacetime. A wormhole can be used to travel long distances". A wormhole can also be thought of as a shortcut.

Why are worm holes useful? A wormhole allows us to travel a long distance in a short amount of time. What are some real world applications of worm holes? In Star Trek, a worm hole is used to travel from one point to another. Wormholes can be used to quickly transport items or human beings (ie. to give aid) and for instantaneous messaging.

In video games, wormholes are used as a portal to a new world. Each new world has its own challenges. Rather than having a player run from the player's current position to the destination, we will now have the player enter the wormhole and be teleported directly to a new world.

A wormhole can be used to get us to our homebase. Our homebase can hold our achievements, be a place where we can acquire our next task, or serve as a reference point.

Chapter 2.Teleportation by Manipulating CFRAME

In Roblox, we have the capability to create a wormhole and use it to teleport the Avatar to any location. This opens the door to many different ways to design a game.

In Roblox, the Avatar that we control is composed of many different parts (such as head, arms, and legs). We can use this to our advantage. We will focus on the upper torso. **The upper torso has a property called `CFrame`, which stands for coordinate frame. This `CFrame` is described by 3 numbers and the 3 numbers refer to the x, y, and z position.**

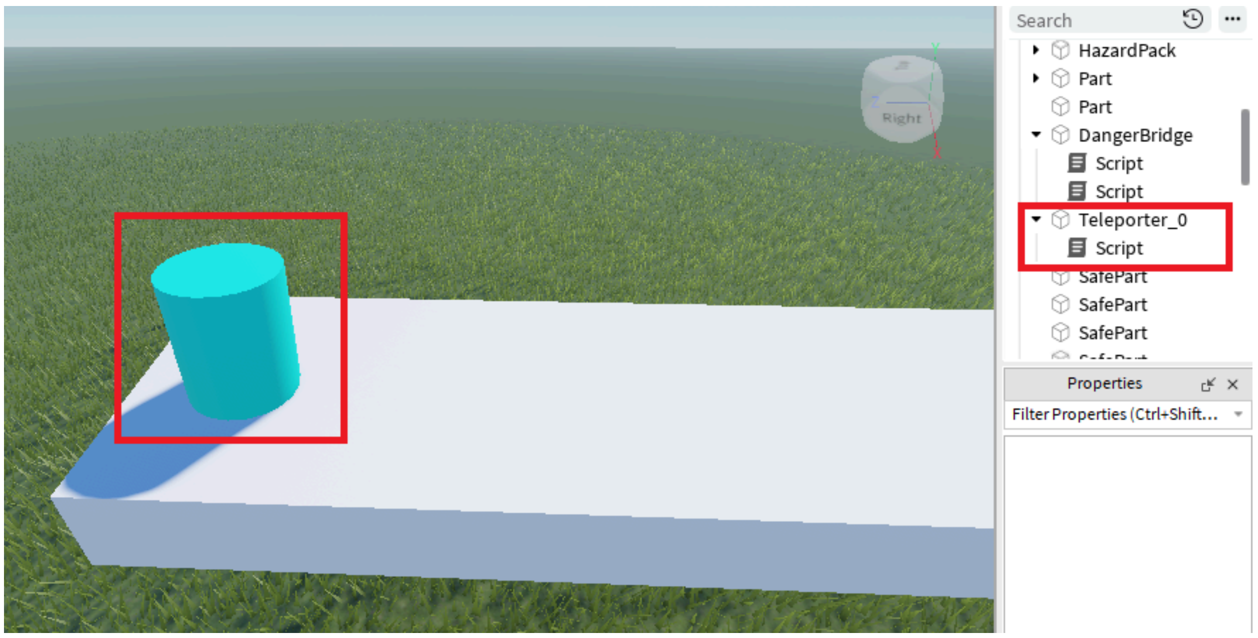
By storing a new value into the `CFrame` of the upper torso, we are instantly placed to a new location. Look at the code below. Notice that we created a new `CFrame` and this new `CFrame` has inputs `78, 2.449, and-138.28`. The numbers represent the x, y, and z coordinates. After we create and initialize the new CFrame, we must assign the new data to our existing upper torso.

The code to assign the upper torso to our new position is

```
hit.Parent.UpperTorso.CFrame = CFrame.new( 78, 2.449,-138.28 )
```

2.1. Create Teleporter

Task: Add a cylinder at the end of the `DangerBridge`, make it blue, and name it `Teleporter_0`. Add a script into the `Teleporter_0`.



The code below is shorthand notation for a touch event. **Task:** Copy this code into the script.

***** NOTE *** This shorthand notation can be tricky to read. Separating the code into function definition and event will make it easier to read, test, and maintain.**

```
script.Parent.Touched:connect(function(hit)
      if hit.Parent:FindFirstChild( "Humanoid" ) then
            hit.Parent.UpperTorso.CFrame = CFrame.new( 78,
      20.449,-138.28)
      end
end)
```

Task: Challenge 0

Create two more teleports that are named “`Teleporter_1`” and “`Teleporter_2`”. Change the values of the `CFrame` so that each teleporter will teleport you to a different area of the video game.

Remember to anchor both teleporters and make sure that there is a “touch” event.

Chapter 3. Animation Using Tween Service

What is animation? Animation is a flip book. Images are stacked on top of each other and there is a delay in between switching from one image to another.

We have already done animation by making our cube flash at different frequencies and at different colors. This was accomplished by adding a delay in between changing the `BrickColor` property value. This was basic animation but the animation was not a smooth transition from one state to another.

Roblox has a dedicated service that allows us to schedule animation and this service allows a smooth transition from one state to another. **This service is called `TweenService`.**

The first step is to get the service from Roblox and this is done using

```
local TweenService = game:GetService("TweenService")
```

The above line of code retrieves a `TweenService` and stores a reference to it into a local variable called `TweenService`.

The second step is to specify the animation properties by defining the `TweenInfo`

***** NOTE The `TweenInfo` defines the animation sequence, NOT how a part will be animated. This is an important difference. *****

The TweenInfo will define the duration of the animation and if the animation should run once or multiple times.

According to Roblox, the `TweenInfo` takes inputs as parameters. **DO NOT WRITE THE CODE BELOW. The code below is just for study.**

```
TweenInfo.new
(
      number time = 1.0,
      Enum easingStyle = Enum.EasingStyle.Quad,
      Enum easingDirection = Enum.EasingDirection.Out,
      number repeatCount = 0,
      bool reverses = false,
      number delayTime = 0
)
```

Explanation of Properties

number TweenInfo.Time - The duration of the tween animation. The unit is seconds.

Enum TweenInfo.EasingDirection- The direction in which the EasingStyle executes.

number TweenInfo.DelayTime- The amount of time that elapses before the tween animation starts again. The unit is seconds.

number TweenInfo.RepeatCount- The number of times the tween repeats after tweening once.

Enum TweenInfo.EasingStyle- The style in which the tween executes.

bool TweenInfo.Reverses- Whether or not the tween does the reverse once the initial tween is completed

The third step is to define how the part will be animated and then create the Tween. Remember that animation is based on a transition from a start state to an end state. The third step is to define the end state

1. For example, when the game begins a platform can have size 10 in the x position (the start state). The end state would be to have the platform change its size to 100 in the x position

2. By Tweening this, the platform would grow by 10 in a smooth animation sequence

```
Create ( Instance instance , TweenInfo tweenInfo , Dictionary
propertyTable )
```

The first parameter specifies the part that we want to animate.

The second part describes the animation sequence, such as duration.

The third parameter specifies the property values that will be changed to create the animation

Chapter 4. DangerBridge Modified

Task: Let's go back and modify an existing part. **Refer back to the `DangerBridge`.** Add another script into `DangerBridge`. **The `DangerBridge` should have 3 scripts.**

Task: **The 3rd script should have the following code.** When testing your Roblox video game, the code below will cause the `DangerBridge` to expand and compress so that the width of the `DangerBridge` will change. The Tween service allows us to set the end state and then the Tween service will animate the `DangerBridge` for us.

```
local myPart = script.Parent

local TweenService = game:GetService("TweenService")

local tweenInfo = TweenInfo.new (
      4,
      Enum.EasingStyle.Linear,
      Enum.EasingDirection.Out,
      -1,
      true,
      .5
)

-- property table. This is the end state
local goal = {}
goal.Size = Vector3.new( 20, myPart.Size.Y, myPart.Size.Z )

wait( 5 )

print( "about to create TweenService" )
local tweenCon = TweenService:Create( myPart, tweenInfo, goal )
print( "done with create TweenService" )

tweenCon:Play()
```

Explanation Of Highlighted Parts

The goal is the end state. The goal tells the Tween Service how the part should look like after the animation sequence is done.

We will wait 5 seconds.

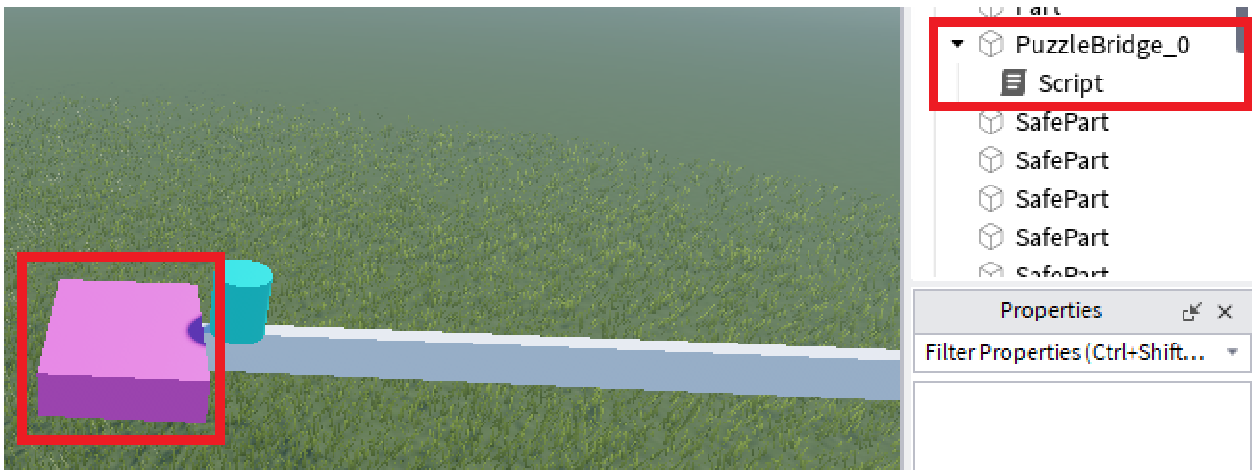
Finally, we tell the Tween service to activate the animation.

Chapter 5. Puzzle Bridge

We will want our video game to have platforms that are dynamically moving from left to right. The goal is to have the game avatar jump from one platform to the next in order to reach the safe area.

For this part of your video game, there will be 5 platforms that are floating in the air. Next, the `Tween Service` is used to have each platform move on its own.

Task: Add a cube into your video game, name it "**`PuzzleBridge_0`**", make it purple, anchor the part, and make sure that "**`PuzzleBridge_0`**" is floating in the air. Finally, add a script to it.



Task: Write the following code into the script. Notice that the code below will have the "**`PuzzleBridge_0`**" move smoothly on only one side.

```
local myPart = script.Parent

local TweenService = game:GetService("TweenService")

local tweenInfo = TweenInfo.new (
      4,
      Enum.EasingStyle.Linear,
      Enum.EasingDirection.Out,
      -1,
      true,
      .5
)

-- property table. This is the end state
local goal = {}
goal.Position = Vector3.new( myPart.Position.X + 20,
myPart.Position.Y, myPart.Position.Z )

wait( 5 )
print( "about to create TweenService" )
local tweenCon = TweenService:Create( myPart, tweenInfo, goal )
print( "done with create TweenService" )
tweenCon:Play()
```

Task: Challenge 0

Repeat the steps from above and name this second part "`PuzzleBridge_1`". **This second puzzle bridge will move in the opposite direction of the first puzzle bridge**. What should be changed to have the effect of moving in the opposite direction?

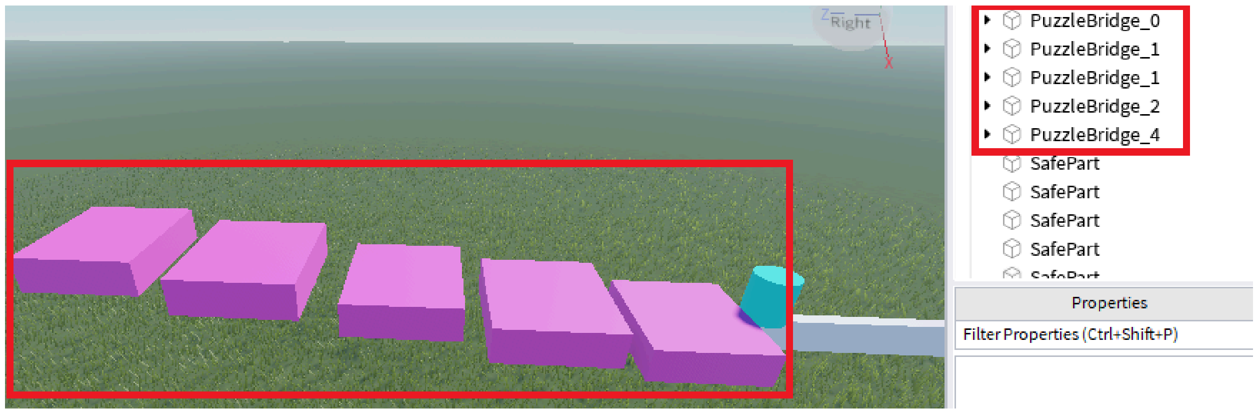
Hint: change the values of `goal.Position`

Task: Challenge 1

Repeat the steps from above but this time, there will be 3 more puzzle bridges ("`PuzzleBridge_2`", "`PuzzleBridge_3`", and "`PuzzleBridge_4`").

The end goal is the following

1. Move to the **left** and return back to the original position. The "`PuzzleBridge_0`", "`PuzzleBridge_2`" and "`PuzzleBridge_4`" will do this.
2. Move to the **right** and return back to the original position. The "`PuzzleBridge_1`", "`PuzzleBridge_3`" will do this.



Chapter 6 Dialog Boxes

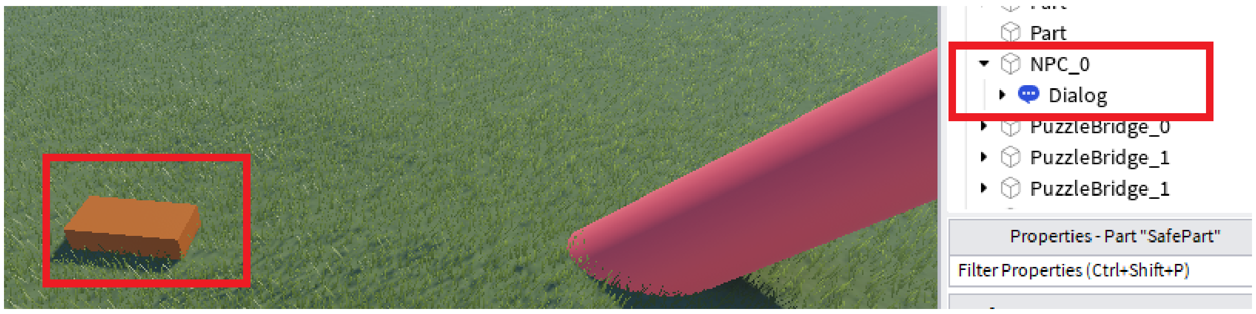
Conversation within a video game is important because a conversation is a story. The story is able to set up the challenge via an introduction and the introduction lets the game player(s) know how the challenge came to be. Next, the story introduces the problem and then directs the player to solve the problem by giving out a task..

A non-playable character (NPC) is an entity that is not under the control of the game player. An NPC will be used to give a task or trivia question to the game player.

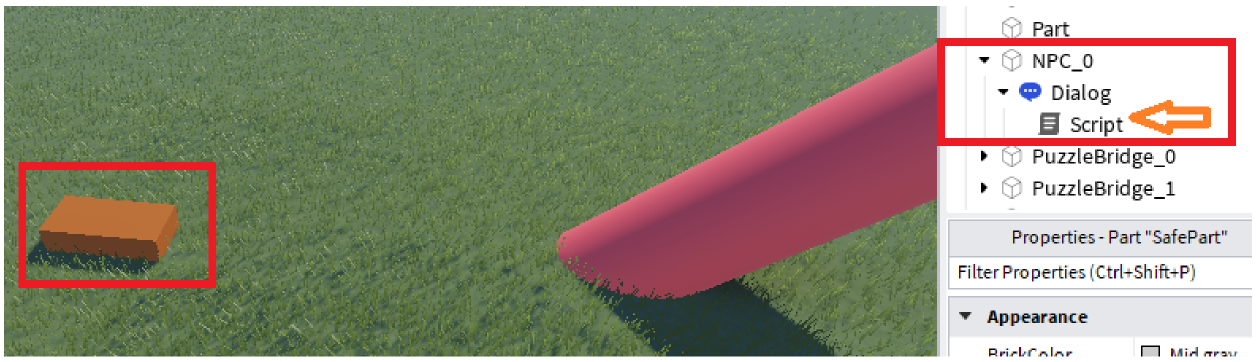
Within Roblox, we can create conversation by using Dialog Boxes. Dialog boxes can be used for the following

1. Give a task to the game player
2. Give a hint

Task: Add a cube at the end of the `PuzzleBridge` area. The color of the cube is brown and give it a name of “`NPC_0`”. Next, hover the mouse cursor over to the Explorer panel and put the mouse cursor on “`NPC_0`”. We are going to click on the “+” sign and search for “`Dialog`”. Click on “`Dialog`” and a dialog part appears under the “`NPC_0`”.


Part
NPC_0
Dialog
PuzzleBridge_0
PuzzleBridge_1
PuzzleBridge_1
Properties - Part "SafePart"
Filter Properties (Ctrl+Shift+P)

Task: Then, hover the mouse cursor over to the “`Dialog`” and click on “+”. Finally, add a script into the “`Dialog`”.


Part
NPC_0
Dialog
Script
PuzzleBridge_0
PuzzleBridge_1
Properties - Part "SafePart"
Filter Properties (Ctrl+Shift+P)
Appearance

Task: Then, write the following code into the script of the "`Dialog`". The code below is the initial prompt of the NPC. When playing the video game, there will be a question mark that appears on top of the cube. When clicking on the question mark, the `dialog1.InitialPrompt = "Hello World"` is what will appear.

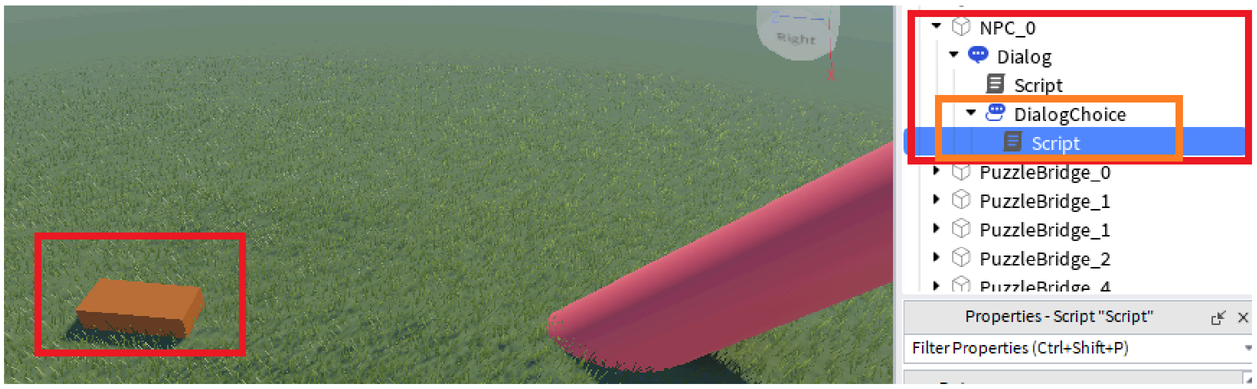
```
print("Hello world!")

local dialog1 = script.Parent
dialog1.InitialPrompt = "Hello World"
dialog1.Tone = 1
dialog1.ConversationDistance = 10
```

When we play the game, the "`Hello World`" message is displayed. We will want the game player to have selectable options. **We will use "`DialogChoice`" to implement selectable options, which is the next section.**

6.1. Selectable Dialog Choices

Task: Hover the mouse cursor over to “Dialog” of NPC_0. Next, click on the “+” and type in “DialogChoice” and select “DialogChoice”.

Task: Next, hover the mouse cursor over to the “DialogChoice” and click on the “+”. Type in script and select script. The structure should look like below.



A “DialogChoice” is one option that the game player can select. Double click on the script of the first “DialogChoice”. Type the code below into the script.

```
local thisResponse              = script.Parent
thisResponse.ResponseDialog = "response 1"
thisResponse.UserDialog     = "tell me story 1"
thisResponse.Name           = "c1"
```

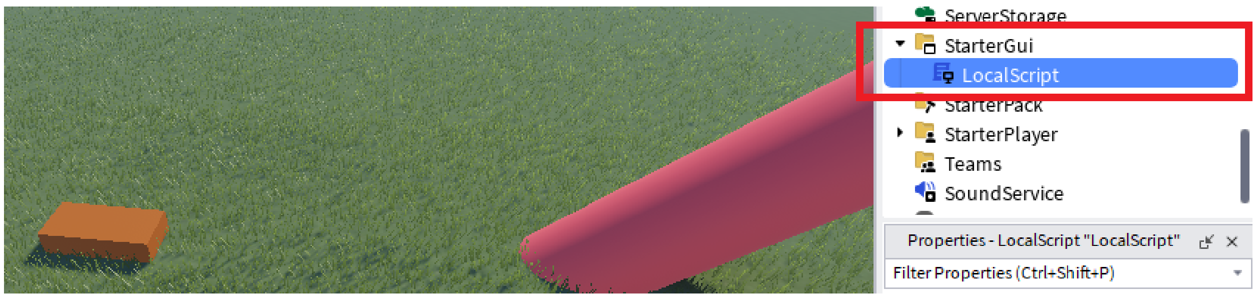
The is one `thisResponse.UserDialog     = "tell me story 1"` option that is presented to the game player. When the game player clicks on it, the response will be `thisResponse.ResponseDialog = "response 1"`.

The `thisResponse.Name           = "c1"` **is very important** because it gives a name to the DialogChoice. When a game player clicks on one of many DialogChoice, how do we know which DialogChoice was clicked on? We have to check the `thisResponse.Name`.

6.2. Decode The `DialogChoice` That Was Clicked On

Any time we have a mouse or keyboard event, we need to use a `LocalScript`. A `LocalScript` can only be in a specific folder. For us, we will put a `LocalScript` into the "`StarterGui`" folder.

Task: Hover the mouse cursor over to the "`StarterGui`" folder and click on the "+". Next, type in "`LocalScript`' and click on it.



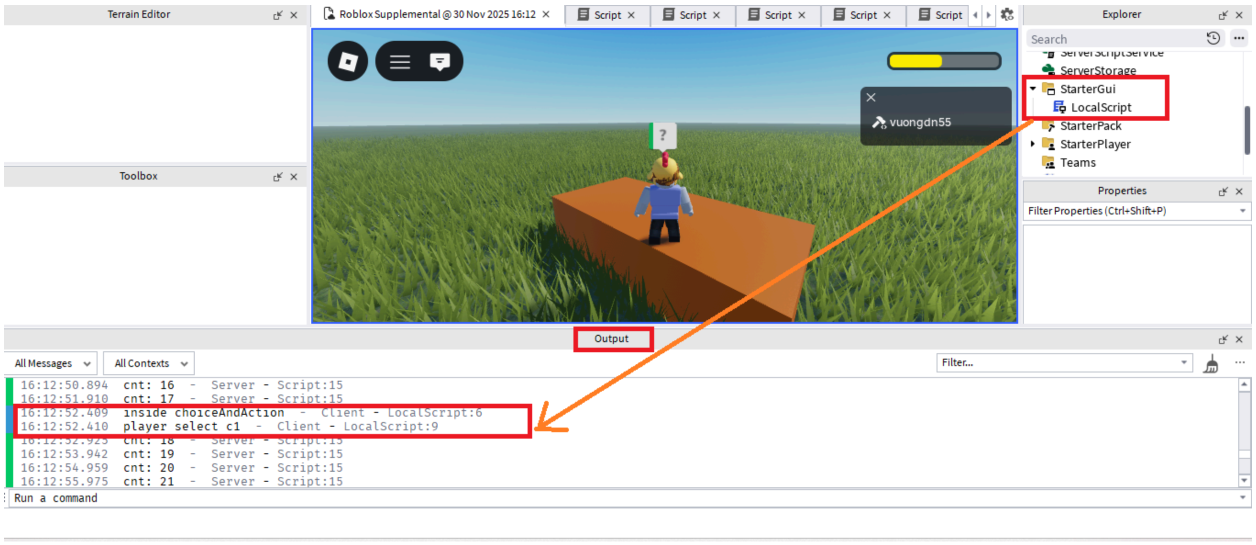
The code of a LocalScript is used to detect keyboard and mouse button clicks. **Task:** Type the following lines of code into the LocalScript.

```
local dialogObj = game.Workspace:WaitForChild("NPC_0")


-- function
dialogObj.Dialog.DialogChoiceSelected:connect( function(player,
choice )
      print( "inside choiceAndAction" )
      if( choice.Name == "c1" ) then
            player.Character.Humanoid.Health = 0
            print( "player select c1" )

      elseif ( choice.Name == "c2" ) then
            player.Character.Humanoid.Health = 10

            print( "player select c2" )
      else
            print( "player did not select a c1 or c2" )

      end
end)
```

The `DialogChoice` that the game player clicked on is referenced by "**`choice`**". The NPC could give us many choices and we must check which choice was clicked on. The "`if-elseif-else`" structure will check which `DialogChoice` was clicked on by comparing the **`.Name`** property.

Also note that we are traversing the tree structure to reference a game object by its name. The code **`game.Workspace:WaitForChild("NPC_0")`** does this for us.

Play the game, approach the NPC_0, select an option, and the output console will print out the message that we put in. See the image below.



```
16:12:50.894  cnt: 16  -  Server - Script:15
16:12:51.910  cnt: 17  -  Server - Script:15
16:12:52.409  inside choiceAndAction  -  Client - LocalScript:6
16:12:52.410  player select c1  -  Client - LocalScript:9
16:12:52.925  cnt: 18  -  Server - Script:15
16:12:53.942  cnt: 19  -  Server - Script:15
16:12:54.959  cnt: 20  -  Server - Script:15
16:12:55.975  cnt: 21  -  Server - Script:15
Run a command
```

Task: Challenge 0. Modify the code above so that that player's Health **is not** 0 when choice "c1" is clicked on.

Task: Challenge 1. Add code into the "`if( choice.Name == "c1" ) then`" so that the avatar gets super speed. Put this code underneath the Health code.

HINT: remember that **`WalkSpeed`** is the property that determines the speed of the avatar.

Task: Challenge 2.

Add a 2nd `DialogChoice`. The name of this 2nd `DialogChoice` is "**c2**".

Modify the `LocalScript` so that if "c2" is chosen, the avatar gets super jump.

HINT: remember that **`.JumpPower`** and **`.JumpHeight`** control the force of a jump.

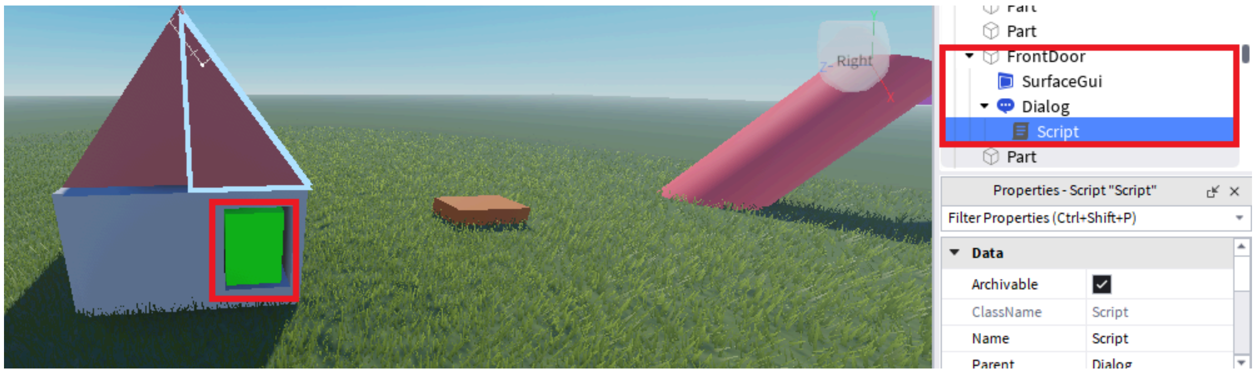
Chapter 7 Combine Dialog And Tween Service

The Tween service can be used with `Dialog` to create a video game that changes its game state based on the choice that we click on.

Task: 7.1. Create A House With A Front Door

The house should be next to the brown dialog block from chapter 6. The part of the front door should be named “`FrontDoor`”.

Task: 7.2. The Front Door Is The Dialog

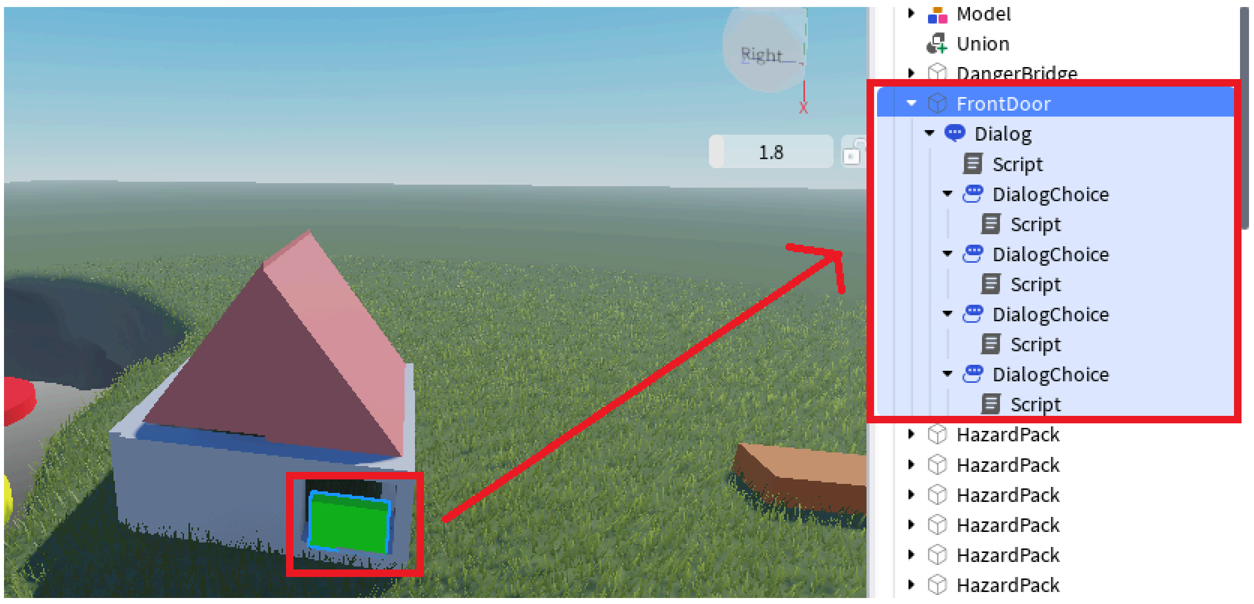

Right
Part
Part
FrontDoor
SurfaceGui
Dialog
Script
Part
Properties - Script "Script"
Filter Properties (Ctrl+Shift+P)
Data

Archivable	✓
ClassName	Script
Name	Script
Parent	Dialog

Task: 7.2.0. The Front Door should have a `Dialog`. Attach a script to the `Dialog` and the code below should be written for the script.

```
print("Hello world!")
local dialog1 = script.Parent
dialog1.InitialPrompt = "What is 5x5?"
dialog1.Tone = 1
dialog1.ConversationDistance = 10
```

Task: 7.2.1. The Front Door requires answering a question to open the door. The `Front Door` should present 4 `DialogChoices`. Try on your own to set up the 4 dialog choices. **The structure should look like below.**



7.3. A LocalScript will catch the choice that the game player clicked on. Task: Attach another LocalScript into StarterGui.

7.3.1. When the correct choice is clicked on, use the Tween Service to slide the door to the right side. **Task: The content of the 2nd LocalScript is below.**

For this example, the third dialog choice is the correct answer of 25. **Once the game player clicks on the answer of 25, the LocalScript will execute the "c3" branch.**

```
local dialogObj = game.Workspace:WaitForChild("FrontDoor")

-- function
dialogObj.Dialog.DialogChoiceSelected:connect( function(player, choice )

      print( "inside choiceAndAction" )
      if( choice.Name == "c1" ) then
            print( "player select c1" )

      elseif ( choice.Name == "c2" ) then

            print( "player select c2" )
      elseif ( choice.Name == "c3" ) then


            local myPart = dialogObj

            local TweenService = game:GetService("TweenService")

            local tweenInfo = TweenInfo.new (
                  4,
                  Enum.EasingStyle.Linear,
                  Enum.EasingDirection.Out,
                  -1,
                  true,
                  .5
            )

            -- property table. This is the end state
            local goal = {}
            goal.Position = Vector3.new( -42, 15.499, 240.781 )

            wait( .5 )
            print( "about to create TweenService" )

            local tweenCon = TweenService:Create( myPart, tweenInfo, goal )

            print( "done with create TweenService" )
            tweenCon:Play()
            wait(5)
            tweenCon:Cancel()

            print( "player select c3" )
      else
            print( "player did not select a c1, c2, or c3" )

      end
end)
```

Chapter 8. Multiple Animation Sequence Using Tween Service

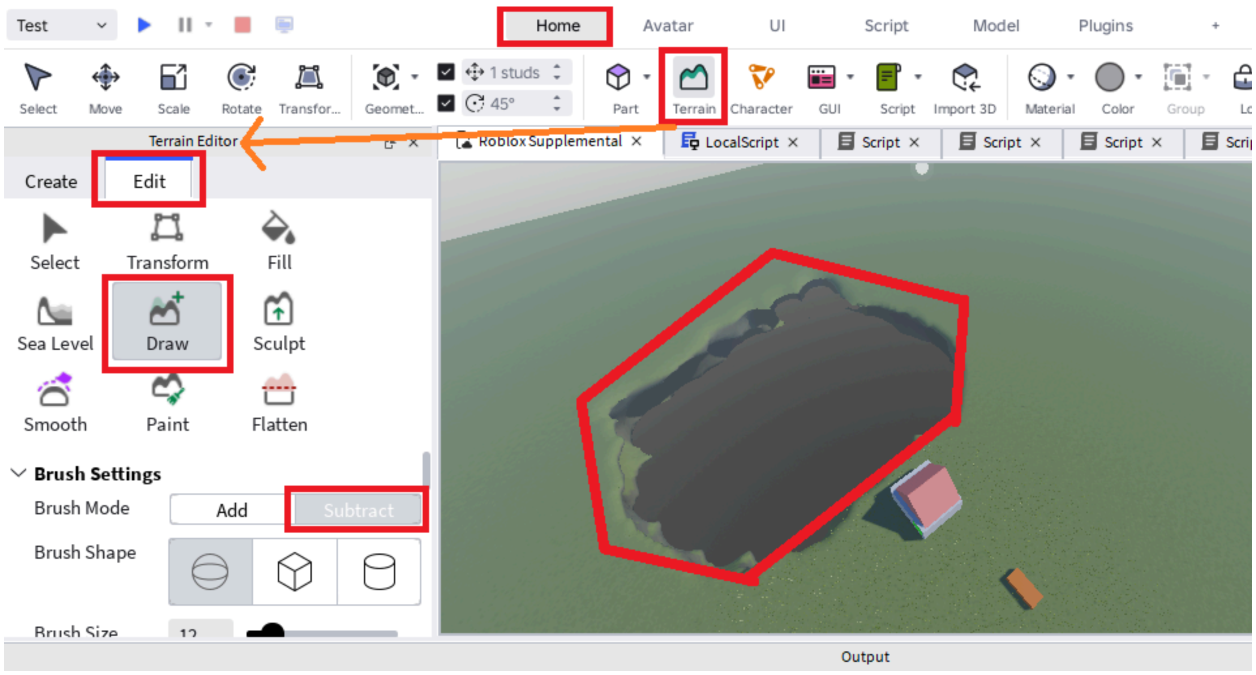
So far, all of the animations that we have used with Tween Service only run once. This means that the animation only has one state.

What if the goal was to create an animation sequence that has multiple states and we cycle through each state? **This is possible!** We can start an animation with `:Play()`, wait for a little bit with `wait(5)`, and then stop the animation with `:Cancel();`

For this chapter, the goal is to create an animation sequence that has multiple states.

Task: 8.1. Make A Hole Next To The House

Use the Editor and make a large hole next to the house. Recreate the image that you see below.

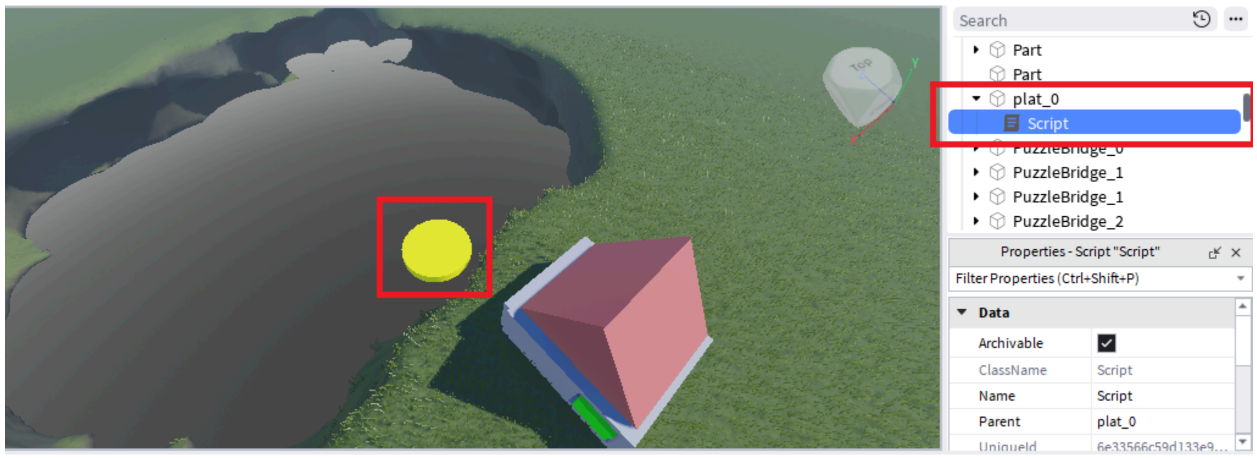


8.2 Circular Movement

Next, we will need to create a platform that moves in a circular pattern. This requires the platform to have multiple states. For this project, each state is a position that the platform can be in.

Task: Insert a cylinder into the video game, name it `plat_0`, make it yellow, position it in an area where the avatar can jump onto it, and make it anchored.

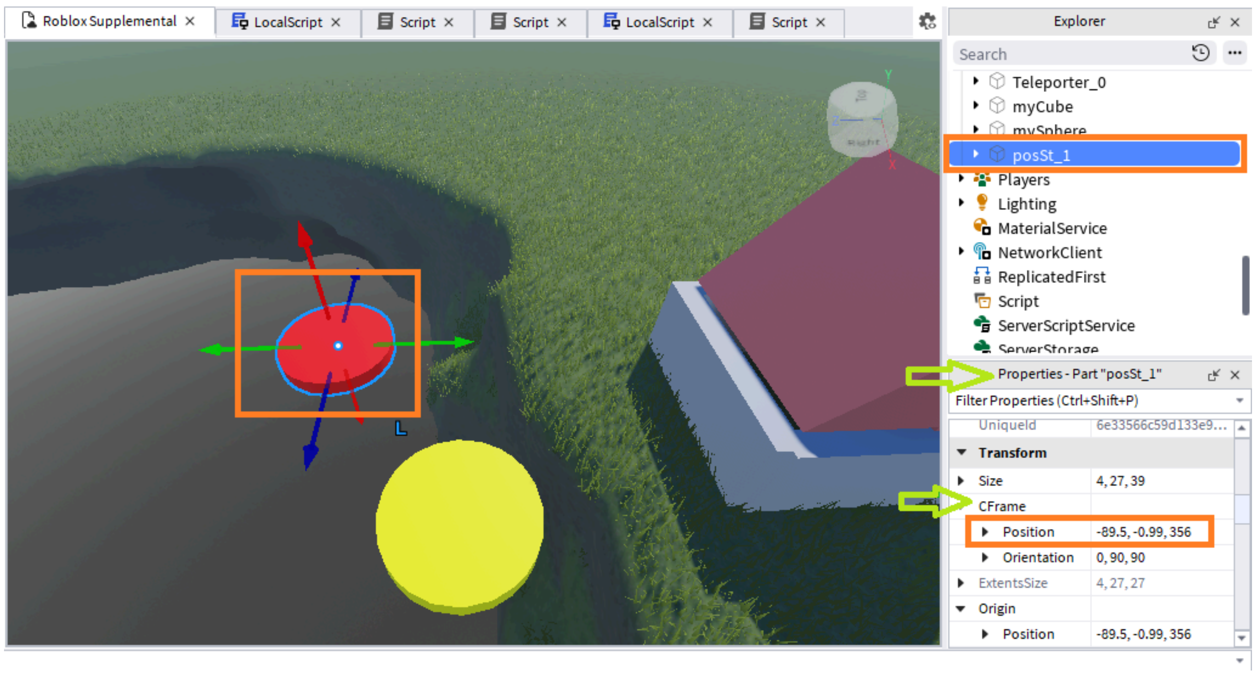
Task: Next, add a script to it. No code will be written into this script right not but code will be written later on.



Archivable	✓
ClassName	Script
Name	Script
Parent	plat_0
UniqueId	6e33566c59d133e9...

By default, the current position (remember that it is a Vector3 of x, y, and z) of the `plat_0` is the starting state. The platform must have 5 position states. To figure out the position states, do the following.

Task: Insert another cylinder into the video game. Name it "posSt_1", make it red, and position it right and then forward with respect of the yellow, "plat_0". See below.



UniqueId	6e33566c59d133e9...
Transform	
Size	4, 27, 39
CFrame	
Position	-89.5, -0.99, 356
Orientation	0, 90, 90
ExtentsSize	4, 27, 27
Origin	
Position	-89.5, -0.99, 356

Task: From the "Properties" panel, scroll down until you see "CFrame" and then "Position", which is the **x, y, and z** value of its Vector3. **Remember these values because we will be using them later on.**

Task: Write down these position values because it will be the 2nd position state. Everyone's numbers will be different. My numbers are just an example.

Task: Click on the yellow platform that is named "`plat_0`". Next, click on its script. Finally, write the following code into the script. The orange text below is the numbers of the 2nd position state. This means that "`plat_0`" is going to animate itself to the 2nd position state and this is why we need to put it as our **goal**.

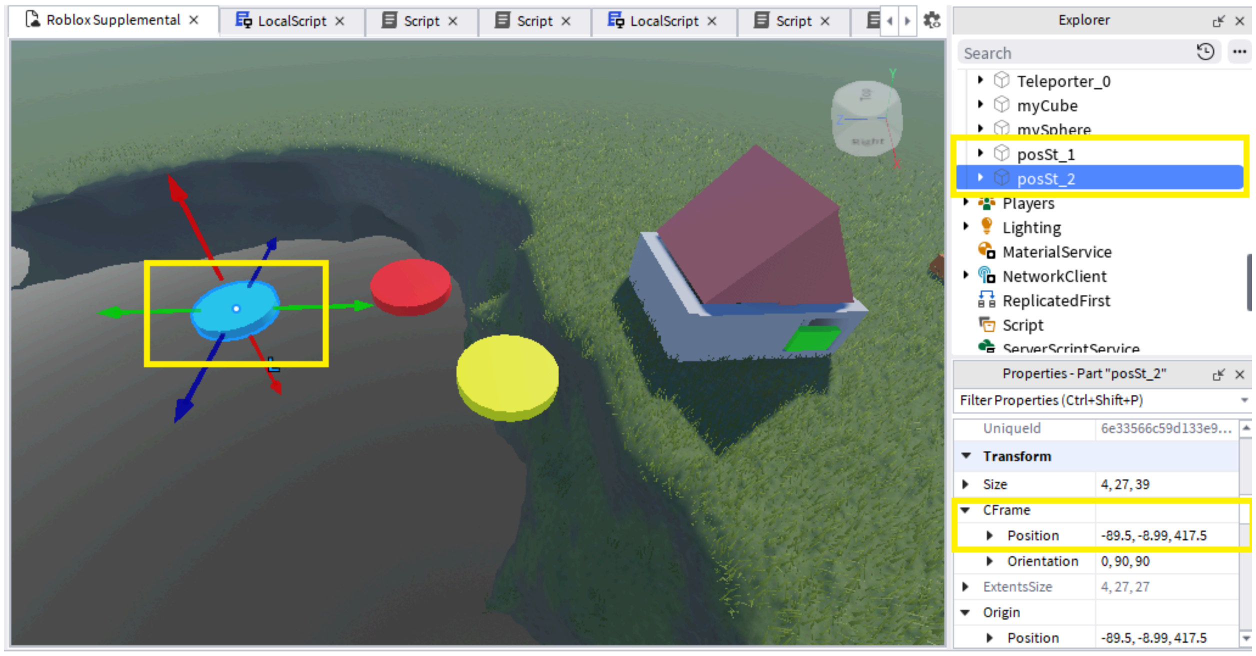
After typing out the code, play the video game and the yellow, "`plat_0`", will begin to move itself into the position of the red platform.

```
local myPart = script.Parent

local TweenService = game:GetService("TweenService")

local tweenInfo = TweenInfo.new (
      4,
      Enum.EasingStyle.Linear,
      Enum.EasingDirection.Out,
      -1,
      true,
      .5
)

-- property table. This is the end state
local goal = {}
goal.Position = Vector3.new( -89.5, -0.99, 356 )

wait( 5 )
print( "about to create TweenService" )
local tweenCon = TweenService:Create( myPart, tweenInfo, goal )
print( "done with create TweenService" )
tweenCon:Play()


wait( 5 )
```

8.3 More Position States

The above code is a nice start because the red, "`plat_0`" moves into the 2nd position state. Now, we are going to add more position states.

Task: Add another cylinder into the video game, name it "`posSt_2`", and make it blue. Position the blue, "`posSt_2`" forward from the red, "`posSt_1`". **Next, write down the Vector3 values of its position**. Remember that these values will be used later on. See below.



Properties - Part "posSt_2"

UniqueId	6e33566c59d133e9...
Transform	
Size	4, 27, 39
CFrame	
Position	-89.5, -8.99, 417.5
Orientation	0, 90, 90
ExtentsSize	4, 27, 27
Origin	
Position	-89.5, -8.99, 417.5

Task: Click on the yellow platform that is named "`plat_0`" and then click on the script. **The script already exists and has code from the work that we previously completed. We are going to add code onto what you already have.**

Next, we will add the following code onto the existing code. **Task: Only add the green part onto the existing code. The numbers in yellow match the numbers of the 2nd platform that was previously written down.**

```
local myPart = script.Parent

local TweenService = game:GetService("TweenService")

local tweenInfo = TweenInfo.new (
      4,
      Enum.EasingStyle.Linear,
      Enum.EasingDirection.Out,
      -1,
      true,
      .5
)

-- property table. This is the end state
local goal = {}
goal.Position = Vector3.new( -89.5, -0.99, 356 )

wait( 5 )
print( "about to create TweenService" )
local tweenCon = TweenService:Create( myPart, tweenInfo, goal )
print( "done with create TweenService" )
tweenCon:Play()


wait( 5 )


-- position state
tweenCon:Cancel() —- this cancels the previous animation

-- property table. This is the end state
local goal = {}
goal.Position = Vector3.new( -89.5, -8.99, 417.5 )

print( "about to create TweenService" )
local tweenCon = TweenService:Create( myPart, tweenInfo, goal )
print( "done with create TweenService" )
tweenCon:Play()
```

Task: Challenge

The code from above has the yellow platform that is named "`plat_0`" moving from the initial position state to another position state. For this challenge, add onto the script of "plat_0" so that it will move into 5 position states.

Task: Super Challenge

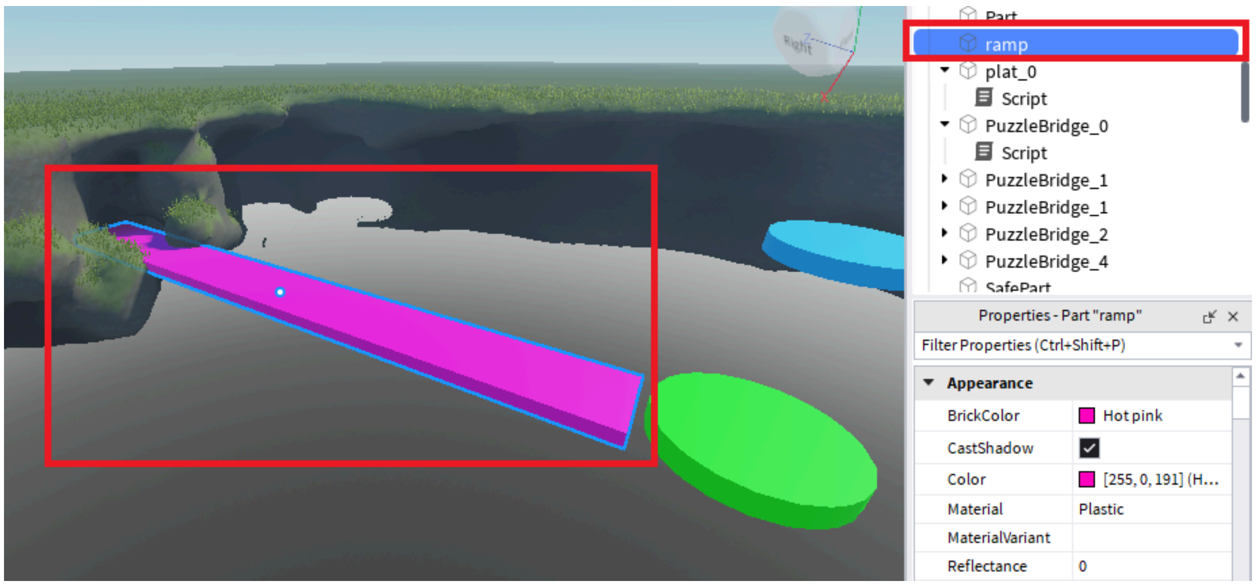
The challenge from above will see the "`plat_0`" move into 5 position states but then "`plat_0`" will stop.

Modify the script of "`plat_0`" so that the "`plat_0`" will continue to cycle through all 5 position states in sequence forever. **HINT: cancel each previous animation to start the next animation sequence. What can be used to run code forever?**

Finish Ramp

Task: Add a cube into the video game. Make it purple and name it "`ramp`". Finally, position the ramp next to the green platform and anchor the ramp.

This ramp will allow the game player to hop off the rotating platform and make it to the other side. See below.



BrickColor	Hot pink
CastShadow	✓
Color	[255, 0, 191] (H...
Material	Plastic
MaterialVariant	
Reflectance	0

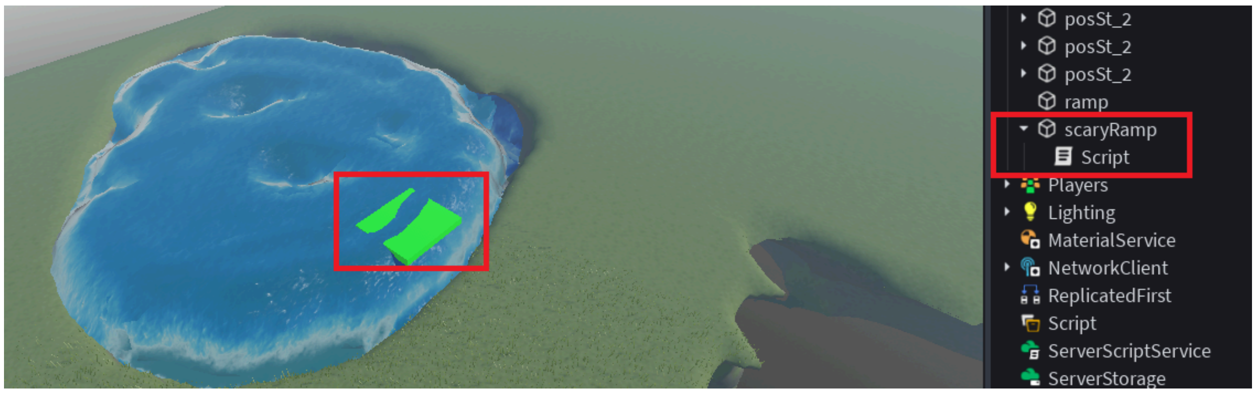
Chapter 9. One Final Project

This is a final project before completing Blue belt. This is a continuation of the previous work.

9.1. Creating the scary pond (can be done with Editor instead of code)

This requires creating a large body of water using the Terrain Editor. The scary pond must be made to look like the image below.

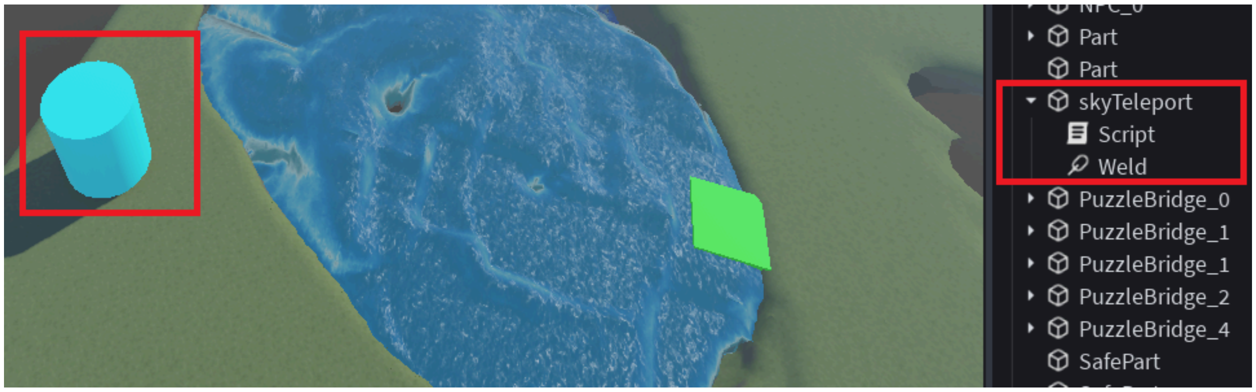
Task: **9.2. Next, add a cube into the game.** Then, make it color grey, name the cube "`scaryRamp`" and add a script into the "`scaryRamp`". Place the "`scaryRamp`" into the water. See below..



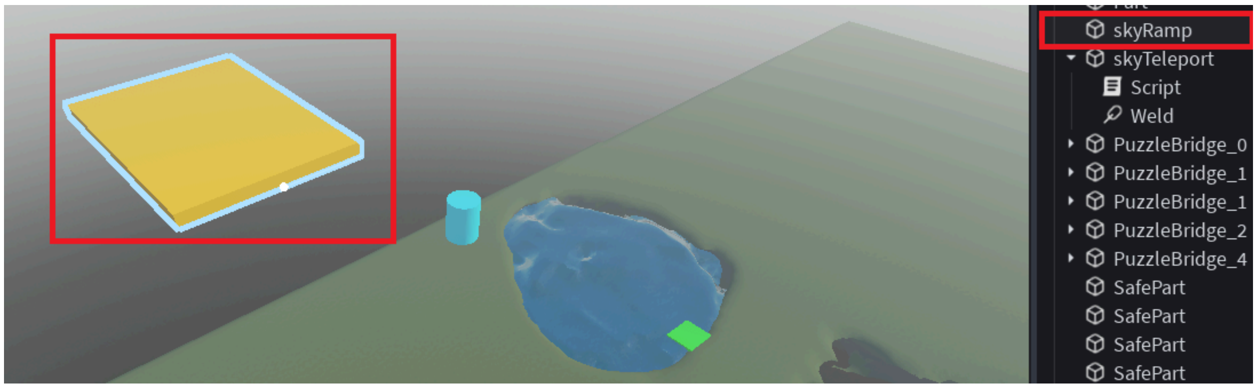
Requirements

1. The "`scaryRamp`" will use `TweenService` to move from one side to the other **only when a "touch" event is detected between the avatar and ramp.**
2. Once the ramp starts moving, it will move back and forth forever. Don't need a while loop to do this

Task: **9.3. Add a cylinder into the video game.** Next, name it "`skyTeleport`", add a script to it, make it blue, and place it on the other side of the scary pond.



Task: **9.4. Add a cube into the video game.** Next, name it "`skyRamp`", add a script to it, make it orange, make it float in the air, and anchor it.



Task: Remember the "`skyTeleport`" from step 9.3? When the avatar touches it, the "`skyTeleport`" will teleport the avatar up to the "`skyRamp`". Write the code that will teleport the avatar to the "`skyRamp`" when the avatar touches the "`skyTeleport`".

Once your avatar is at the top, then you have completed the Roblox supplemental curriculum.
Congratulations!!! You will now be moving onto the Purple belt curriculum!