Roblox Supplemental 3

Goals
1. Dialog is used to direct the player towards challenges
2. Use dialog to create a single dialog bubble
3. We can also determine when the dialog should start, by clicking or by distance
4. Camera manipulation using tween service
5. Control of camera using script

Chapter 1. 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 challenge.

Within Roblox, we can create conversation by using Dialog Boxes.

1.1. Setup Dialog System On Cube Part

1.1.a. Input a cube part into your game. The cube part should have the name "**Part**" and it should be inside "**Workspace**".
1.1.b. Click on the part and press F2 to rename it "**Part**"

1.1.c. Hover over the part that you included in step 1.1.b and click on the plus icon. Next, type in "**Dialog**" and select it.
1.1.d. Hover over the "**Dialog**" and click on the plus icon. Next, type in "**Script**".
1.1.e. Type the code below into your "**Script**" from step 2.1. exactly

```
print("Hello world!")
local dialog1 = script.Parent

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

1.2. Clickable Response Dialog Box

Add First Clickable Response Dialog Box

1.2.a. Hover over the dialog that you created in step 1.1.b. and click on the plus icon. Next, select "**DialogChoice**". This is the available choice that can be selected.

1.2.b. Hover over this "**DialogChoice**" dialog box and click on the plus icon. Next, select "**Script**" and type the code below exactly.

```
print("Hello world!")

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

Explanation

When activated, the dialog box displays `thisResponse.UserDialog = "tell me story 1"` to the user.

If the user does click on this response choice, then the dialog box displays `thisResponse.ResponseDialog = "response 1"`

An important property is `thisResponse.Name.` This is the identifier that will be used to help us react to the choice that was made. The above code names the first response dialog box as "**c1**"

Add Second Clickable Response Dialog Box

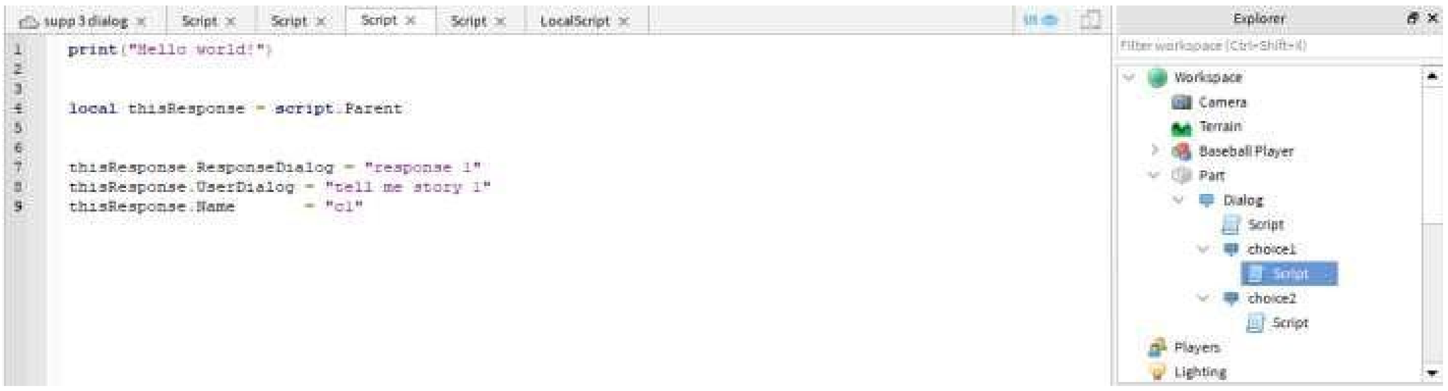
Most games have more than one possible response to choose from. So, let's add another dialog choice block into our game.

Hover over the dialog that you created in step 2 and click on the plus icon. Next, select "**DialogChoice**". This is the available choice that can be selected.

Hover over this "**DialogChoice**" dialog box click on the plus icon. Next, select "**Script**" and type the code below exactly.

```
print("Hello world!")

local thisResponse            = script.Parent
thisResponse.ResponseDialog = "response 2"
thisResponse.UserDialog     = "tell me story 2"
thisResponse.Name           = "c2"
```



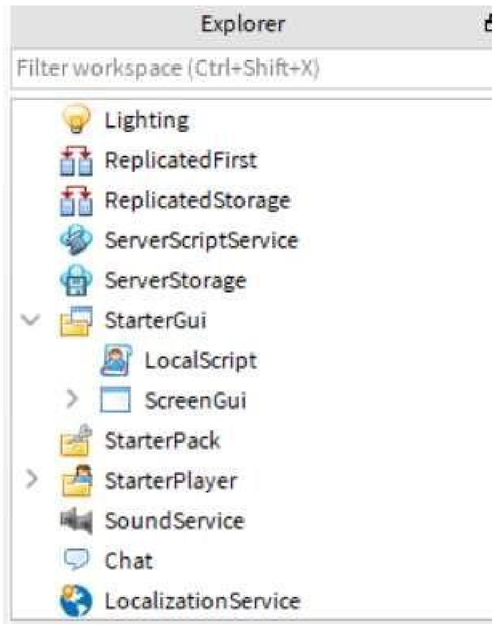
```
print("Hello world!")


local thisResponse = script.Parent


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

1.3. Catch The DialogChoice Made By The User

Chapter 1.1. created a simple dialog system. An important part of game play is to catch the choice made by the user and then react accordingly. In order to do this, we need a "**LocalScript**" that is stored into a special folder. The "**LocalScript**" contains the code that performs the event detection and handling and the special folder is "**StarterGui**".



The following code should be put inside the folder "**StarterGui**". Hover over the "**StarterGui**" folder and click on the plus icon. Next, type in "**LocalScript**". The code below goes inside this "**LocalScript**". Type the code exactly.

NOTE: The code below is shorthand notation for the event "connect" with a nameless function. It is advised to split them into two different sections. However, the style below is used to show that there are multiple ways to solve a problem.

```
print("Hello world!")

local dialogObj =
game.Workspace.Part.Dialog

-- function
workspace.Part.Dialog.DialogChoiceSelected:connect( function(player, choice )
     print( "inside choiceAndAction" )

     if( choice.Name == "c1" ) then
     player.Character.Humanoid.Health = 50
        print( "player select c1" )


     elseif ( choice.Name == "c2" ) then
          player.Character.Humanoid.Health = 10
          player.Character.Humanoid.HeadScale = 150
          print( "player select c2" )

     else
       print( "player did not select a c1 or c2" )
       player.Character.Humanoid.HeadScale = 150

     end

end)
```

Explanation

```
workspace.Part.Dialog.DialogChoiceSelected:connect( function(player, choice )
```

This is the event listener and handler combined together. Notice that the ***DialogChoiceSelected*** *is the event and the* ***:connect*** *links the event to the nameless function (which is the event handler).*

The ***player*** *is the game object that activated the dialog and* ***choice*** *is the response dialog that the player clicked on. These two things contain all of the information we need to create a game that reacts or adapts to the choices made by the player. This type of game is called a role player game (RPG).*

```
if( choice.Name == "c1" ) then
player.Character.Humanoid.Health = 50
print( "player select c1" )


elseif ( choice.Name == "c2" ) then
player.Character.Humanoid.Health = 10
player.Character.Humanoid.HeadScale = 150
print( "player select c2" )

else
print( "player did not select a c1 or c2" )
  player.Character.Humanoid.HeadScale = 150

end
```

The code above tells us which game object activated the dialog (ie. player) and the response dialog that was chosen (choice) by the player. All of the information we need is contained within the header of the nameless function!!!

Since we have multiple choices available, we must manually check to see which one that the player clicked on. From the code above, if "c1" was the response dialog that was clicked on, then we can use that to manipulate the game object by updating the property values of player by making the health 50.

Notices that the "c1" is inside the if - elseif - else block. Have you seen the "c1" before? Go back to chapter 1.2 and you will see that "c1" is the name of the first clickable response dialog box. By giving it a name at chapter 1.2, we are now able to use it during our check to see if the player clicked on the first response dialog box.

Challenge
1. Include a cylinder part into the game and associated a dialog with this cylinder part 2. When your avatar character gets to within 5 inches of the cylinder part, the speech bubble should appear
3. The dialog should have two options that the player can choose from, either say "Yes" or "No".

Chapter 2. RPG Game

A Role Playing Game (RPG) is a game that react to the choices that you make. A very famous RPG game is called Star Wars, Knights Of The Old Republic. This game has won many awards:

1. Game Developers Choice Awards' 2004 game of the year
2. BAFTA Games Awards' best Xbox game of the year
3. Interactive Achievement Awards for best console RPG and best computer RPG.

This game is based on choice, either light or dark. Game objects react to you based on the choices that you make. For example, I approach a non-playable character (NPC) and we start a conversation. During the conversation, the game gives me the option of responding by following the light (ie. "Yes") or by following the dark ("No").

Based on your remark, the other game objects will either offer you a better deal (if you are buying) or charge you extra.

Another example is giving a rude remark to a NPC. However, in this case, the NPC has the answer that you are looking for. This answer satisfies a riddle and giving a rude remark to this NPC will mean you can't complete the riddle. Since the riddle is not satisfied, you are stuck.

Challenge - Create an RPG Via Dialog Box

1. Roblox allows the creation of an RPG by sensing the answer of the avatar. Write the code to catch the value that the avatar clicked on.
   1.1. If the player clicked on a good remark, then say "Hi-yah".
   1.2. However, if the player clicked on a not good remark, then say "I'm leaving now".

Super Challenge

1. If the player clicks on a good remark, the health pack increases its points by 10. This means that if you follow the path of light, the game reacts ***dynamically*** by increasing the health pack by 10.

2. However, if the player clicks on a not good remark, the health pack decreases its points to a value of 0. This means that if you follow the path of dark, the game reacts ***dynamically*** by making the health pack just a paper weight. It won't heal you at all!!!

This is why RPG games are so popular, you the player are in control.

Chapter 3. Camera Control Using Tween

One important aspect of video games is perspective. Gaining perspective allows us to create a game that looks like it is a 3D game. In reality, all video games are 2D games that change the angle and this change in angle introduces a faux Z axis.

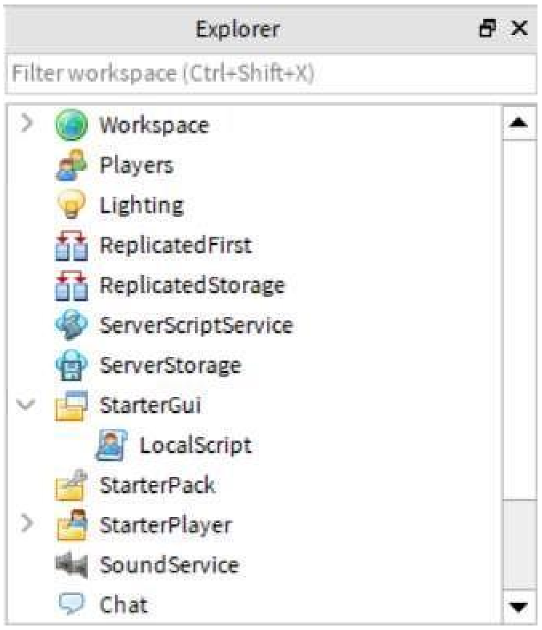
In Roblox, the camera follows the Avatar. We can change the angle of the camera by clicking and holding down the right button of the mouse and then moving the mouse in the left or right direction to spin the camera on an axis. This is manual control. However, Roblox also allows in-game control of the camera. How is this possible?

The camera is an object within the workspace. Like a cube part or the Avatar, the camera is an object with its own unique properties. We can tween these properties to create different in game experiences.

3.1. Script Location and Type

Manipulating the camera requires using a ***LocalScript*** and this ***LocalScript*** must be stored within the **"StarterGui"** folder.

Remember, a game object, like a cube, can have multiple scripts attached to it. A folder is similar. We can have multiple scripts attached to a folder.



3.2. Camera CFrame
Recall that CFrame stands for "**coordinate frame**". We previously used CFrame to teleport the Avatar by changing the CFrame of the upper torso. The camera can be repositioned by giving the camera new x, y, and z coordinates.

The lines of code to focus on below is boldened.

```
local myCam        = workspace.CurrentCamera
```

This means that we are accessing the game hierarchy and getting a reference to the camera. Having a reference to the camera means that we can access and manipulate its property values.

```
myCam.CameraType   = 6
```

This means that we want to put the camera under our control by having the camera be manipulated via our LocalScript

```
local tweenCon     = TweenService:Create(
myCam,
TweenInfo.new(5, Enum.EasingStyle.Linear, Enum.EasingDirection.Out, 5, true, 0),
      {

         CFrame = CFrame.new( 420, 810, 0 ),
         Focus  = CFrame.new( 420, 810, 0 )
      }
)
```

The code above uses the camera as the object to tween. Notice that we are using a dictionary to change the "**CFrame**" and "**Focus**" of our camera object. The "**Focus**" property is the "**point in 3D space where the camera is looking.** "

Go to the "**StarterGUI**" folder and put a ***LocalScript*** into it. This local script should be renamed "**camCon**". Inside the script, type the following code ***exactly***. When done, press F5 and see the tweening of the camera between two different positions and focus point.

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

local myCam        = workspace.CurrentCamera

--myCam.CameraType    = Enum.CameraType.Scriptable
myCam.CameraType    = 6

local tweenCon      = TweenService:Create(
myCam,
TweenInfo.new(5, Enum.EasingStyle.Linear, Enum.EasingDirection.Out, 5, true, 0),
     {

        CFrame = CFrame.new( 420, 810, 0 ),
        Focus  = CFrame.new( 420, 810, 0 )
     }
)


 -- wait
wait( 5 )
print( "about to manip camera" )

tweenCon:Play()


wait( 5 )
tweenCon:Cancel()

print( "done manip camera" )

-- ================================================ camera control 2
--
--
-- ================================================ local tweenCon
= TweenService:Create(
myCam,
TweenInfo.new(5, Enum.EasingStyle.Linear, Enum.EasingDirection.Out, 5, true, 0),
     {
          CFrame = CFrame.new( 420, 10, 0 ),
          Focus  = CFrame.new( 420, 10, 0 )
      }
)
wait( 10 )
print( "<<<<<<<<< 2 manip camera" )
tweenCon:Play()
print( "<<<<<<<<< 2 done manip camera" )
```

One Final Roblox Project

Instructions
1. **ALL requirements** are satisfied by writing Lua code unless it is labeled with the words
=> **(can be done with Editor instead of code)**
2. the final project is to be done individually and with minimal help from Sensei
3. all requirements and sub-requirements must be satisfied
4. student must show the implementation by demonstrating the game to the Sensei
5. student can use previous scripts as a reference

Requirements

Chapter 1. create a Story mode

1.1. the Story mode requires your avatar talking to a totem pole. The totem pole will describes the challenges that the human player must complete

Chapter 2. create a totem pole that presents a word or math problem using dialog and answers using choices.

2.1. if the player chooses the correct answer, then the totem pole will respond by giving one character of a keycode

2.2. this requirement has 4 total keycodes. This means that point 2.1. above is the first keycode and so 3 more totem poles are needed.

2.3. each totem pole will present a different problem and display different answers.

2.4. the player must visit all 4 totem poles and answer correctly all 4 challenges. Remember that each totem pole will give one character of the keycode.
2.5. once all 4 keycodes have been acquired, the player then goes to a house (see Chapter 3 below).
2.6. the house is another dialog box that will display the prompt, “Select the correct keycode”. There are two answer and one of the answers has all of the 4 keycodes.

For example, the 4 totem poles give the keycodes as A, B, C, and D.

Approach the house and the house will display the prompt, “Select the correct keycode”. The house will then display two answers and one of the answer is ABCD. The user selects answer ABCD. See Chapter 4 for what happens when the user clicks on the correct keycode.

Chapter 3. use composition of parts to create a house

3.1. initially, the door of the house will be locked. This means that the door is in a fixed x, y, and z position.

Chapter 4. unlocking the house by writing Lua code

When the player approaches the house, a dialog box will appear. The dialog will ask the player to enter the 4 keycodes acquired in step 2.

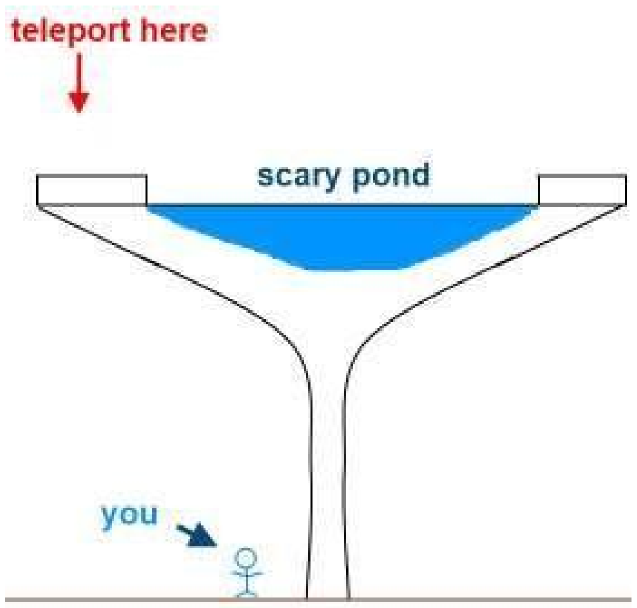
In order to unlock the house, the player must enter the 4 key codes in the correct order. The player already acquired the 4 keycodes needed to unlock the house by completing challenges (see step 2 above).

4.1. when the player has entered the 4 keycodes in the correct order, the door will slide right to left and the opening is used to enter the house. The animation of the door sliding from right to left requires writing Lua code to change the door's lateral position (is this x or y or z?)

4.2. inside the house is a teleportation portal that will teleport the player to the scary pond, which is chapter 5 (see below)

Chapter 5. creating the scary pond (can be done with Editor instead of code)

5.1. this requires creating a large body of water using the Terrain Editor. The scary pond must be made to look like the image below. The player **does not have enough** JumpPower to jump from the bottom and reach the top. Manipulating the jump power to get to the top of the Scary Pond is not an available option. **The only way to reach the top is to satisfy Chapter 4 from above.**

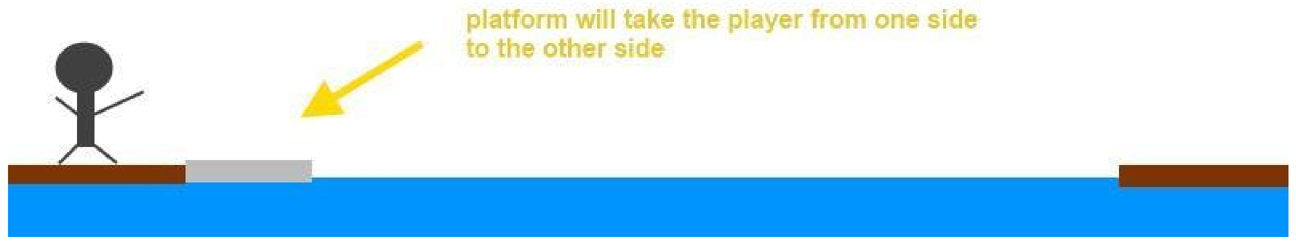


Chapter 6. crossing the scary pond

It would be easy to cross the scary pond by jumping in and swimming. However, this is too easy. Instead, the scary pond will be crossed by jumping on a platform. The platform then takes the player across.

6.1. This requires 3 more totem poles that the player approaches to begin the dialog. A 4th totem pole will ask “Select the correct keycode”. This is similar to Chapter 2 and 3.

The player can **ONLY** cross the large body of water by traveling on the platform. **When the game starts, the platform does not exist.** However, the platform **only appears** if the player can correctly select the correct keycode from totem pole 4.



6.2. Once the player selects the correct keycode from totem pole 4, then the platform will spawn dynamically (HINT: use cloning). Write the Lua code to have the platform spawn when the player selects the correct all 3 questions are answered correctly.

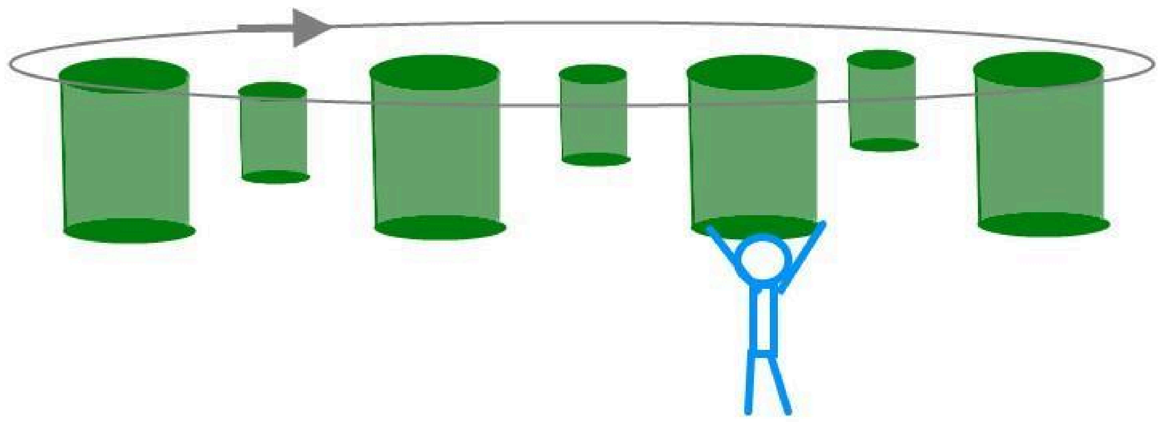
6.3. The player then jumps onto the platform and the platform will take the player across the scary pond.

Write the Lua code to have the platform travel autonomously from left to right and this will be done forever. Is Tween needed to make the platform move autonomously?

7. Cross the valley
The only way to cross the valley is to build a bridge. **When the game starts, the bridge does not exists.** When the player presses the 'b' keyboard button, the bridge will be assembled by repeatedly placing small platforms in a linear fashion and this will create the bridge.

*** **NOTE** *** the bridge **CANNOT** be created by using a single cube that is stretched from one plateau to another. **This requires writing Lua code that uses cloning and manipulating the positioning of the clones** (x and y position? Vector3?)

8. Skyway Rail - Write Lua Code
This requires creating a rotating group of vertical cylinders. The vertical, rotating cylinders are of color green and can be made of any material you like.



The cylinders will move in an oval path (see picture above). The grey indicates the path that each cylinder should follow. The grey **ONLY** shows the path as an example - **IT SHOULD NOT BE** implemented in the game. The cylinders should follow that path forever.

The goal is for the player to jump (use spacebar on the keyboard) and link up with one of the vertical cylinders. The player must jump from one rotating platform to another without falling off or else the player starts over again.

Challenge

1. Write the Lua code to implement Skyway Rail.
2. Cloning must be used
3. Tween service must be used

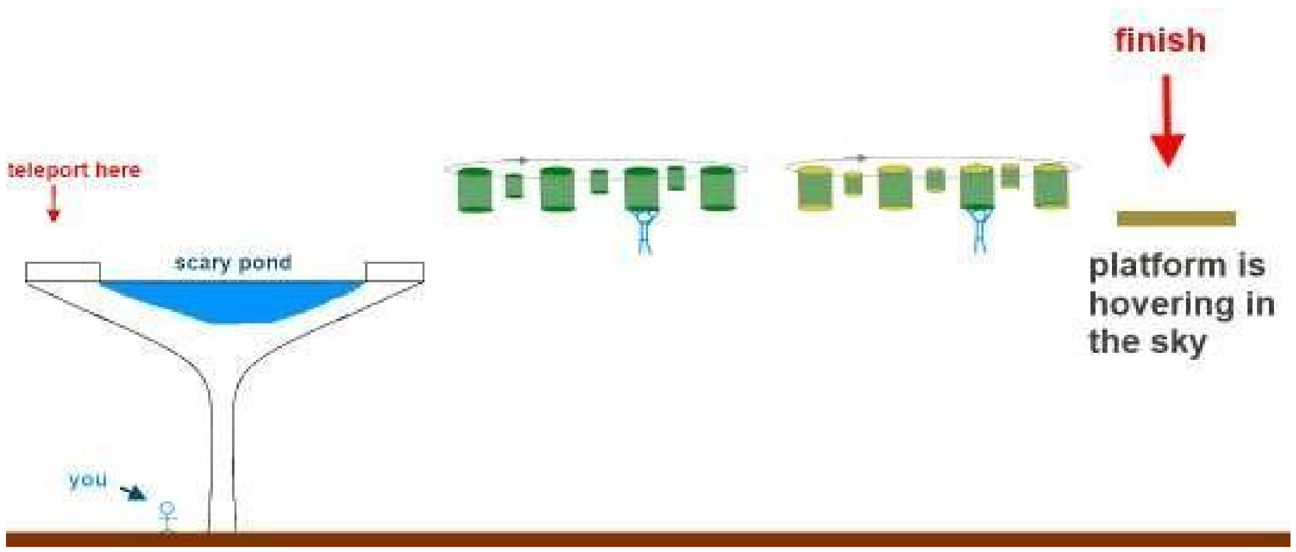
9. Multiple Skyway Rail - Write Lua Code
Do the same from requirement 10 and create another set of Skyway Rails. This set of vertical, rotating cylinders are of color gold and are made of any material you like. The goal is for the player to jump from one skyway rail to another skyway rail. There should be at least 4 sets of Skyway Rails and each set is different from the other set.

Challenge

1. Write the Lua code to implement Gold Skyway Rail.
2. Cloning must be used
3. Tween service must be used

10. Finish platform - Write Lua Code

There exists a gold platform and this gold platform is on the other side of the Gold Skyway Rail. Write the Lua code to have the platform hover in the sky. The goal is to have the player jump from the Skyway Rail onto the Gold Skyway Rail. Finally, jump onto the gold platform to complete the game.


finish
teleport here
scary pond
platform is hovering in the sky
you