Courtesy of Color Documentation

Austin Foulks

Last Update:

July 3, 2023

V 1.0.0

Table of Contents

Purpose

Welcome to the Courtesy of Color tool! If you are reading this, you must want to add a little more complexity to your game's systems via color. That was the goal, anyway. I've seen many games, like the web browser game, Primary, and wondered if they could be taken a step further. What if the abilities and systems in a game like Primary weren't simply turned on or off based on the player's color? This tool presents another way of going about it.

This tool makes connected systems vary between 0% and 100% activity, and it compares colors to do that. But what the tool accomplishes is all up to user creativity, as the systems that Courtesy of Color hooks up to must be designed by other users. This tool will require some setup on the user's part, as well as their own coded systems to connect to Courtesy of Color, but there are so many more directions the system can go in than I can think of.

As far as my testing has gone, this tool only works in 2D projects.

CC Scripts

*For clarity, all exposed variables in Unity's inspector have a tooltip. Hover the mouse over the name to make the tooltip appear.

Read this section if you want a rundown over what each script of the CC tool manages, otherwise, feel free to skip down to the How To Use section.

Marked Position

This script is all about getting the color of a 2D object by using a raycast, going through all collisions, picking a chosen collision, using the object attached to the collider, getting the RGB of the object through a pixel on its texture or the primary color of the material, and converting that RGB value to HSB, which is more appropriate given the way humans perceive color.

The ray's origin and the code that gets the color from the ray are constantly updated, so options have been provided to disable this in the inspector.

Color Controller

This script is responsible for using designer set values (a target HSB, an HSB range, and an HSB influence, which sets the color that will output a one, how active the controller is for colors similar to the target HSB, and how much control the hue, saturation, and brightness have on the output), comparing the hue, saturation, and brightness individually between the target color and the color gotten by the Marked Position, compressing the difference to a value between zero and one using the HSB ranges, combining those values down to one output still between zero and one using the HSB influences, and passing that output to other functions connected via Unity Events.

This script is constantly updating, so options have been provided to disable updating in the inspector.

Controller Output

This script is an empty class for Unity Events that takes in a float, which allows designers to hook up their programmer's systems to the Color Controller and get the value that the Color Controller is outputting.

How To Use

*If these instructions are hard to follow, there is also a Youtube tutorial in the Links section.

Add Marked Position Component

Pick an existing object in your scene where you'd want to get the color from (you can get the color of that object or the color behind said object) or create a new prefab if you intend on using multiple Marked Positions. Add the script "MarkedPosition" to your object. The component uses the transform of the object it is attached to to fire the ray from, so keeping the component in its own prefab and adding that prefab to another object, like the player, allows for more precise positioning.

In the inspector you'll see a field called Object Reference. You can leave this blank if you were able to attach the component to your desired object directly via the two proposed methods above. If you cannot, then you can bring that desired object to your component instead.

You'll have to select the Render Method depending on whether the color comes from a texture or a material. If Collision To Get is set to one, it'll use the object of the first collision. Increase this value if your Marked Position is being blocked by other assets with colliders, or set the Marked Position to always get the last collision. Keep in mind, if Collision To Get is higher than the number of collisions in the raycast, it will default to the first collision, but you can make the default the last collision. Set the other fields in the inspector as you wish, but for a top down 2D game, you'll likely want the ray direction to be forward, and for a sidescroller, you'll likely want the ray direction to be down.

Everything will work correctly whether the component is attached to the parent object or a child object.

Format Your Functions

Pick an object whose properties you'd want to fluctuate in some way. This could even be the same object you picked to have the Marked Position attached to. You'll have to set up a class for this object yourself, such as getting a reference and storing the data you want to fluctuate in a new variable. If you do not know how to do this, I'm afraid you'll have to find outside resources on how to code in Unity.

Once it is set up, create a new function that takes in a float. This is the function that will be called by the Color Controller's Unity Event so the parameter will always be between 0 and 1. You can do whatever you want with it, but this will let you set up a system that can be not active, partially active, or fully active. Because of the limitless things you could do with the output, there are many ways to set up the function, such as this function to change a character's speed:

```
public void ChangeSpeed(float colorControllerOutput)
{
    currentSpeedUp = speedBoost * colorControllerOutput;
    moveSpeed = normalMoveSpeed + currentSpeedUp;
}
```

Add Color Controller Component

Create a new prefab and add the script "ColorController" to the object. This will make it easier to reuse and create new Color Controllers. For the Marked Position field, give it the object/prefab whose position you wanted to use to get color from above. Use the color picker to select your target color, set the range values based on how many gradients around the target color you want giving a partial output (the range is ±), and set the influence values to change how

much of an impact hue, saturation, and brightness have on the final output (all three influences at 100 means the hue/saturation/brightness all have 33% control of the output).

Now click the plus under Controller Output to add an element to the Unity Event. Drag the object you set up to have a fluctuating system above into the field. On the right, find the component in the dropdown that contains the function made for this Unity Event and select that function. Make sure to choose the one at the top under “Dynamic Float” or else it will not function properly.

If you have any other properties or systems you want affected by this Color Controller, add them to the Unity Event. Chances are you’ll want different properties of the same object changing, and it’ll be more interesting connecting each of those properties to a different color controller. You could make property X get larger the closer the Marked Position is to blue while property Y gets larger the closer the Marked Position is to white. You’ll need a Color Controller with blue as its target color and one with white as its target color. Each Color Controller you have active will need to be in the scene.

I would not recommend calling the same function from different Color Controllers, but don’t let me stop you. Who knows how chaotic the output will be.

Links

Youtube Tutorial:

https://youtu.be/IXz_w8nCKak