UNITY: MONO-SERVICE HIGH-LEVEL DOCUMENTATION

BY: HUMANITARIAN OPERATIONS

CONTENTS

OVERVIEW:

This system was developed by the Unity Game Programmers within Humanitarian Operations for the game designers to develop different kind of games without having to write any code whatsoever.

The name for the system is Mono-Service which is short for Mono-Behaviour Service which is the Unity's Standard API.

This system adds extra functionalities and features "Components" to Mono-Behaviour which we call them services to avoid rewriting the same functionalities in every new project.

REQUIREMENTS:

For you to use this system, you will need to know the basics of using Unity:

- Work with Unity default components for Box Colliders, Mesh Renderers, etc...
- Build the project for different platforms.
- Importing and exporting assets.
- Very basic knowledge of coding aspects in general like parameters, variables, etc…

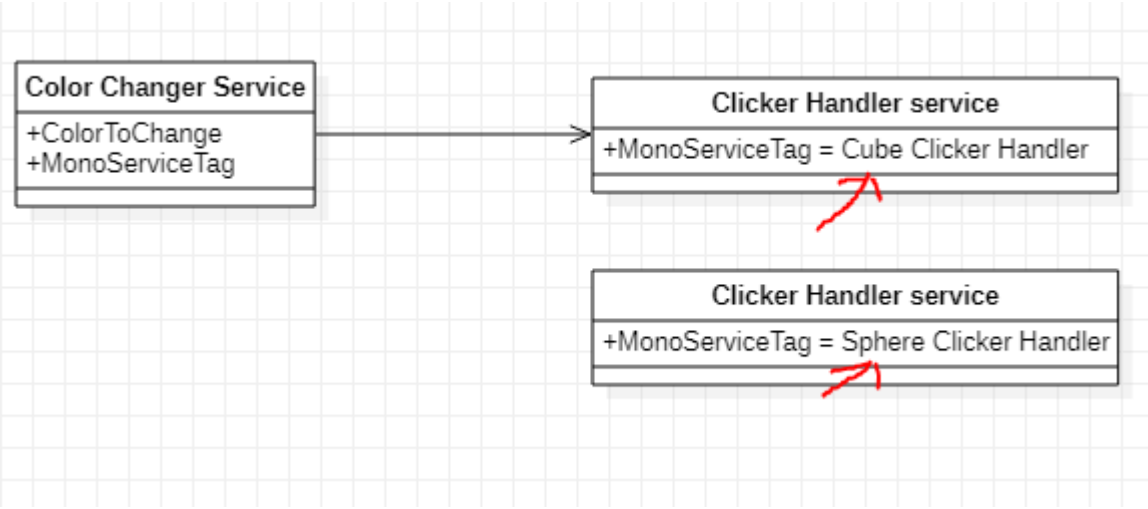
SYSTEM CONCEPT EXAMPLE:

The system is responsible for tying up the game mechanics together for them to be synced and to listen to each other.

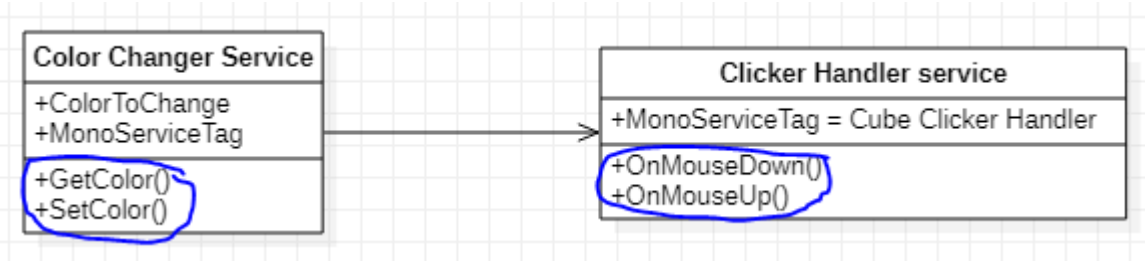
Let's say for example you wanted when you click on a cube, another cube will change its colour, you will need 2 services in this instant which are a service to hander the clicks that will be attached to the first cube and another service to change the colour that will be attached to the other cube and we need the colour changer service to listen or to be synced to the clicker handler service.

These services will usually hold settings that you configure like what colour will it be changed to in the colour changer service case. We call these parameters.

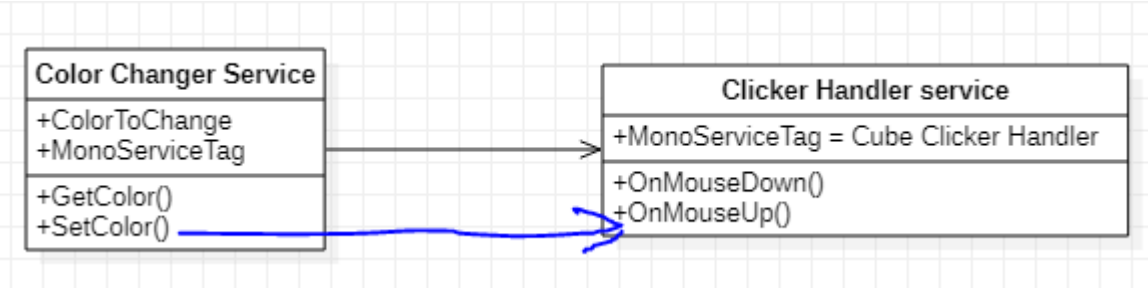
Every single service will have a tag name so the system knows which service you would like to listen to, because let's say you have 2 clicker Handler services one is for a cube and the other is for a sphere but the colour changer would like to listen to the cube clicker handler, you name the first clicker service "Cube Clicker Handler" and the second service "Sphere Clicker handler" and then hook the colour changer service to the Cube Clicker handler Service.



Every Mono-Service will have a list of commands, the colour changer service will have a command for getting the colour and a command for setting the colour and the clicker handler service will have a command for on mouse down, meaning when the user will start clicking on the more and on mouse up when the user stop pressing on the more button.

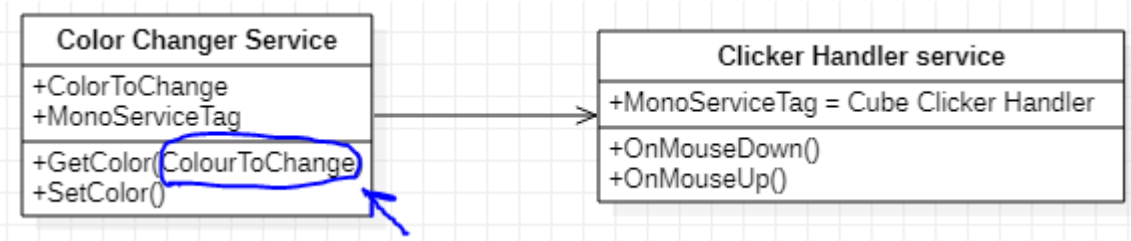


In addition of hooking a mono-service to another mono-service, you will also need to hook their commands from one another. The Set Colour in the colour changer service can be hooked to neither On Mouse Down nor On Mouse Up in the Clicker Handler Service. I will choose to hook it up with the On Mouse Up in the case.

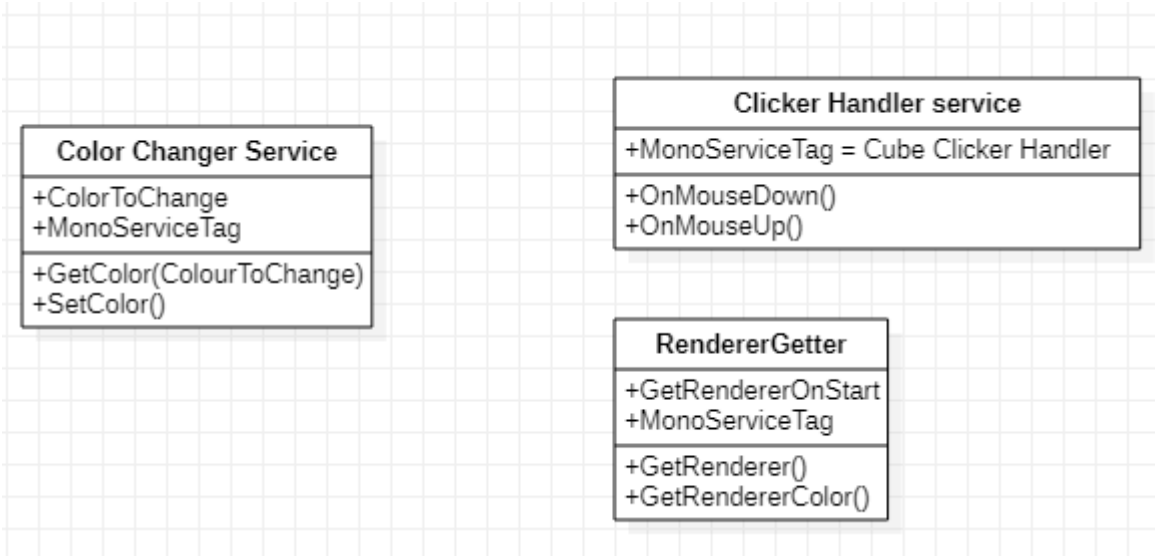


Now the colour of the cube will change when the other cube was clicked on by the player.

Bear in mind that certain commands might have parameters within them. This will allow you to pass settings in between Mono-Services.

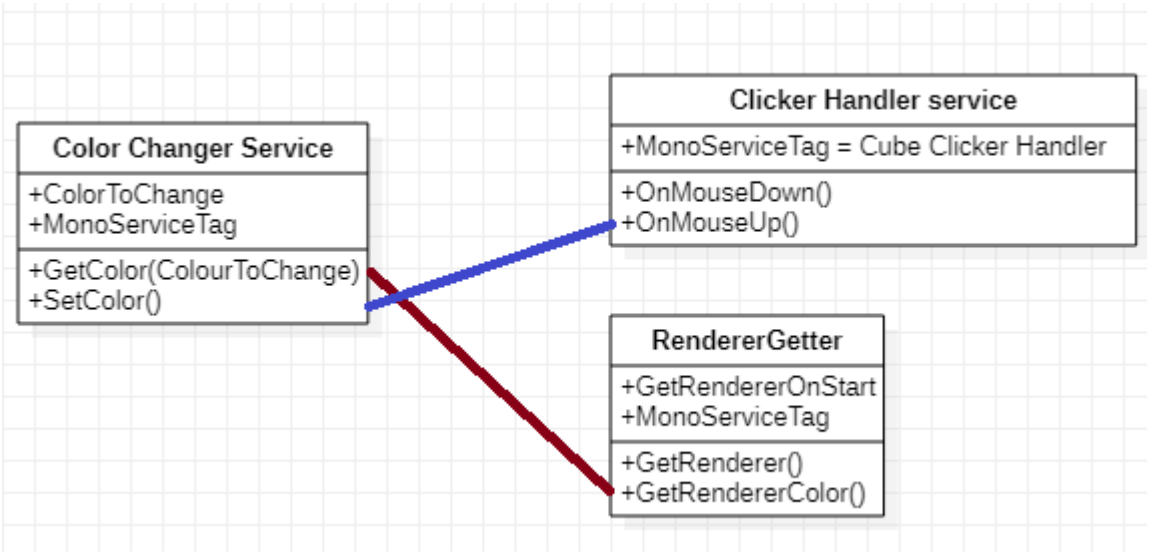


Let’s say the colour that you want to change is based on another cube colour in the scene so in that case we need to add another service that can get the colour of the new cube. The Mono-Service needed is Renderer Getter which will get the renderer component of the cube along with its colour. In this case we will be having 3 Mono-Services in total:



Note: Get Renderer on start means that the service will get the renderer of the cube when the game starts.

All we must do now is to hook them together, the Get Colour Command in the Colour Changer Service will be hooked to the Get Renderer Colour Command in the Renderer Getter and the Set Colour Command in the Colour Changer Service Will Be hooked to the on mouse up Command in the Clicker Handler Service.



Note: all the service commands will only pass one parameter. To keep it as tidy and simple as possible.

NAMING CONVENTIONS:

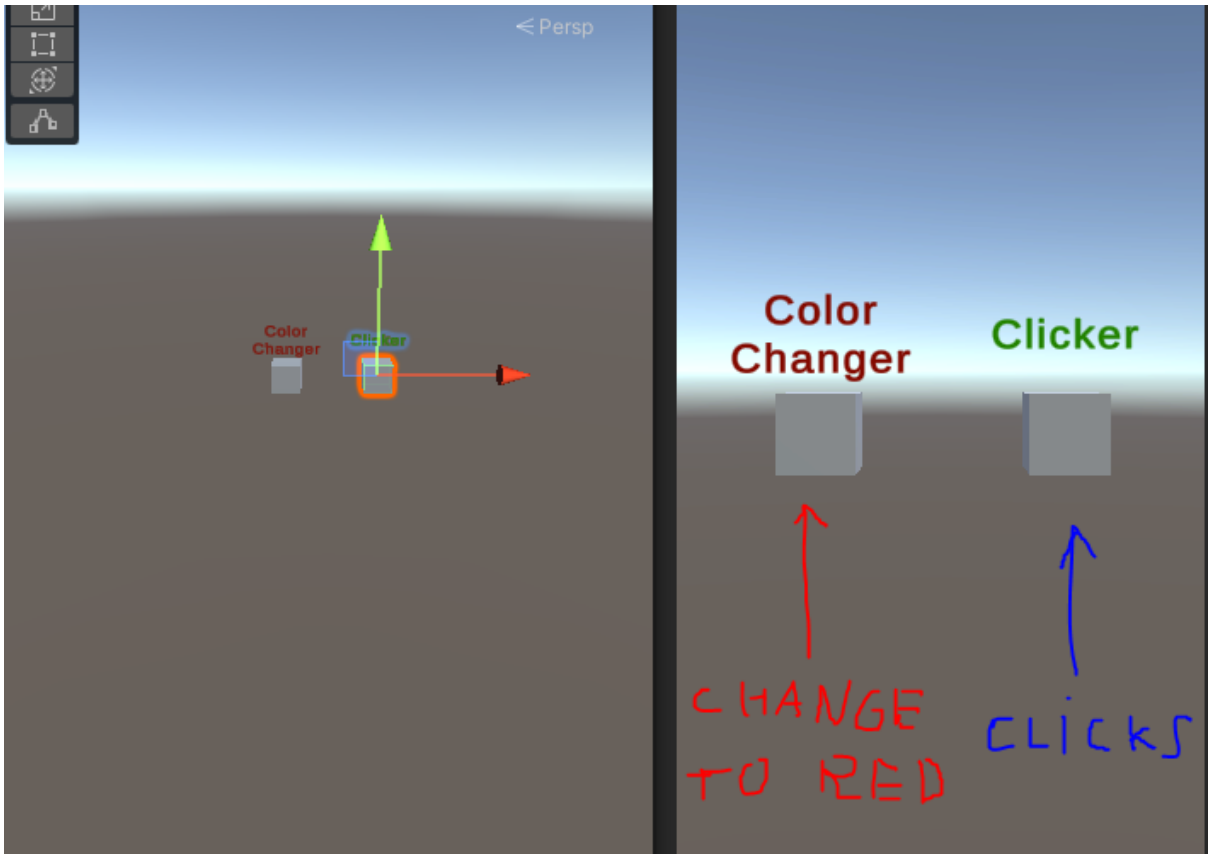
- The mono service tags are usually the game object name following by the mono service name, for example when a cube has a clicker, the tag will be CubeClicker.
- Avoid using spaces when naming tags.
- Always use Pascal Casing.

SYSTEM PRACTICAL EXAMPLE:

Now that you understand the basic concept of the system, we will move on to apply this concept into Unity to have a better understanding on how to use the system in practise.

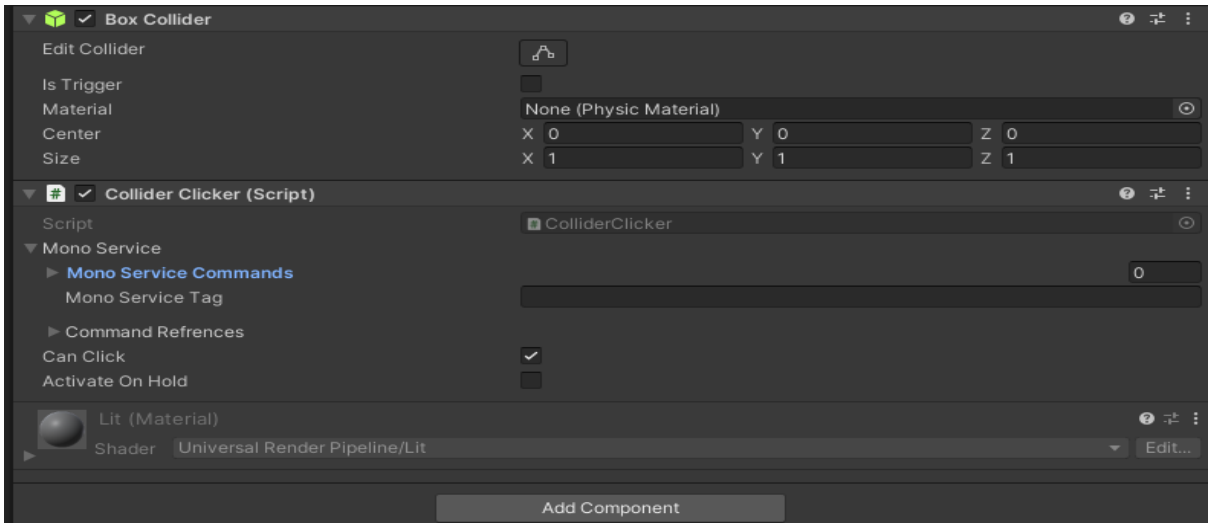
You will first need to have access to the Mono-Service System Package (Please refer to your team leader).

Let's start by creating a new scene and add 2 cubes, one will be responsible for listening to clicks from the player and the other will be responsible for changing its own colour, when the player clicks on the cube clicker, the other cube will change its colour to red:

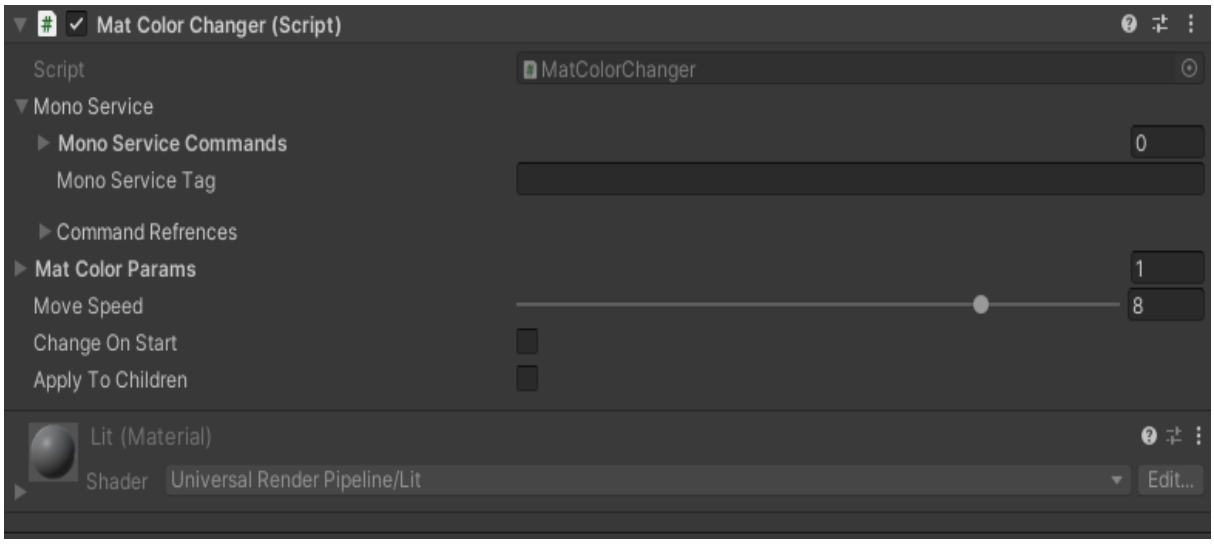


The clicker cube will hold the collider clicker component and the colour changer cube will hold the mat colour changer component:

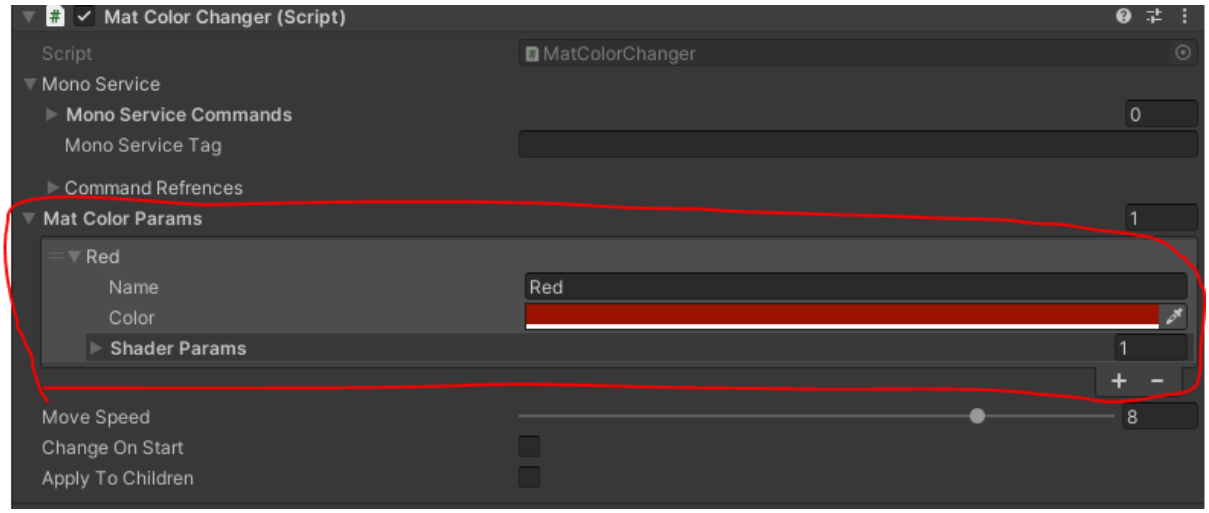
Cube Clicker:



Cube Colour Changer:

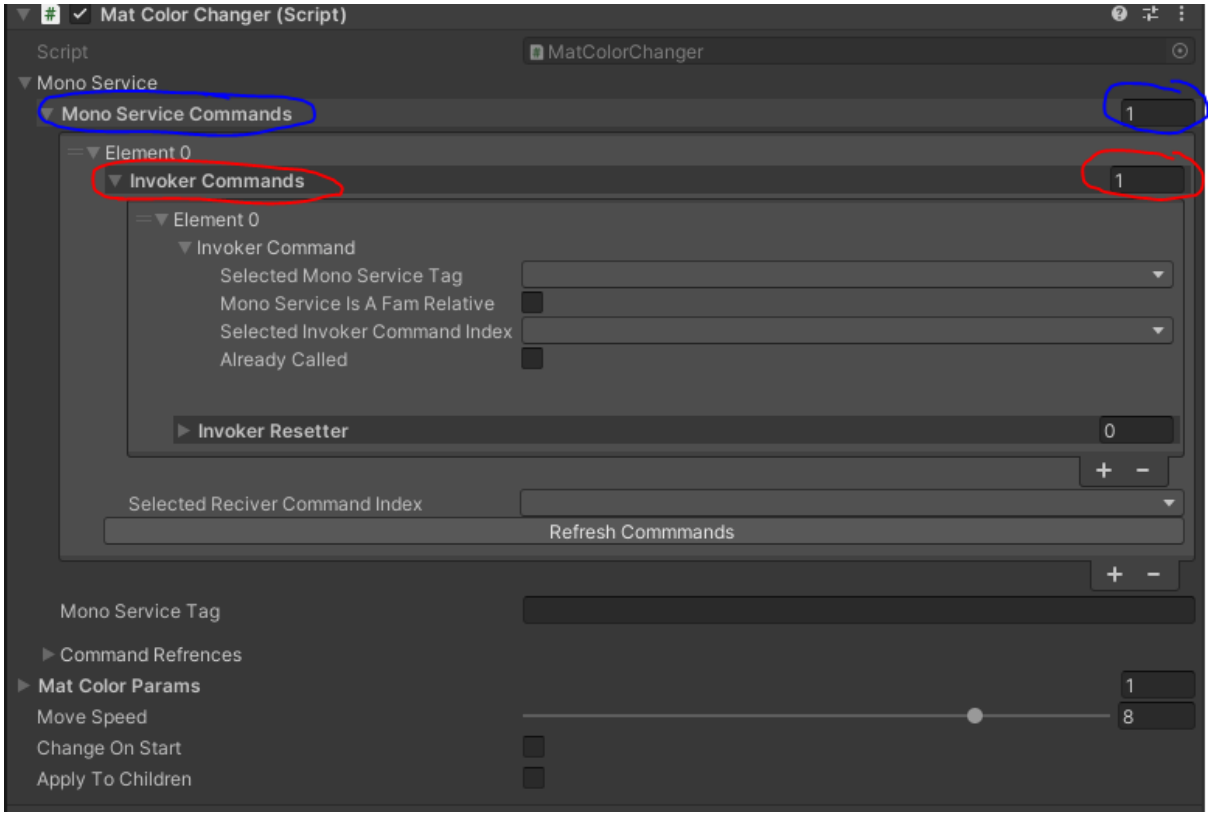


When you expand the Mat Colour Params in the mat colour changer, you can add colours to change them to:

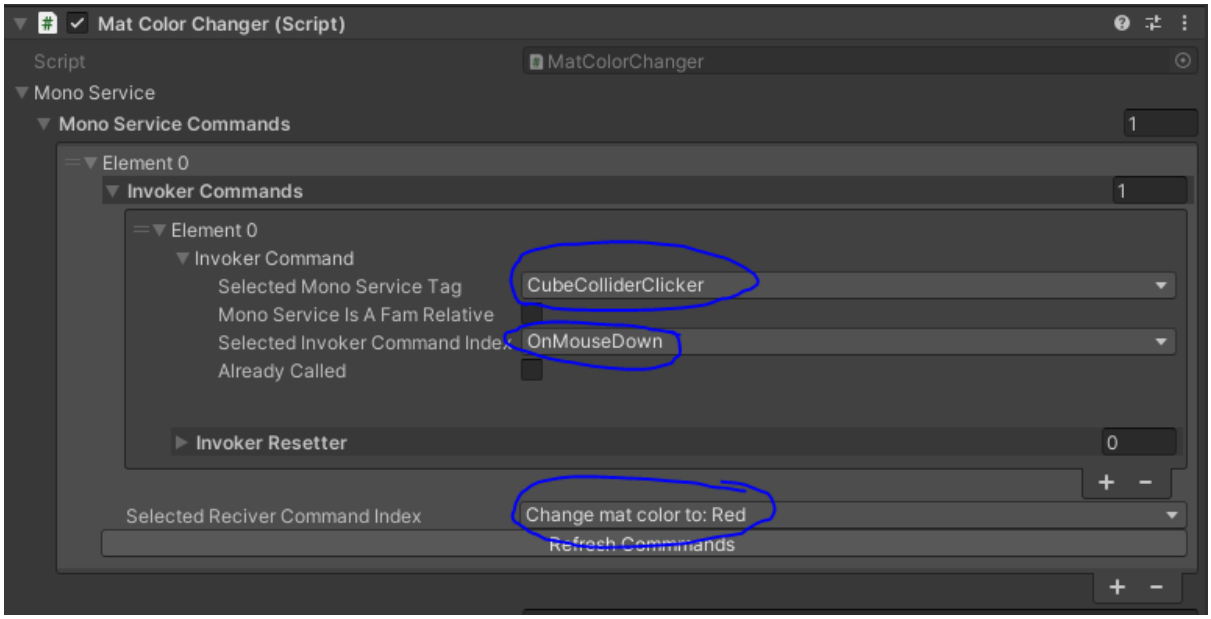


Now we need a way for the Mat Colour Changer Component to listen to the Collider clicker component when it will be clicked. We need to first go back to the collider clicker and assign a tag to it so the Colour Change component can find it:

Now, let's go back to the Mat Colour Changer component to add commands for it to listen to the click, add one Mono Service Command and one invoker command like this:



Now, click on the refresh commands button to refresh its command names:

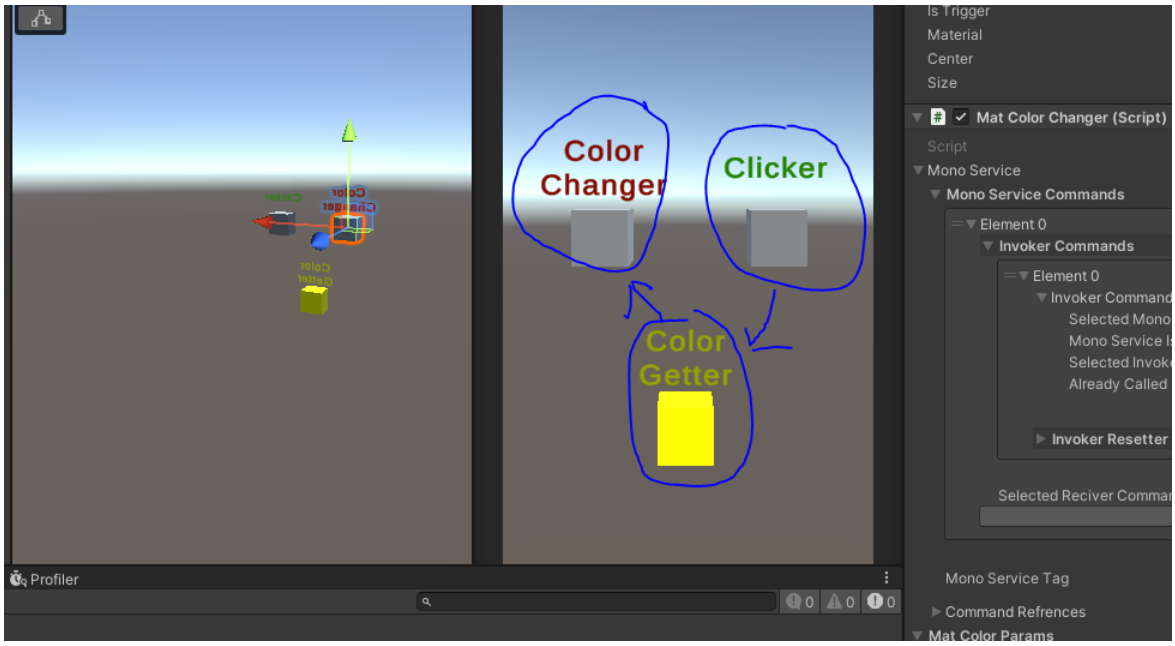


- The selected mono service tag is the tag of the component that you would like to listen to.
- Do not worry about the mono service is a fam relative just yet.
- Selected invoker command index is which command would you like the component to listen to because each component can contain multiple commands.
- The already called parameter will become true when the command is executed which is pressing of the cube collider in this case.
- Do not also worry about the invoker resetter just yet.

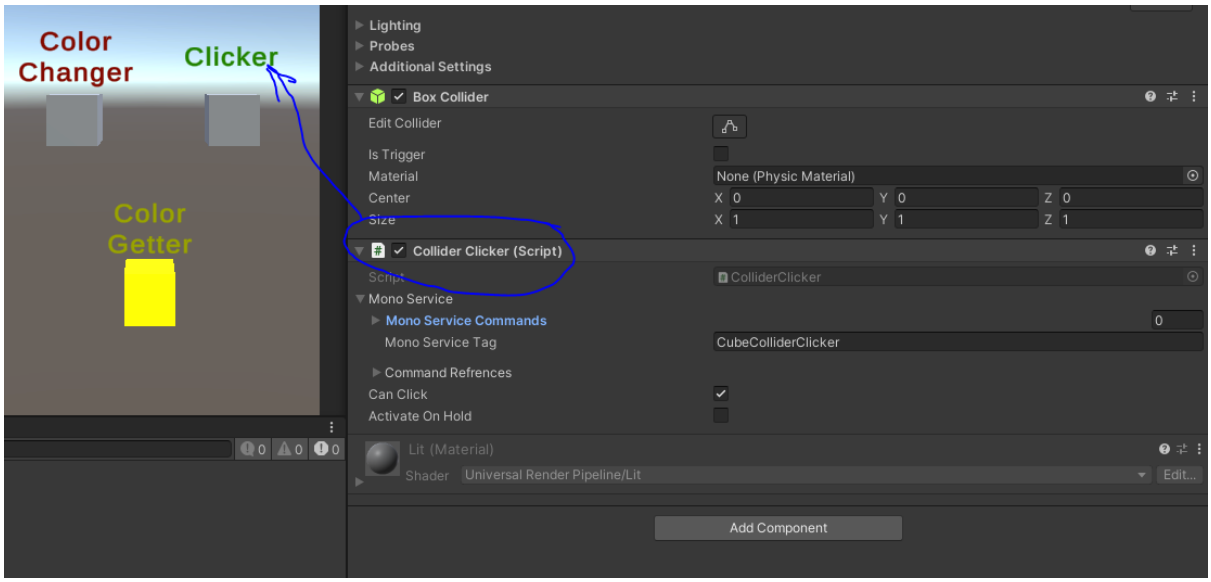
Make sure to always refresh the commands.

And now, you are set to go. Please check the result in this video here: https://streamable.com/3zk93f.

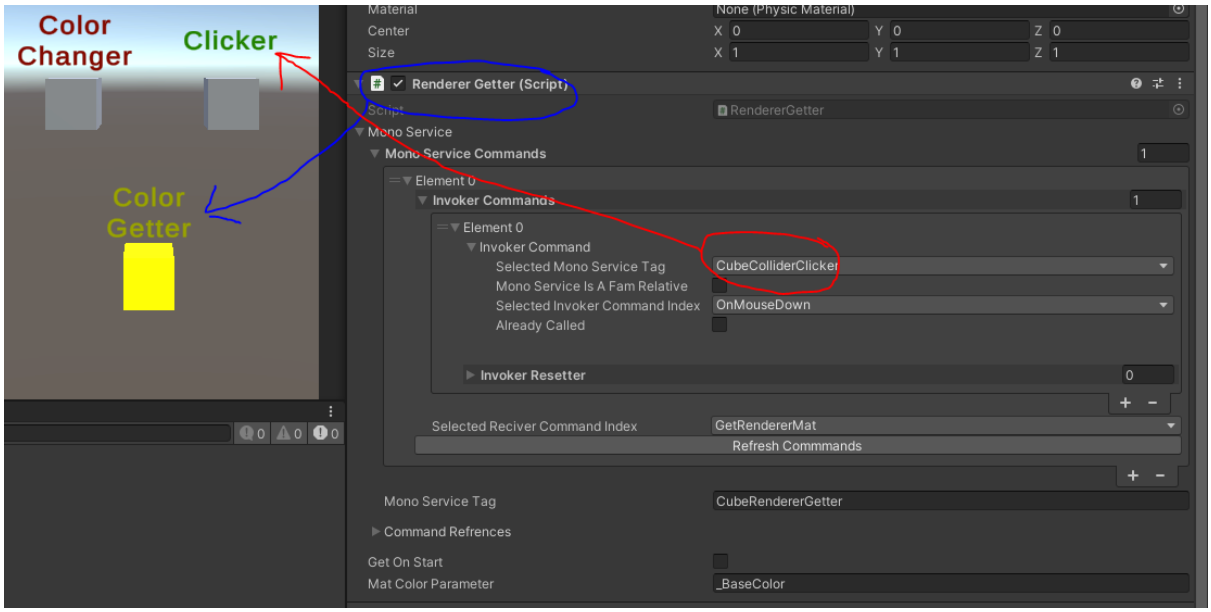
As mentioned before, the system can only pass an object between commands to avoid complexability. Let's say we would want to get a colour of a game object and pass it into to the command. We can still use both of the cubes in the previous example but we will add a new cube with a renderer getter component to get the material of the renderer and pass it through the mat colour changer to get it's colour:



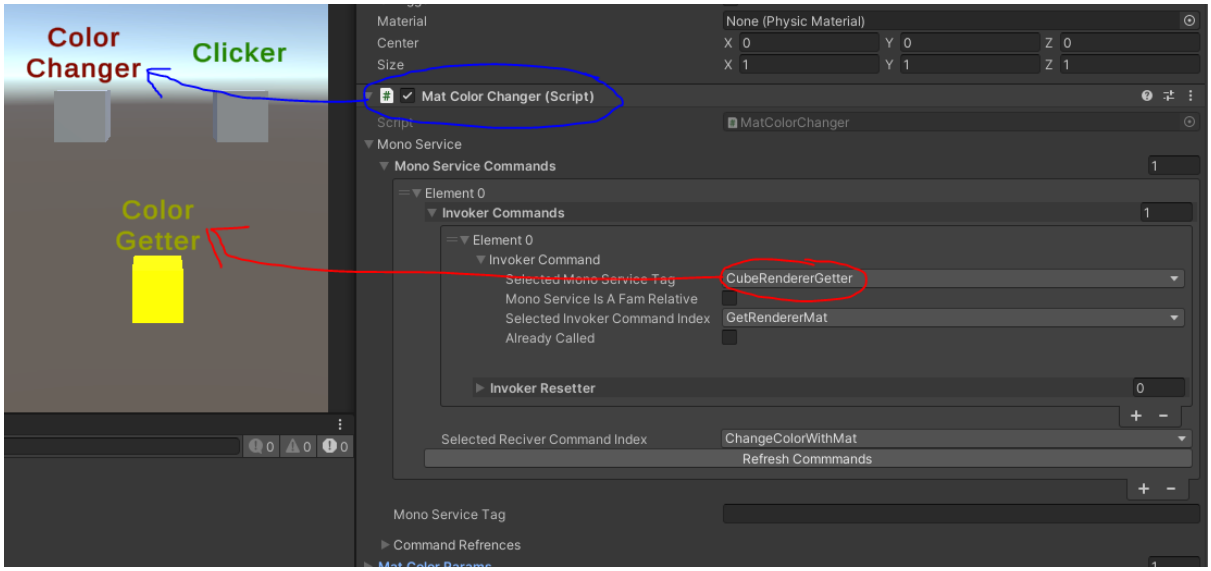
The clicker will hold the clicker component to listen to clicks:



The colour getter will hold the renderer getter component where it will get the material from the mesh renderer, the renderer getter will be receiving commands from the clicker to get the material:

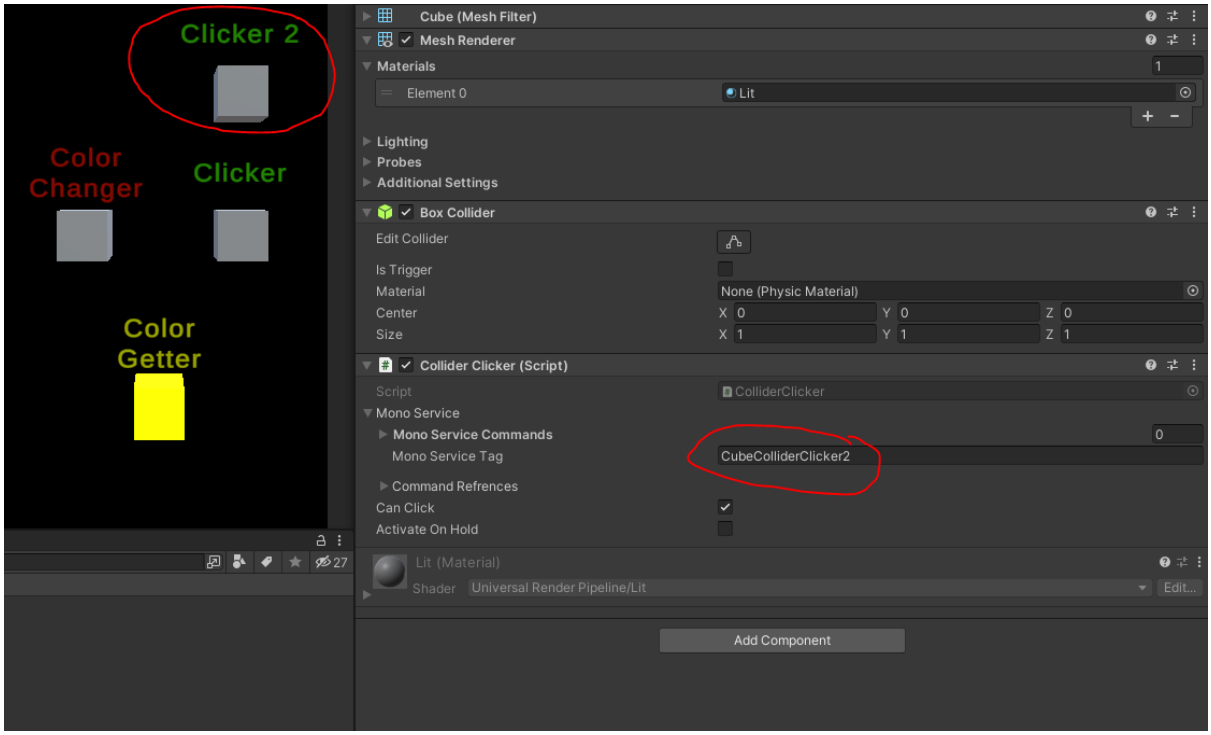


And then, the colour changer will get the material colour from the renderer getter component and change its own material colour:

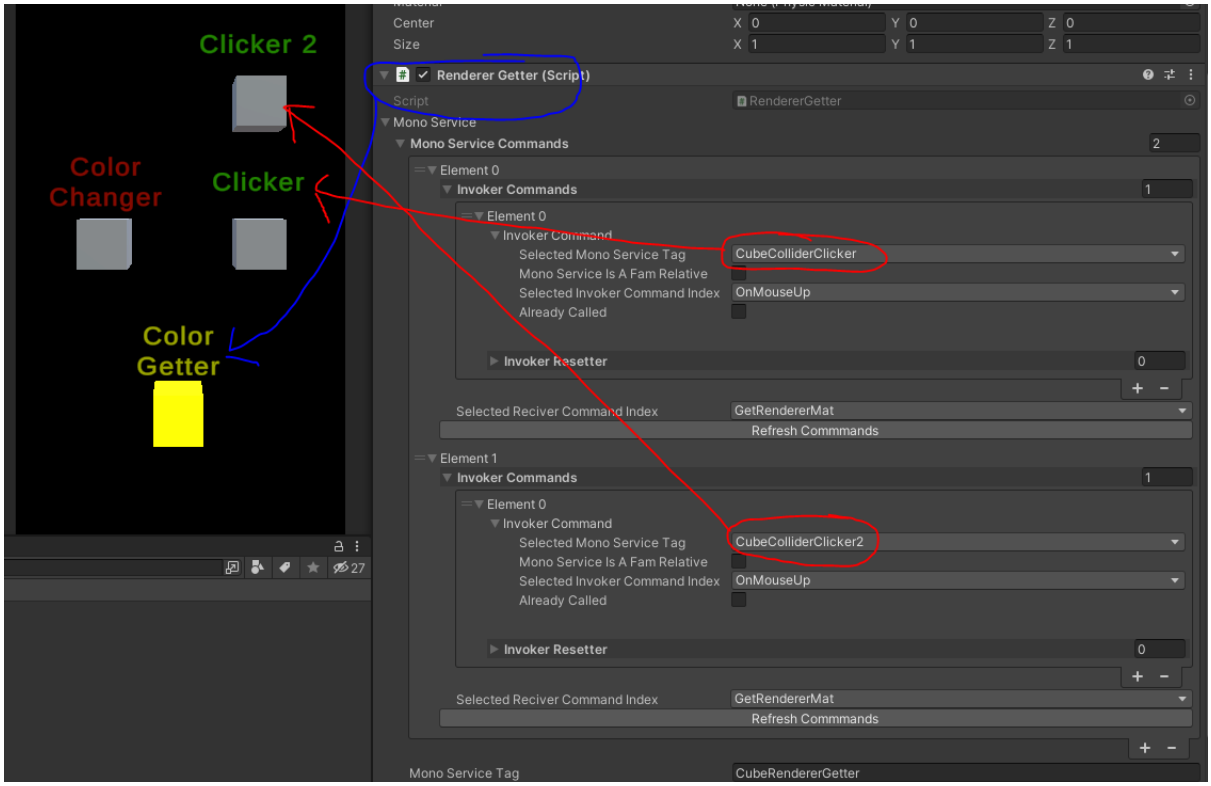


So now, when clicking on the clicker, the colour getter will get the material and send it over to the colour changer to use the passed material colour to change its own material colour to it, check out the final result here: https://streamable.com/n5mb0w

A single component can receive commands from multiple other components. I have added an extra clicker with its own tag:



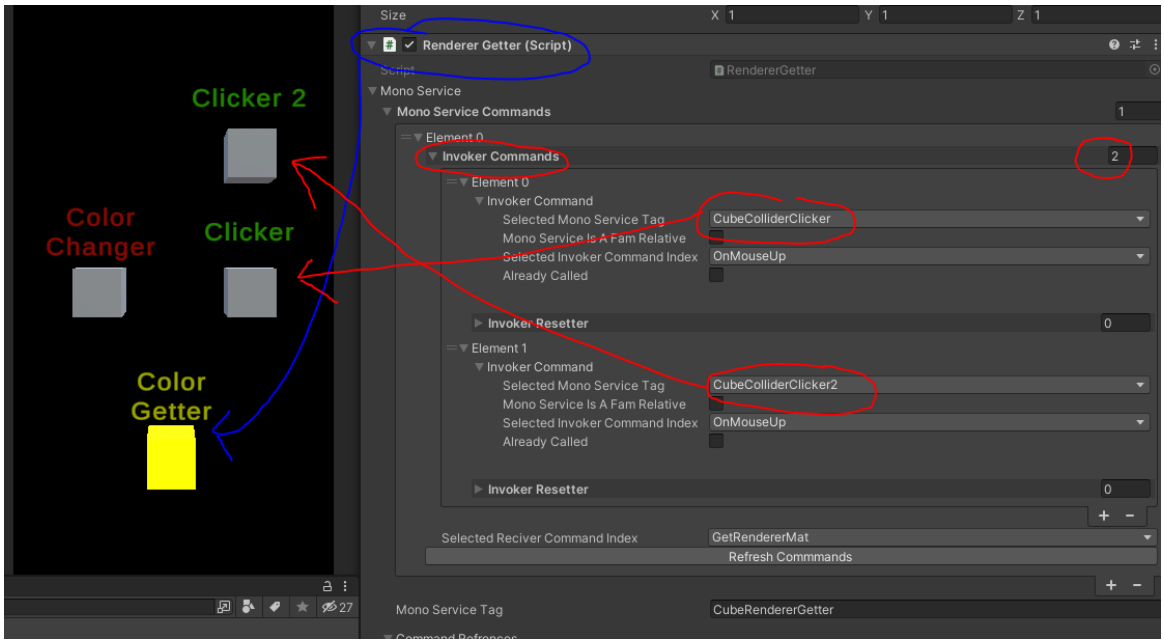
You will have you add an extra mono-service command in the renderer getter component to receive commands from both clickers:



This acts as the “OR” operator where the renderer will get its material when clicker1 OR clicker2 were clicked.

You can check the result here: https://streamable.com/5jigpn

The system also provides the “AND” operator, the way to do it is to add an extra invoker command instead of adding mono-service command:



You can also check the result here: https://streamable.com/gzqa48

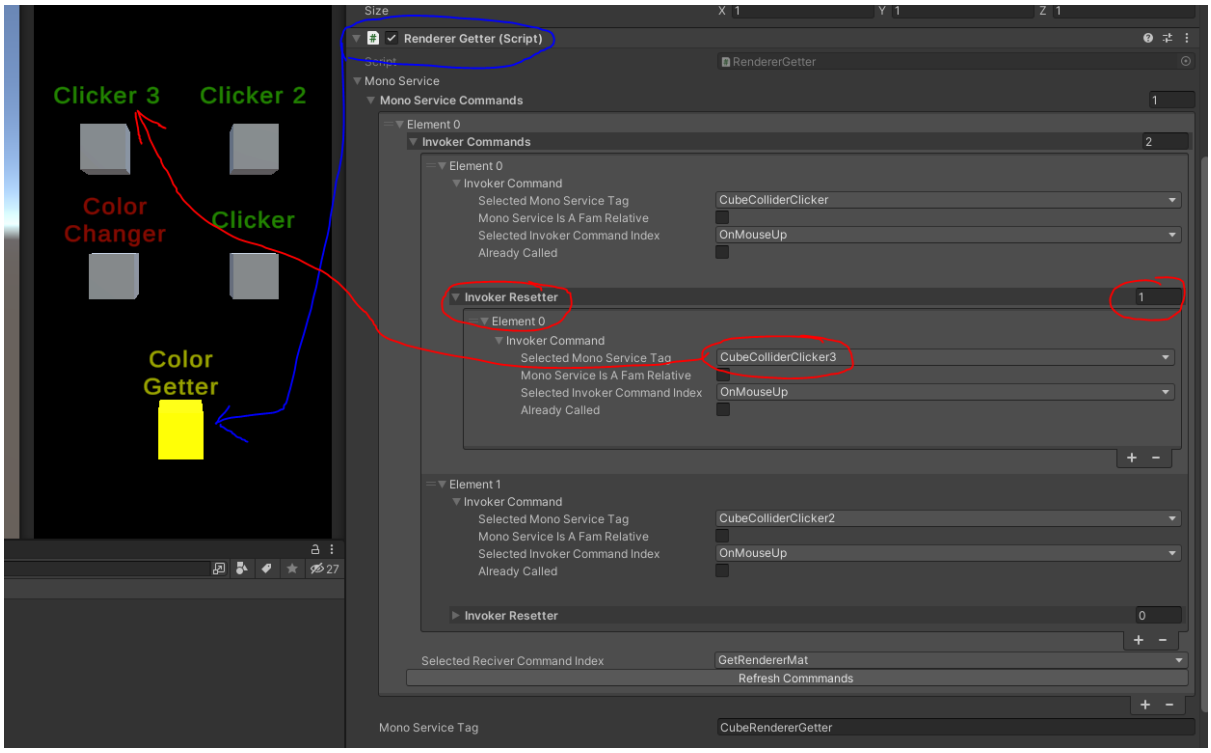
Note: if there are multiple invoker commands, the last invoker command in the list will be passing actual the object. Like in the last example the CubeColliderClicker2 will be passing a parameter object hypothetically speaking because clickers don't usually pass any objects.

The "Already Called" Boolean parameter is to indicate that the command has been received and executed for example, the Already Called parameter in the clicker1 command will swich to true if the player clicks on the clicker cube.

You can check the result here: https://streamable.com/uf1xib

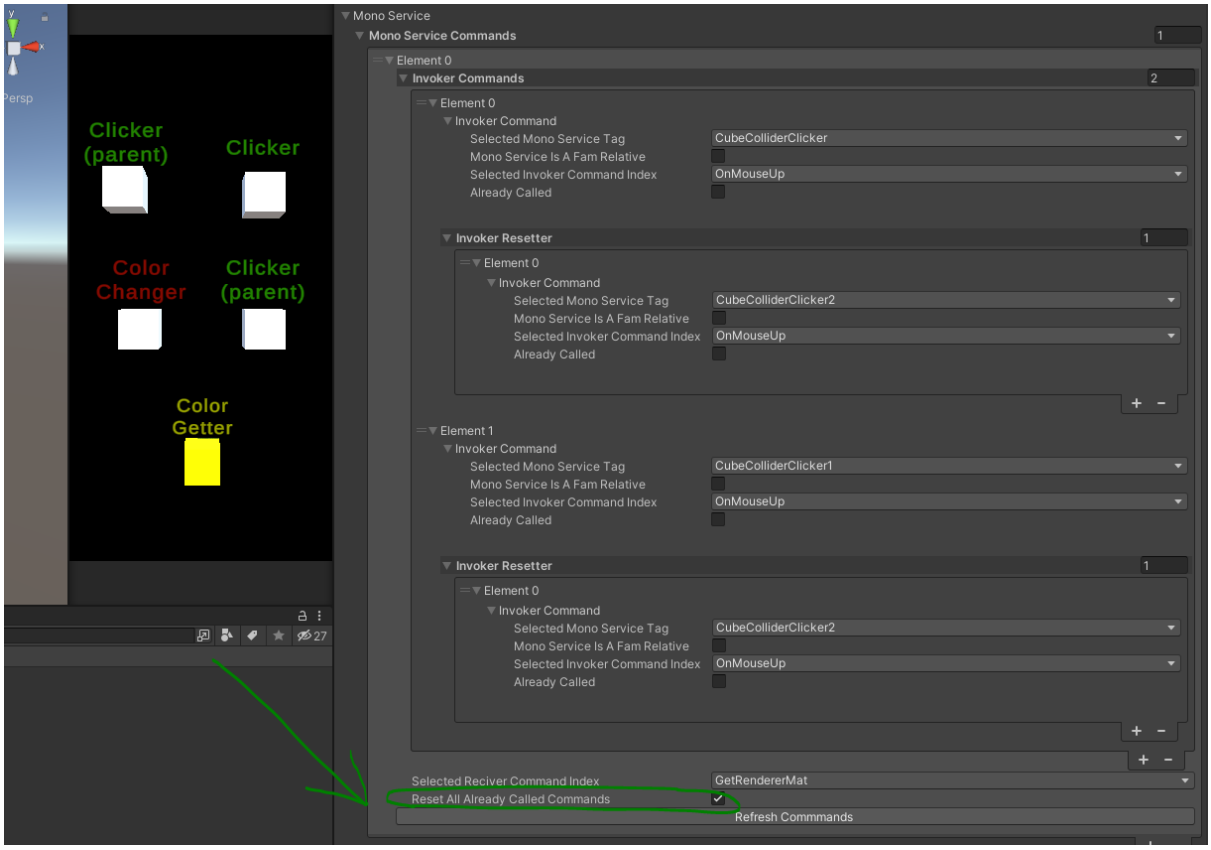
In some cases, you would want to reset or turn the "already called" Boolean parameter off, the way to do it is to use the "Invoker Resetter" where it will receive commands from any invoker.

I added a 3rd clicker so it will turn the "already called" parameter off in the clicker1 invoker command:



You can check the result here: https://streamable.com/wui1s9

There is another Boolean parameter in the Mono-Service Command which is "Reset All Already Called Commands":

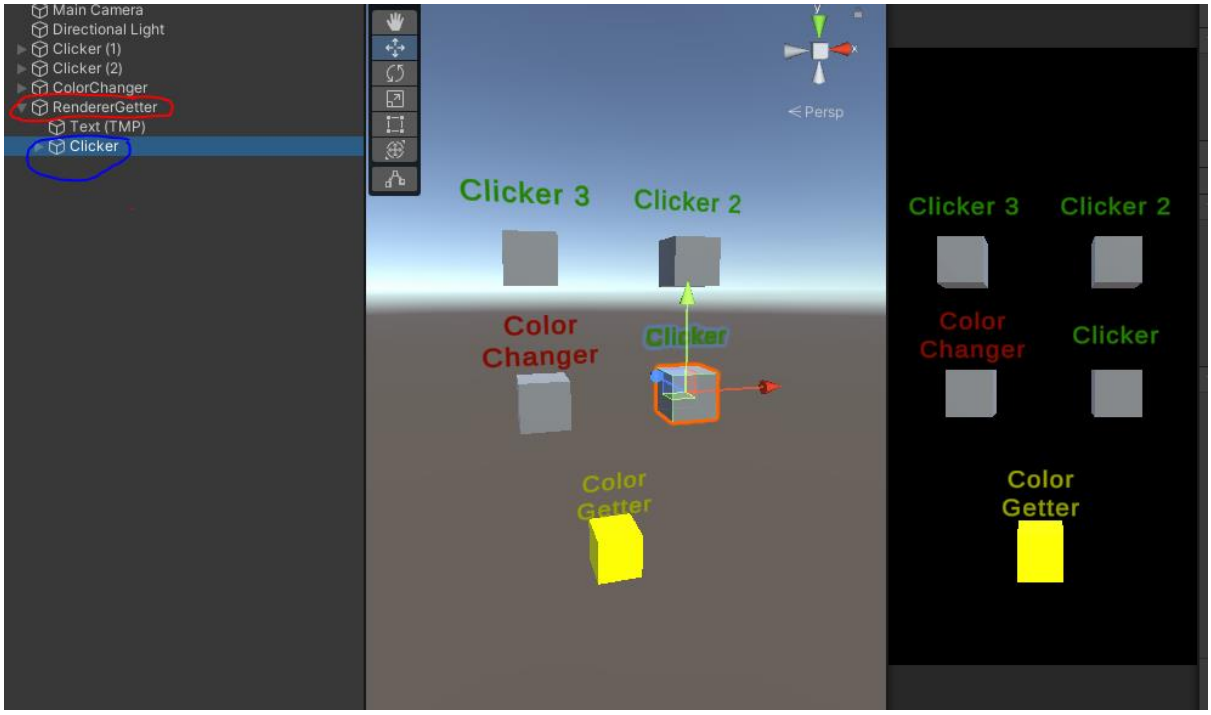


If ticked, when both CubeColliderClicker and the CubeColliderClicker1 are called, the parameter will turn both Already Called off rather than waiting for another command to be invoked for them to be turned off.

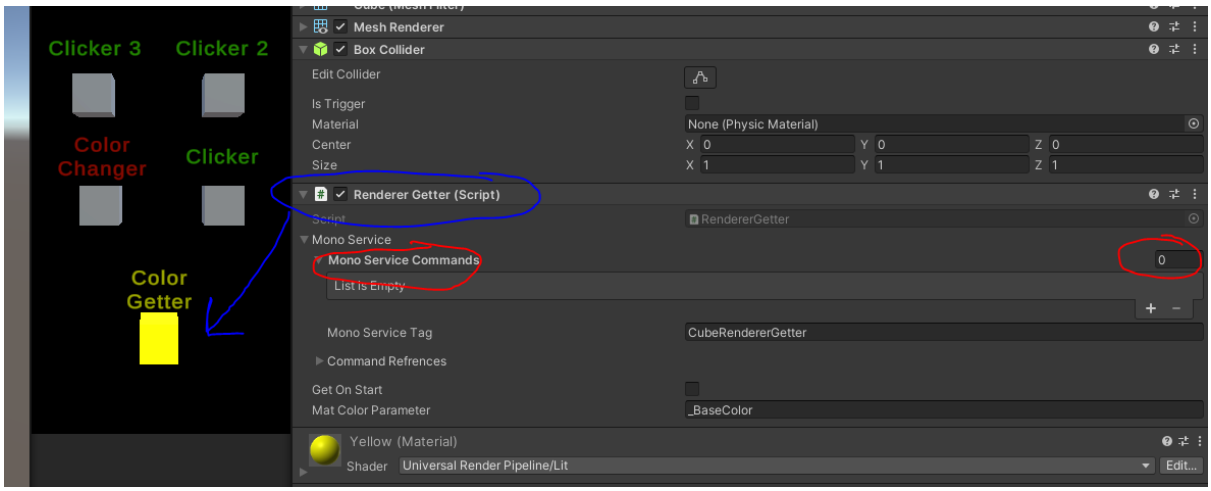
You can check the result here: https://streamable.com/pc2cin

The “Mono-service is a fam member” parameter is when you want the component to only receive commands from children or parents in the hierarchy, keeping the renderer getter and the clicker example:

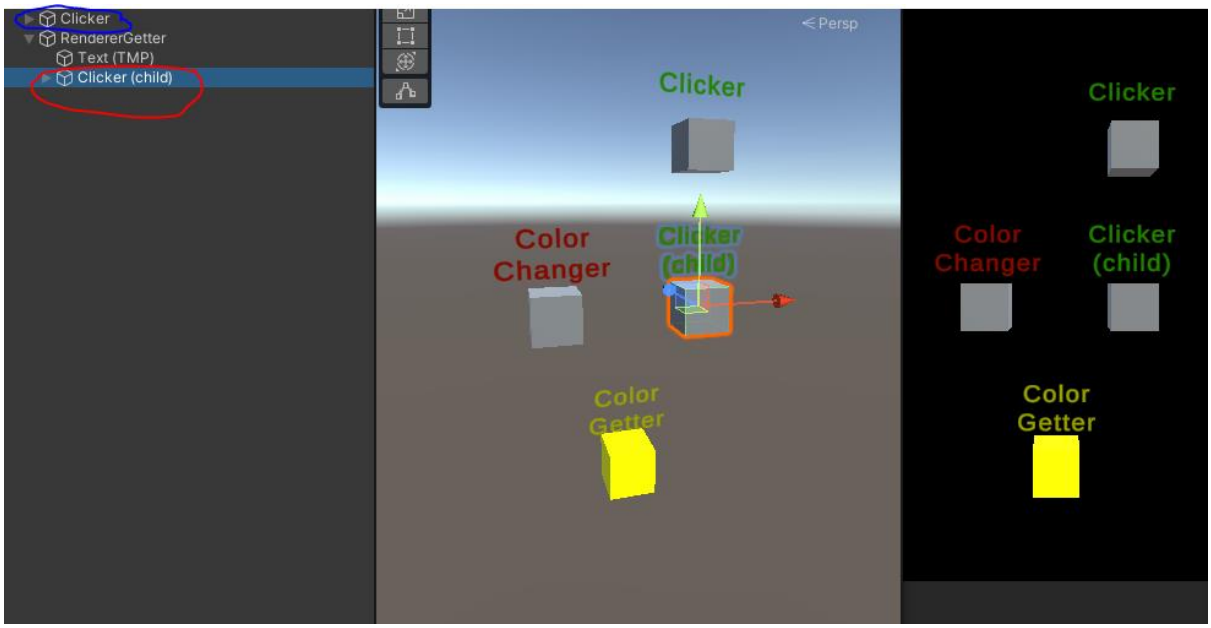
Let’s say if the clicker1 was a child on the renderer getter like this:



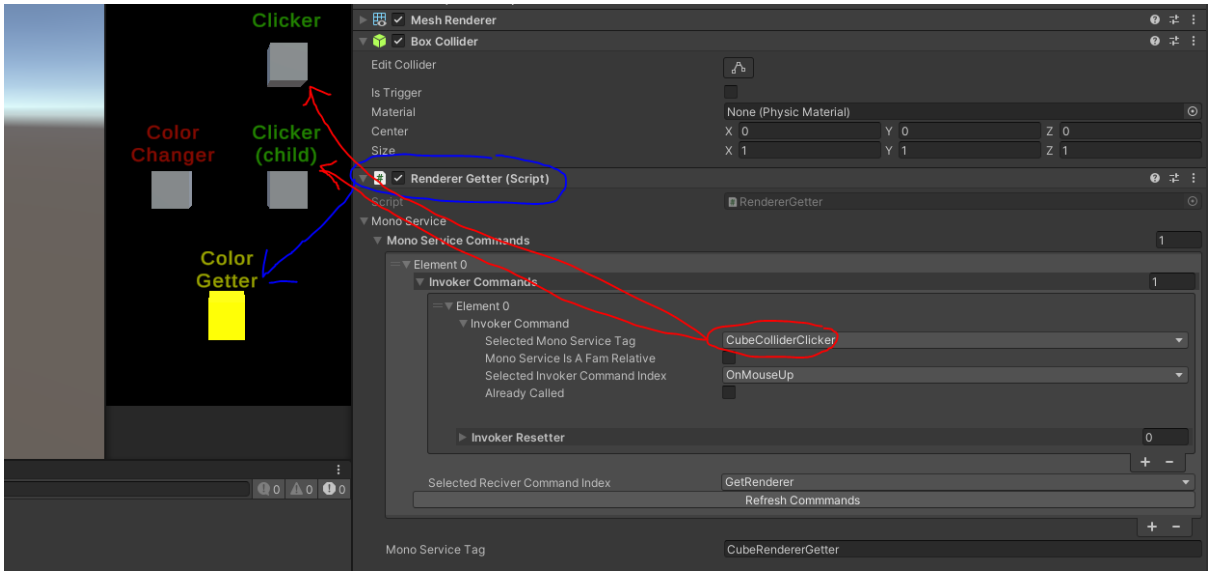
Let’s also clean the commands from the renderer getter to avoid confusion:



I have removed both of clicker2 and clicker3 and added an extra clicker1 but it would not be a child of the colour getter:



Now, let's clean the commands in the renderer getter component and add a new command that receives commands from clicker1:



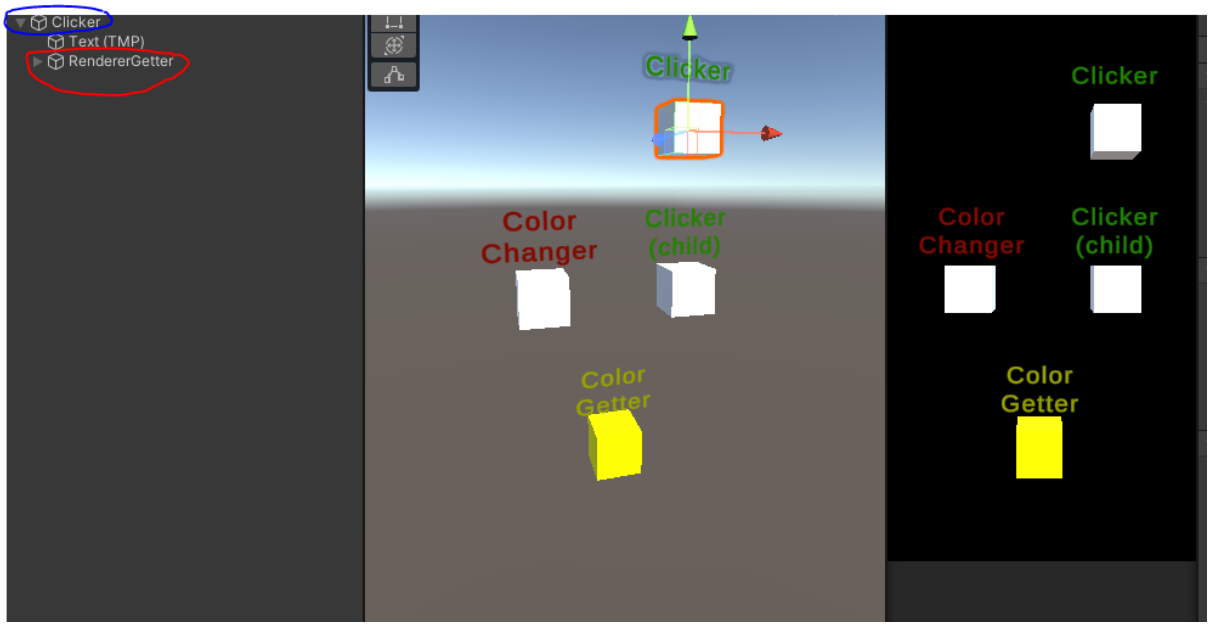
Without ticking the "Mono-service is a fam relative" on, the renderer getter will receive commands from both clickers which is not what we want.

If the parameter is checked, then the renderer getter component will only receive commands from the child clicker:

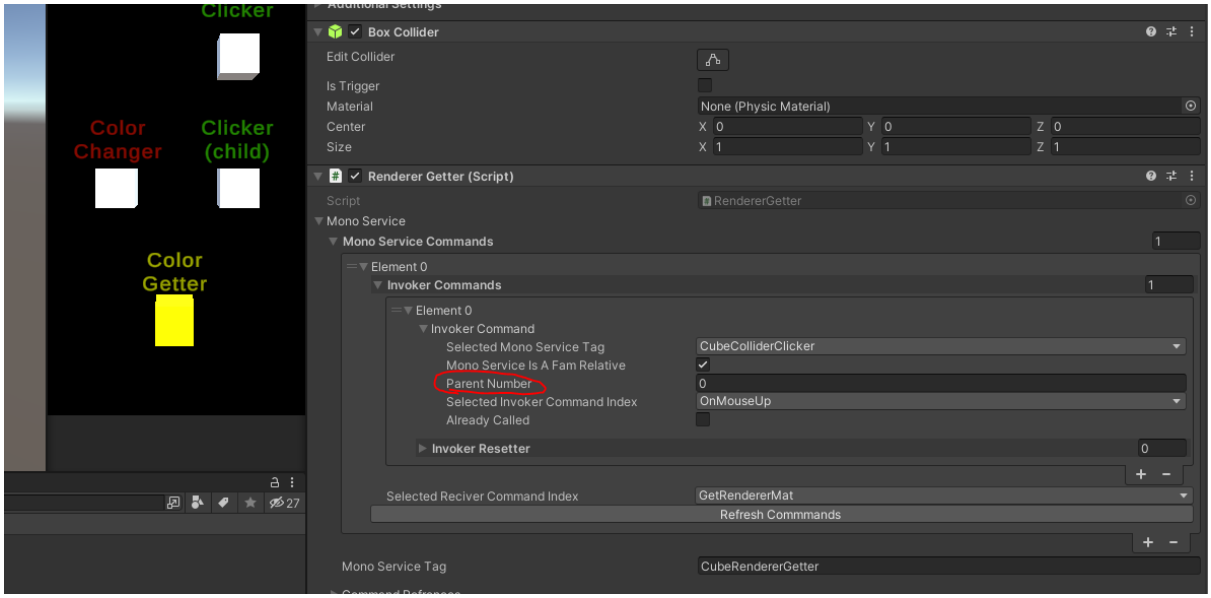
Note: I have added some clicking effects when the cube is being clicked, it will change its own colour to green for better visualisation.

You can check the result here: https://streamable.com/mqiywy

Now what if we had the opposite parenting where the renderer getter is the child, and the clicker is the parent like this:

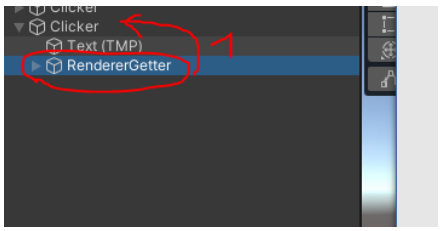


When you tick the “Mono-service is a fam relative”, new parameter will appear which is “parent number”:

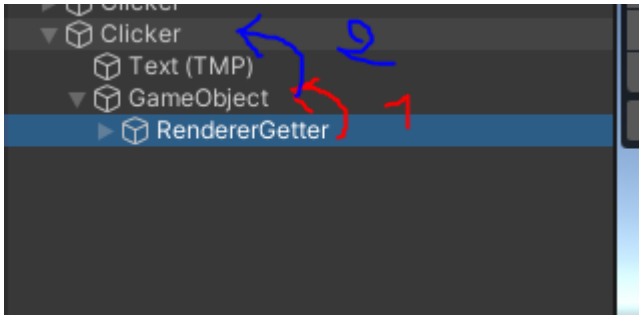


If the component is the parent of the invoker command, like in the first case where the renderer getter was the parent of the clicker1, then the parent number will stay at 0, but if the component is a child of the invoker command, then you count how many parent levels is there:

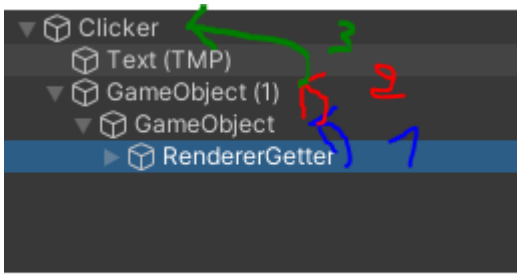
1 parent level:



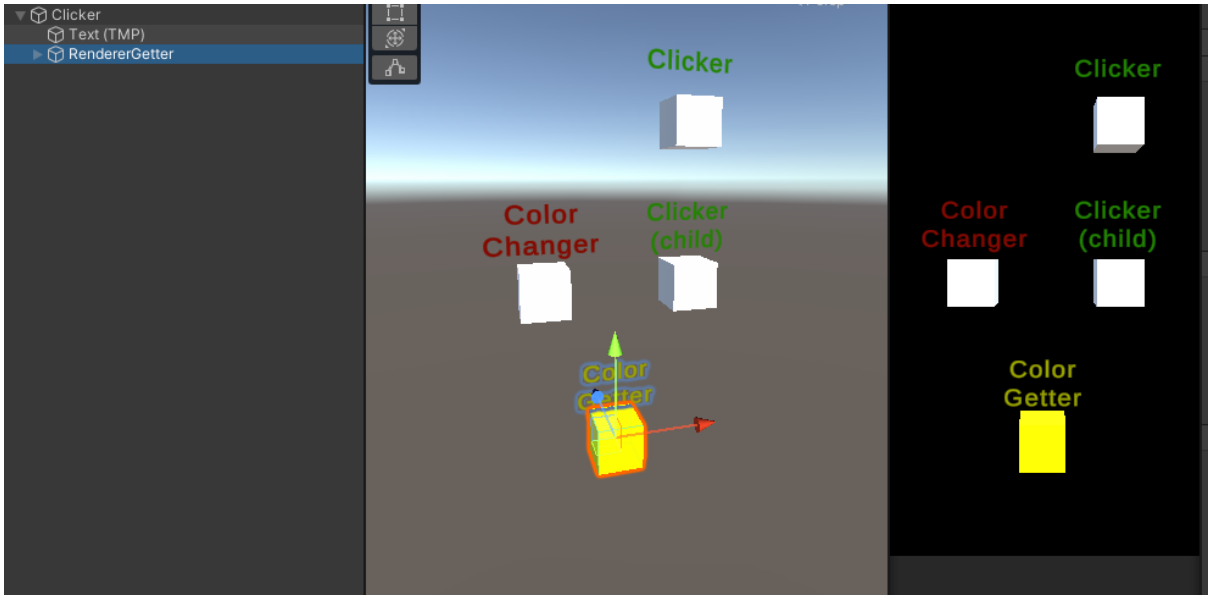
2 parent levels:



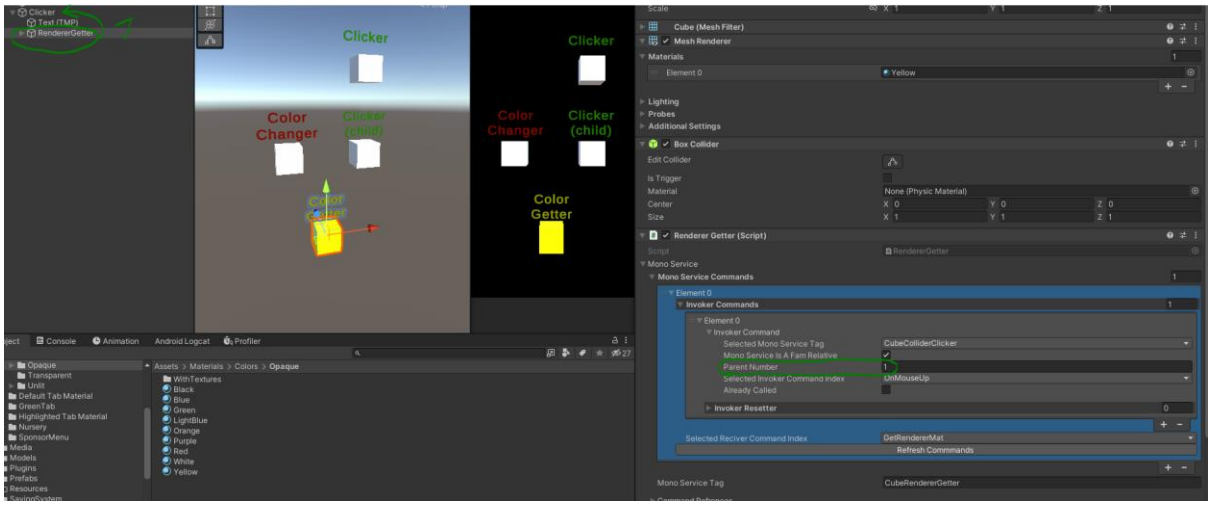
3 parent levels:



To keep things simple, we are only going to use the 1 parent example:



Now, to make this work, the parent number must be 1 because there is only one parent level:



You can check the result here: https://streamable.com/a52y7c