Week 5 Reflection

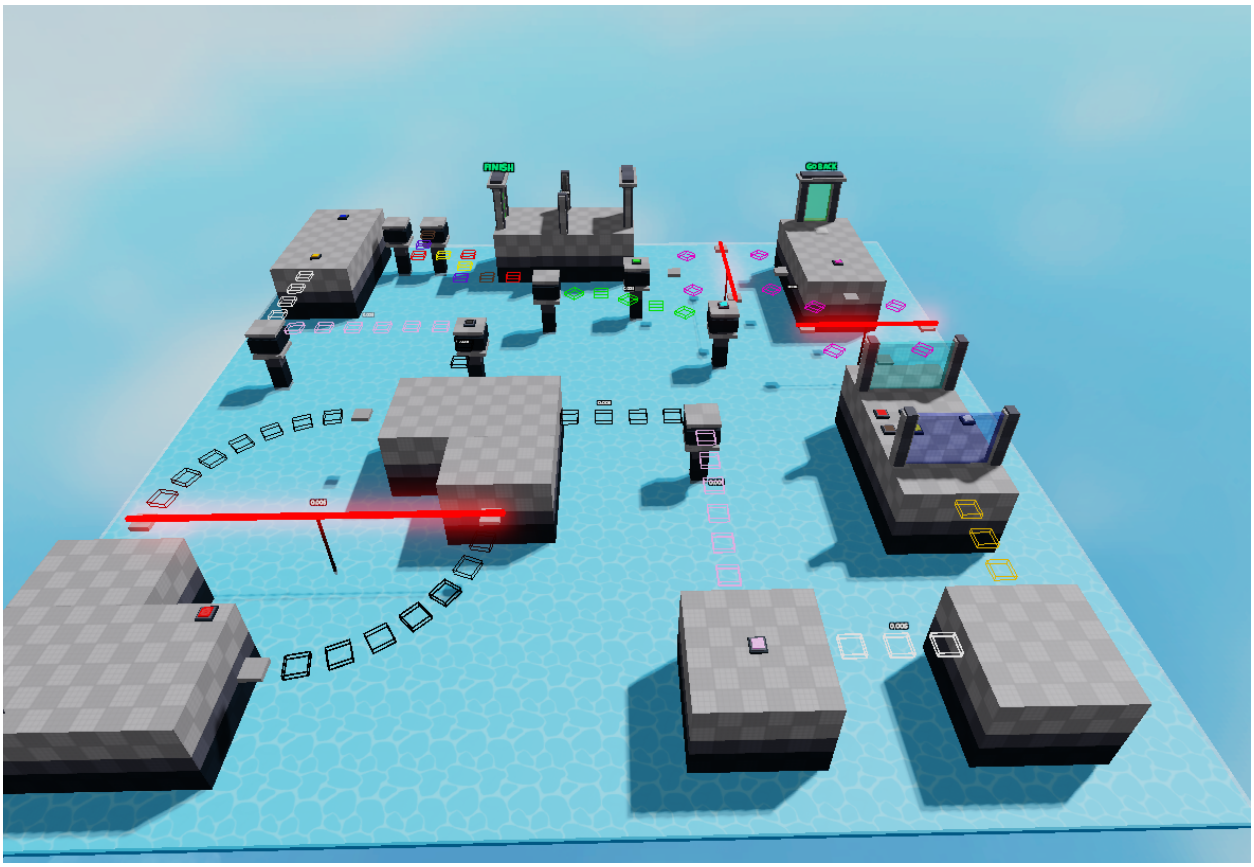
Throughout the past few weeks, I’ve most certainly come a long way.

I began my journey of LUA game development in utter confusion, not knowing how the server-client relationship works, not knowing how to format or organize my scripts, and I doubted whether this was a good idea. As I went on though, through trial and error, I began learning things that made scripting a functional game a lot easier. I also went through some challenges, in which I had to completely scrap some of my code. Still, I learned the most through my failures and continued to learn day in and day out.

Now, I have a fully functional game that I am proud to show and present. The Datastore* system hasn’t been set up on the game yet, but I’ve studied and messed around with Datastores prior. I have a working knowledge.

The game that I am presenting is called **Two Player Obby**

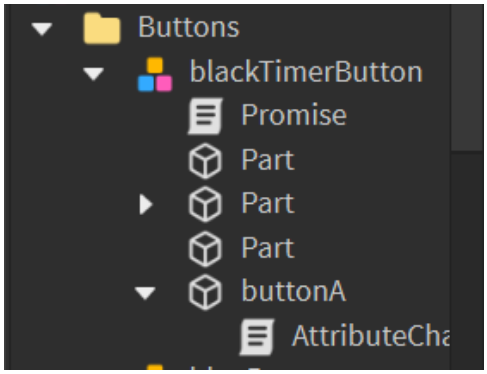
The word Obby is a term that is used within game development to reference an obstacle course — a game where players have to dodge and jump around obstacles in order to reach the end. If you hit an obstacle, you respawn and have to try again.



Perfect-Timing **Course**

The above image is a screenshot of the “Perfect-Timing” course. Players have to step on the buttons to activate the corresponding parts, which have a timer associated with them. Once the timer runs out, the parts disappear.

I organized my scripts like so:



Each button is individually labeled for what function it has. Each button also has a corresponding “promise,” that states what has to be done once the button has been stepped on. It’s like a pressure plate. The listener for the button is the corresponding “AttributeChanger,” which works like so:

```
local part = script.Parent -- define the part as the parent of the script
local beingStoodOn = false
local hitPartDetect = "HumanoidRootPart" -- change to whatever we want 2 listen for
-- in this case, every player has a corresponding "HumanoidRootPart", so we're listening f
or the player to step on the button


part.Touched:Connect(function(hit)

  if hit.Name == hitPartDetect then
    print("fired")
-- print statement for debugging

    part:SetAttribute("stoodOn", 1)
-- I added an attribute to the part which is called "stoodOn"
  end
end)


part.TouchEnded:Connect(function(hit)
-- function for when the touch event has ended, signaling when the player steps off of tth
e button
  if hit.Name == hitPartDetect then

    part:SetAttribute("stoodOn", 0)
  end

end)
```

The Promise itself, which carries out the appearing and disappearing of the parts, looks like so:

```
local buttonA = script.Parent:FindFirstChild("buttonA")
local eventModule = require(game:GetService("ServerScriptService"):WaitForChild("ModuleScr
ipt"))
local overallParent = script.Parent.Parent.Parent
```

```
local targetPartsFolder = overallParent:WaitForChild("Doors&Paths"):WaitForChild("blackPar
ts")
local time = 10
-- iterate through state values within the model and act accordingly

local function fire()
  local doorPart = targetPartsFolder:WaitForChild("Door"):WaitForChild("DoorPart")
  coroutine.wrap(function()
    eventModule.doortimer(doorPart, time, doorPart:WaitForChild("BillboardGui"):WaitForChi
ld("TextLabel"))
  end)()

  local billboardPart1 = targetPartsFolder:WaitForChild("Path1"):WaitForChild("OutlinedPar
t"):WaitForChild("guiPart")
  coroutine.wrap(function()
    eventModule.timer(billboardPart1, time, billboardPart1:WaitForChild("BillboardGui"):Wa
itForChild("TextLabel"))
  end)()

  local billboardPart2 = targetPartsFolder:WaitForChild("Path2")
  coroutine.wrap(function()
  eventModule.timer(billboardPart2:WaitForChild("primaryPart"), time, billboardPart2:WaitF
orChild("BillboardGui"):WaitForChild("TextLabel"))

  end)()


  local billboardPart3 = targetPartsFolder:WaitForChild("Path3")
  coroutine.wrap(function()
    eventModule.timer(billboardPart2:WaitForChild("primaryPart"), time, billboardPart3:Wai
tForChild("BillboardGui"):WaitForChild("TextLabel"))

  end)()

  for i,v in targetPartsFolder:GetDescendants() do
    eventModule.appear(v)
    if v.Name == "DoorPart" then
      eventModule.vanish(v)
    end
  end
  task.wait(time)
  for i,v in targetPartsFolder:GetDescendants() do
    eventModule.vanish(v)
    if v.Name == "DoorPart" then
      eventModule.appear(v)
    end
  end
end

while true do
  task.wait()
  if buttonA:GetAttribute("stoodOn") == 1 then

    fire()
```

```
    end

end
```

As we can see, I imported a separate eventModule that I wrote in order to utilize helper-functions that I will use constantly. This helps organize my code.

The rest of the code within the game will be put on the Github repository. These are the two primary scripts that keep the game running.

*Datastores are the ROBLOX equivalent of Databases, and are stored via their API.