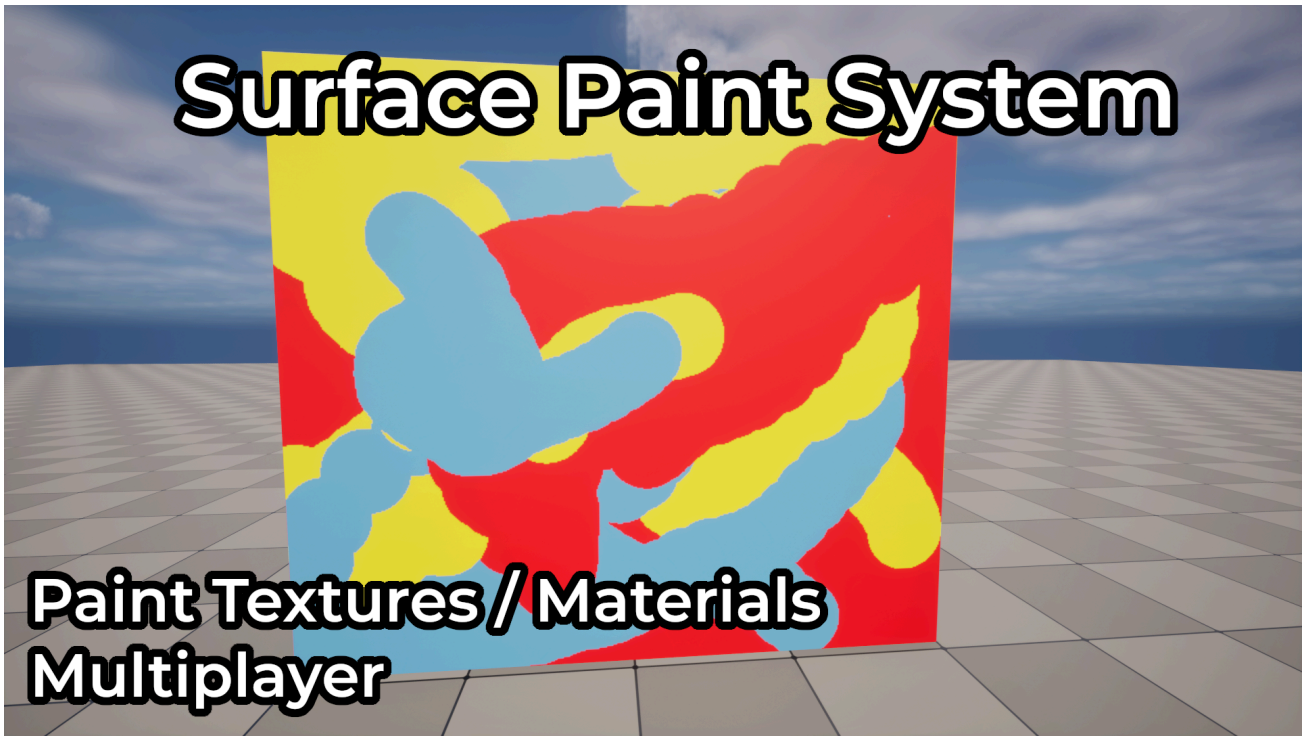
Surface Paint System Plugin

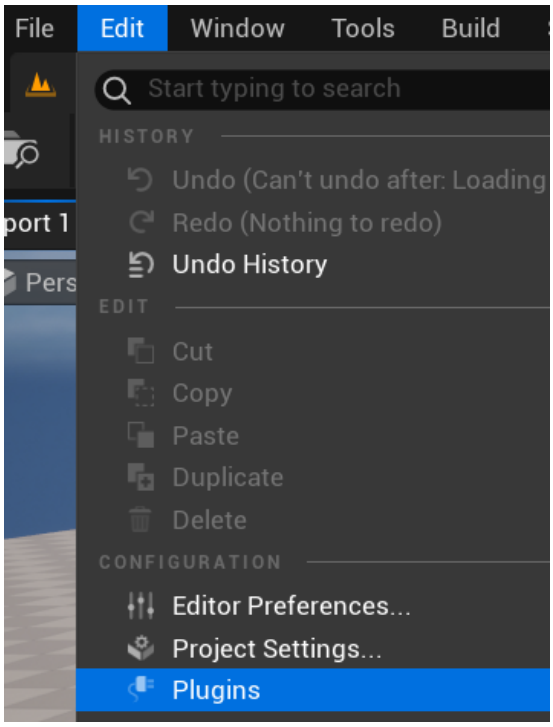
The Surface Paint System is a fully replicated framework for dynamically painting textures or materials onto surfaces at runtime. Apply paint through UV collision, track how much of a surface is covered, and customize everything from paint size to texture. Ideal for marking objectives, decorating environments, or applying elements like sauce or paint in cooking and simulation games.

Help: louisgirard.games@gmail.com



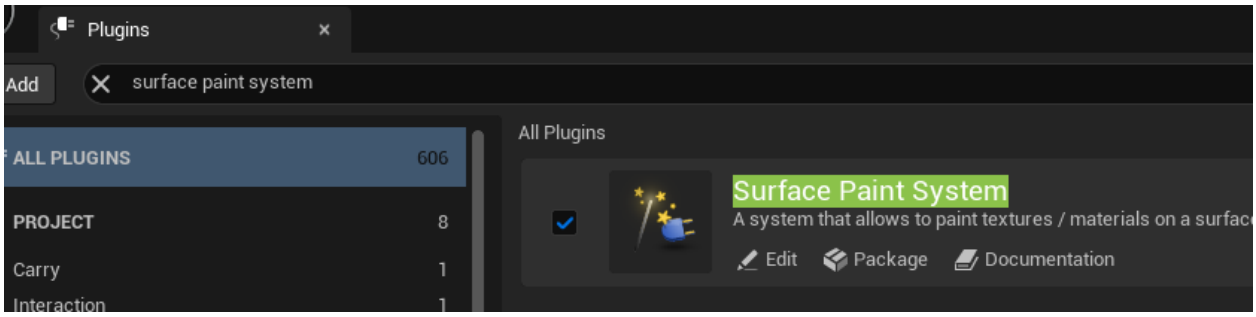
Activating the Plugin

Go to plugins



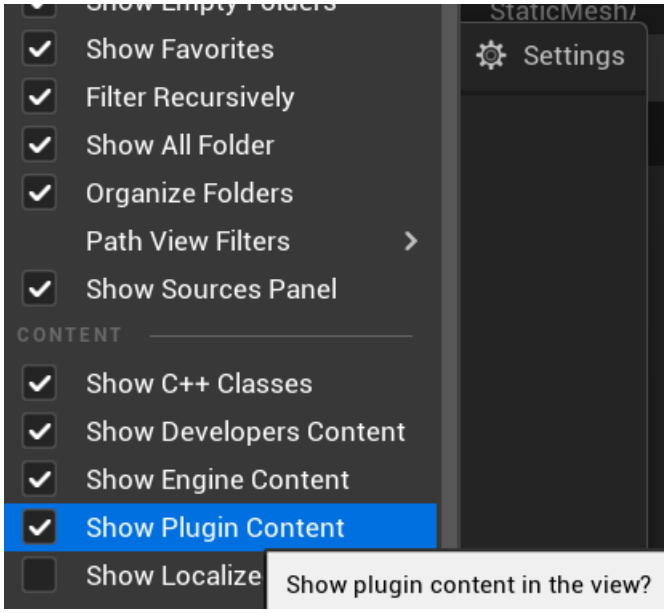
Enable the Surface Paint System plugin

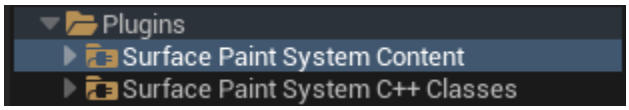
Search for the Surface Paint System and tick the box to enable it.



Sample Content

Once the plugin is enabled you can enable the Plugin Content by going in the Content Browser and tick “Show Plugin Content”.



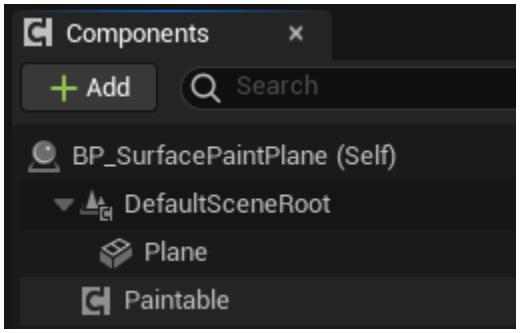


You can launch the **L_SurfacePaintSystem map** and start testing the paint functionalities, or follow the documentation on how to set up your paintables and player.

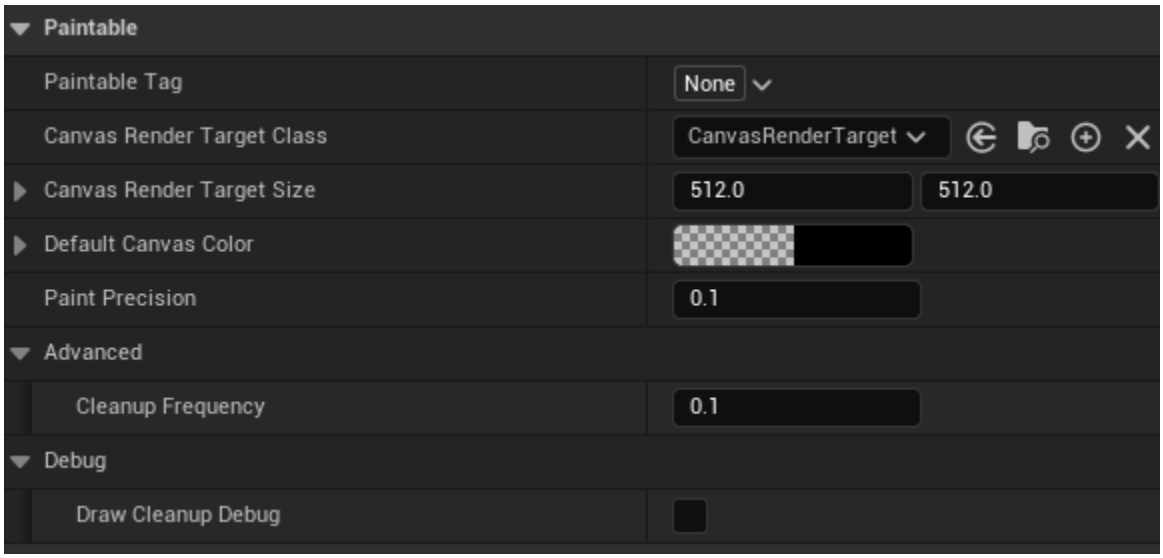
Paintable Component

This is the component that will contain all the information about what is painted on a surface and where. You can add it to any actor that you want to paint.

Let's take a look at the one added on the BP_SurfacePaintPlane example.



Overview



Paintable Tag

You can use this Gameplay Tag to identify the Paintable Component in case you have multiple ones on the same actor.

Canvas Render Target

The Canvas Render Target 2D is the object used to hold the drawn paint textures or materials. It is like a 2D texture.

You can choose the **Canvas Render Target Class**, the default CanvasRenderTarget2D should be enough but you can choose your own if you have specific functionalities.

You can also change the **Canvas Render Target Size,** which is basically the resolution of the canvas. The higher, the more detailed it will be, but the more costly for the system.

Paint Precision

This value represents the precision you want when detecting the Paint Ratio of some paint. The higher, the more costly it will be.
Internally what this does is building a Grid the size of the CanvasRenderTarget * PaintPrecision. So a value of one will be a perfect match with the CanvasRenderTarget but will have more cost when painting onto the component.

Cleanup Frequency

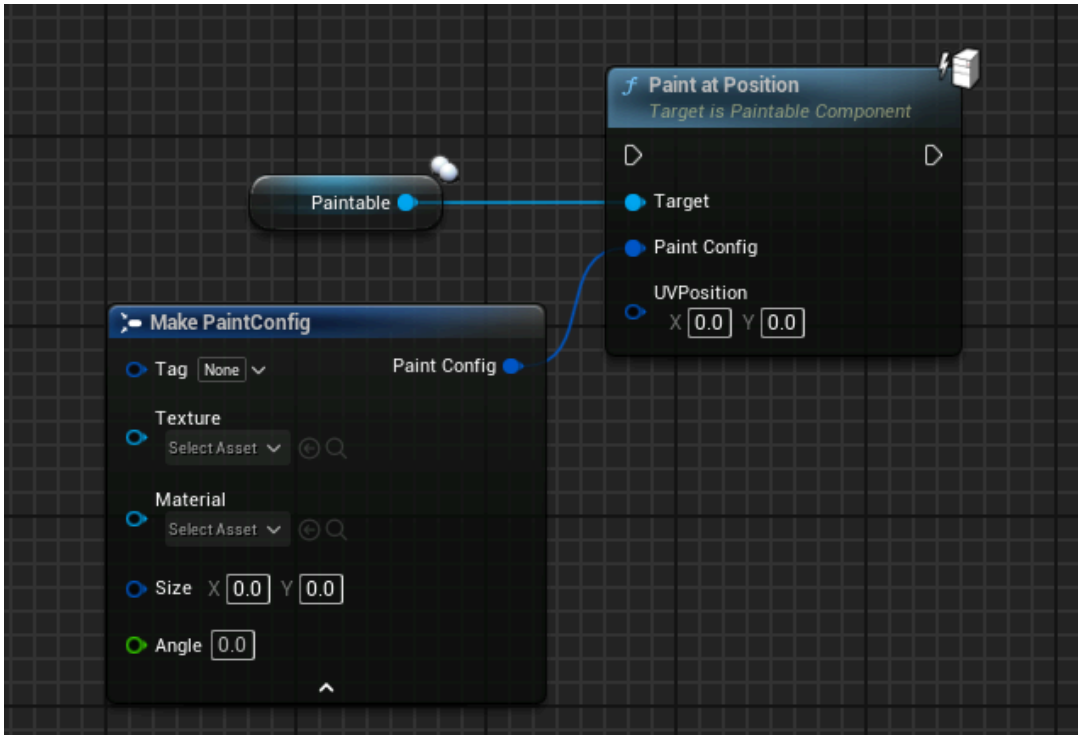
This value is the frequency at which the system will clean up unused data, removing old paint data which is now overlapped by new paint. You should not need to change this by default.

Draw Debug

You can enable some debugs for Cleanup. This will draw boxes on the canvas to display which paint is being cleaned up.

Paint

You can easily paint by passing a **Paint Config** and a **UVPosition**.



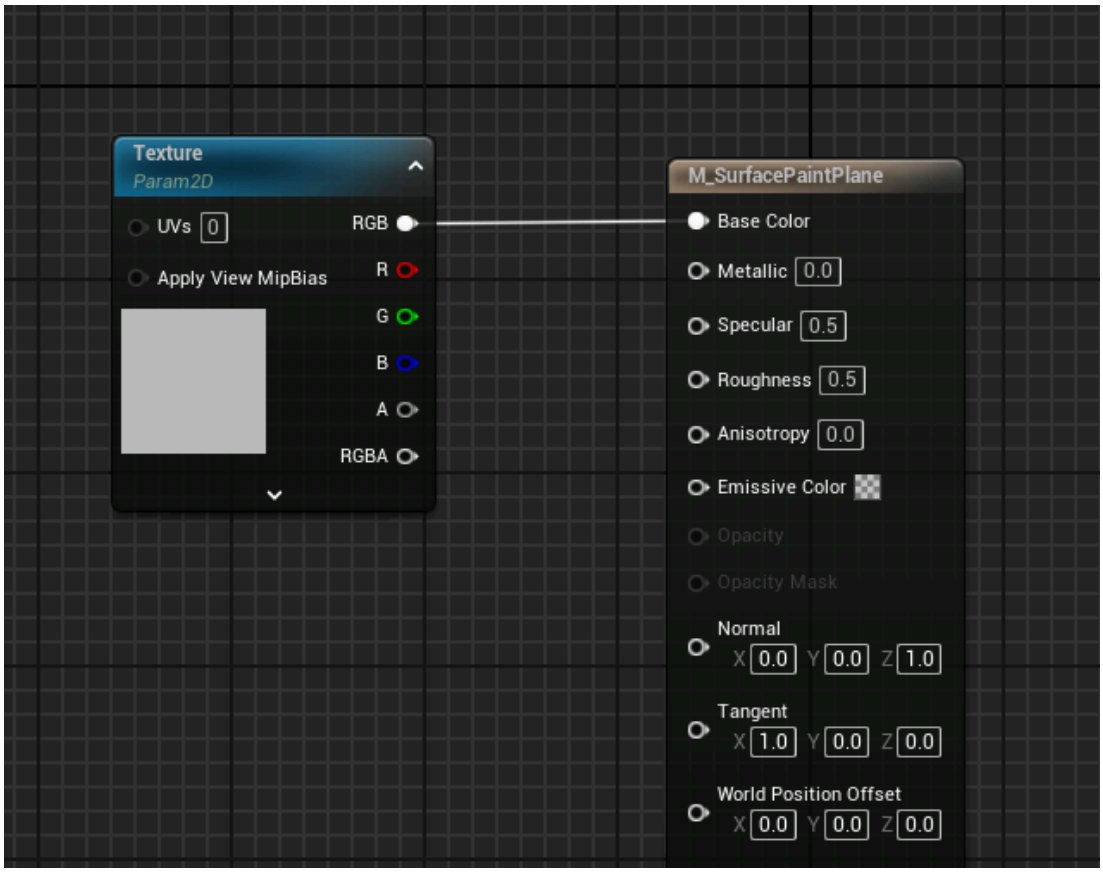
The **Paint Config** allows you to define a Texture or Material to paint, with a size and at an angle. You can also specify a Tag to easily identify the paint.

The **UV Position** is a position between (0, 0) and (1, 1) where the paint will be applied on the canvas.

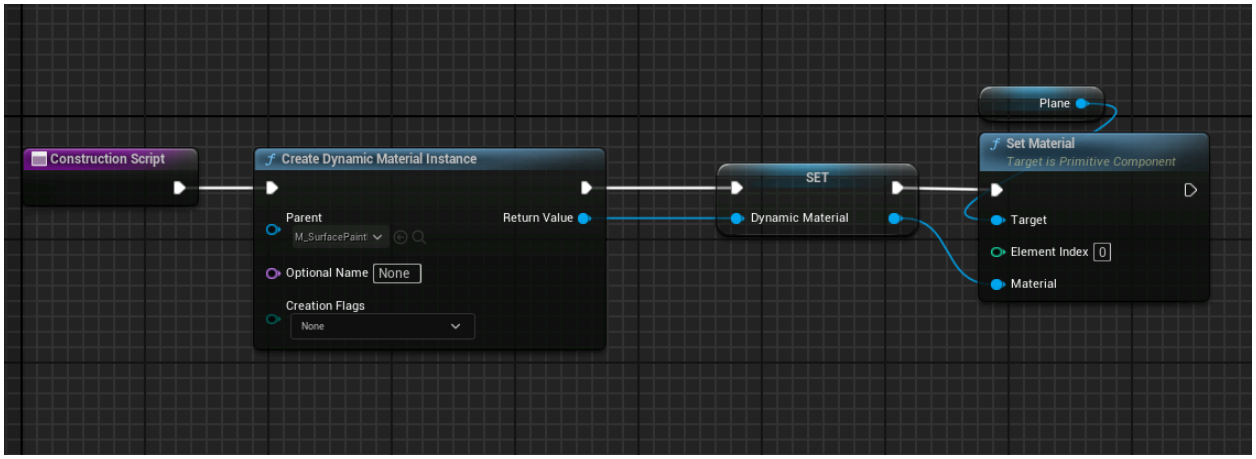
Display the Paint

You can directly display the paint on your mesh using the Canvas Render Target 2D.

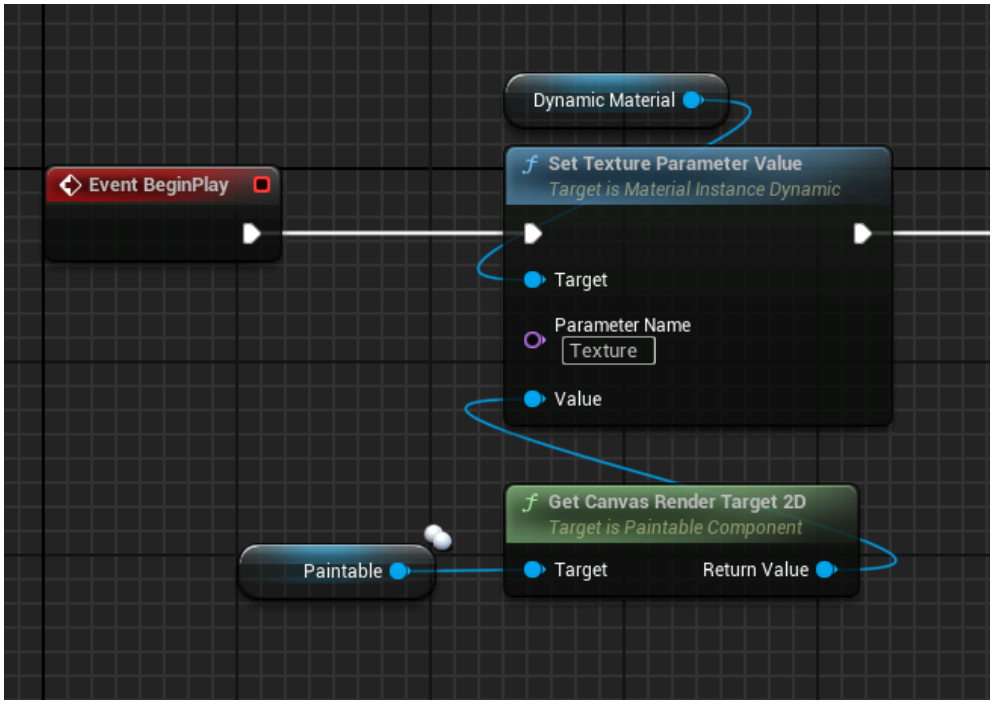
In our example we have the M_SurfacePaintPlane which is a simple Material with a Texture2D set as a parameter.



In the BP_SurfacePaintPlane we create a dynamic instance of this material and apply it to the mesh.



The only thing left is to apply the Canvas Render Target 2D as the texture to display for the material.

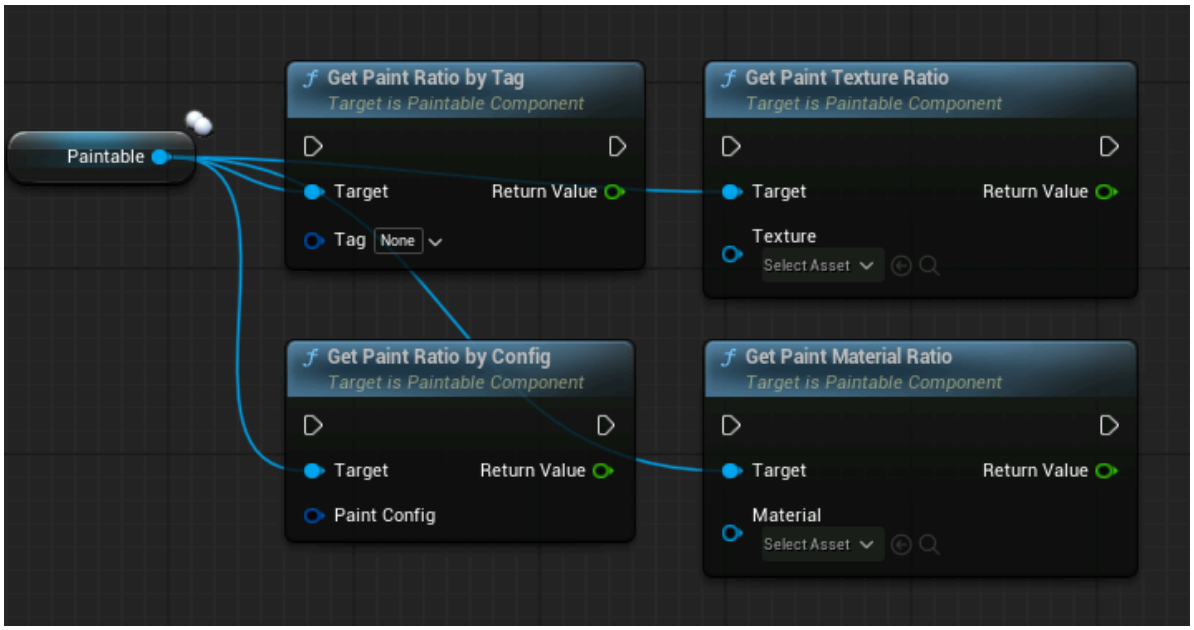


This will make it so that our mesh will display exactly the content of the canvas.

It is only of the usage, but you could easily use this canvas as an alpha texture, and use it in other materials.

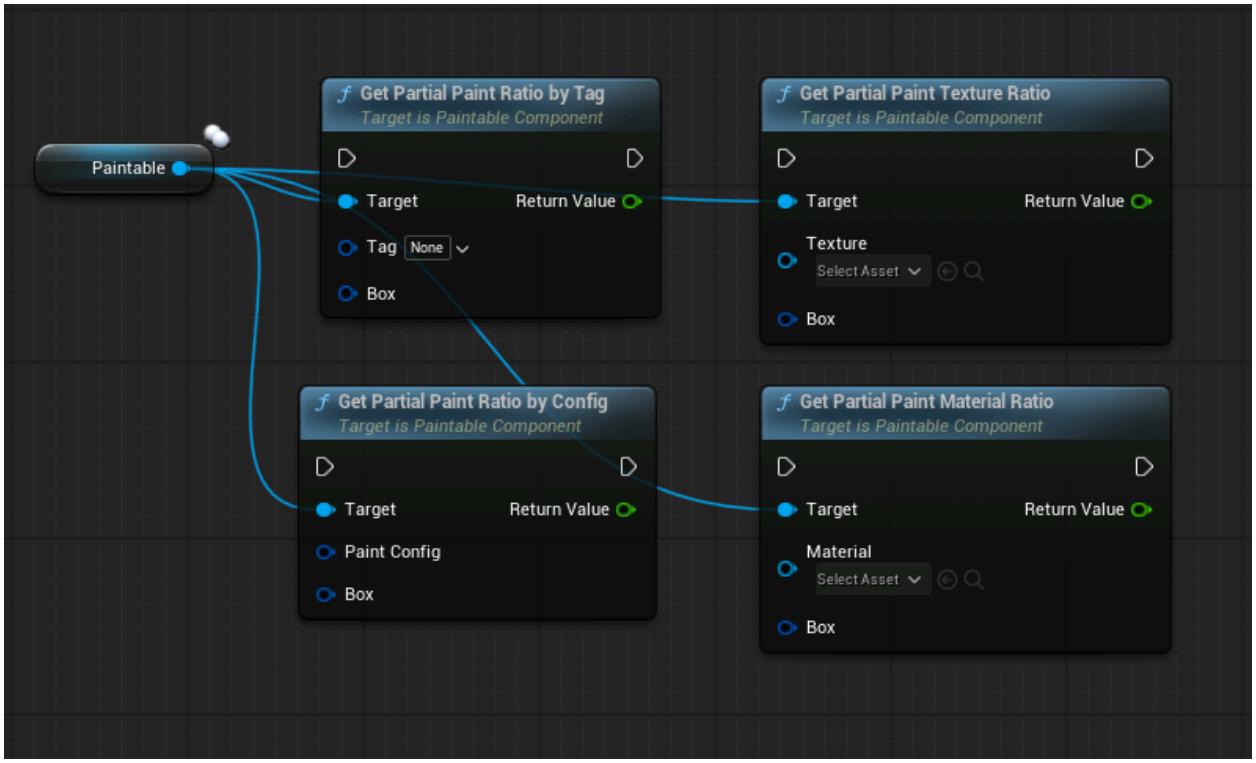
Getting Paint Ratio

If you want you can get the ratio of a certain paint, accessing it by tag, texture, material or config.



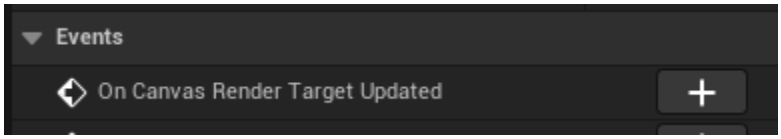
These functions will return the ratio (between 0 and 1) of paint visible on the canvas.

If you only want to have the ratio on a portion of the canvas, you can use these functions instead.



They will return a ratio (between 0 and 1) of paint visible within a portion of the canvas, defined by a box. *The box size should be between (0, 0) and the Canvas Render Target Size.*

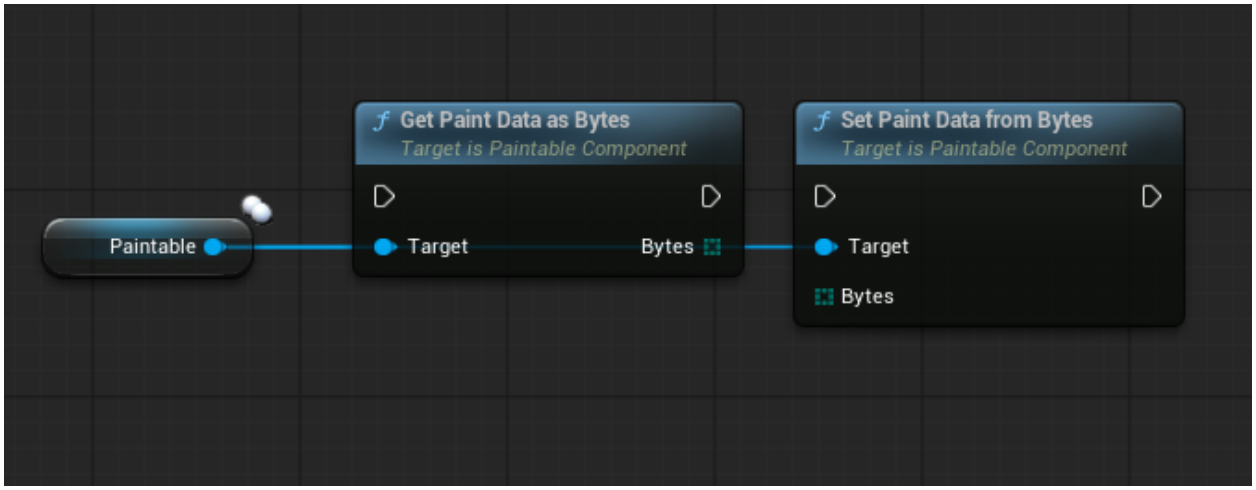
Events



You can bind to the **On Canvas Render Target Updated** to listen when something has been drawn to the canvas.

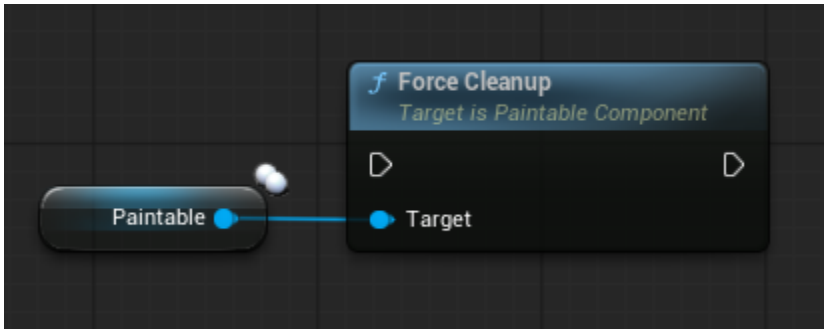
Save

If you use a custom Save System and you need to get the data to save and load your paint component you can use these functions.



This will allow you to get the current data as Bytes and save it. And also to allow loading data from Bytes and set it in the component.

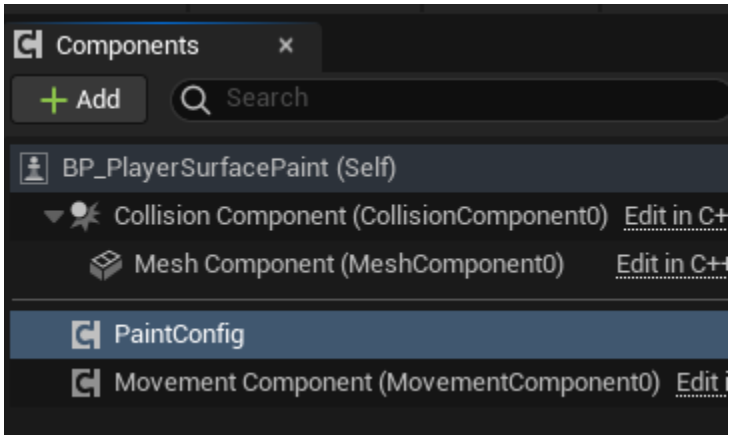
Before the GetPaintDataAsBytes you should call this function which will make sure there is no unnecessary data in the system.



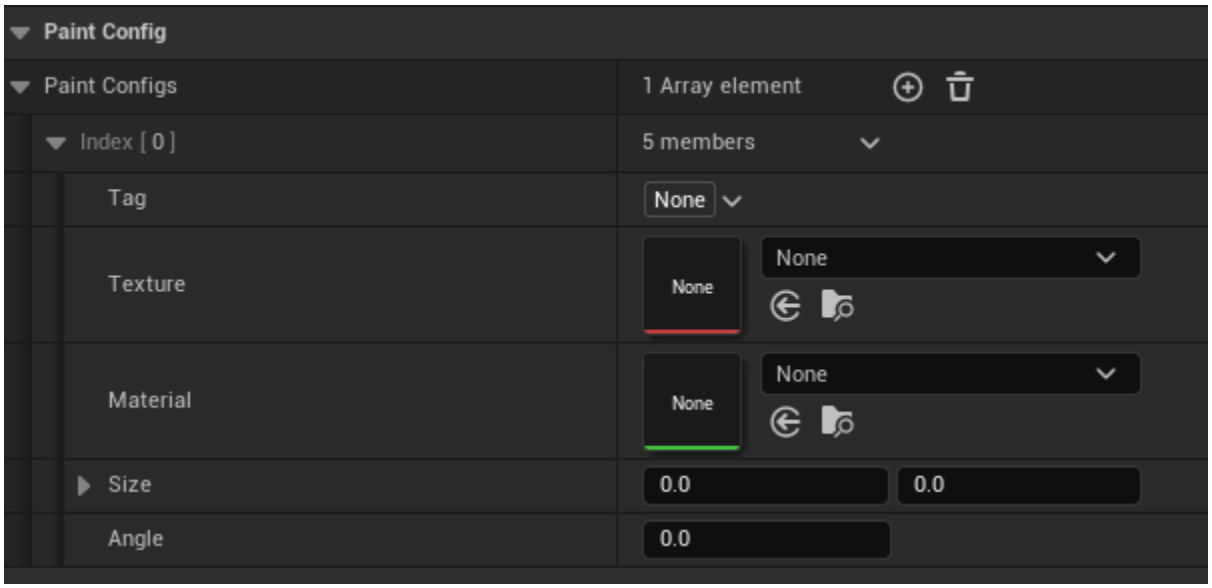
If you already use the [WorldSaveSystem] plugin then you don't have anything else to do than adding the WorldSaveComponent and everything will work out of the box.

Paint Config Component

This component can hold different **Paint Configs**, allowing you to define at one place the type of paint (texture, material) with its size and angle. You will usually have this component on your player or other actor that can have the ability to paint.

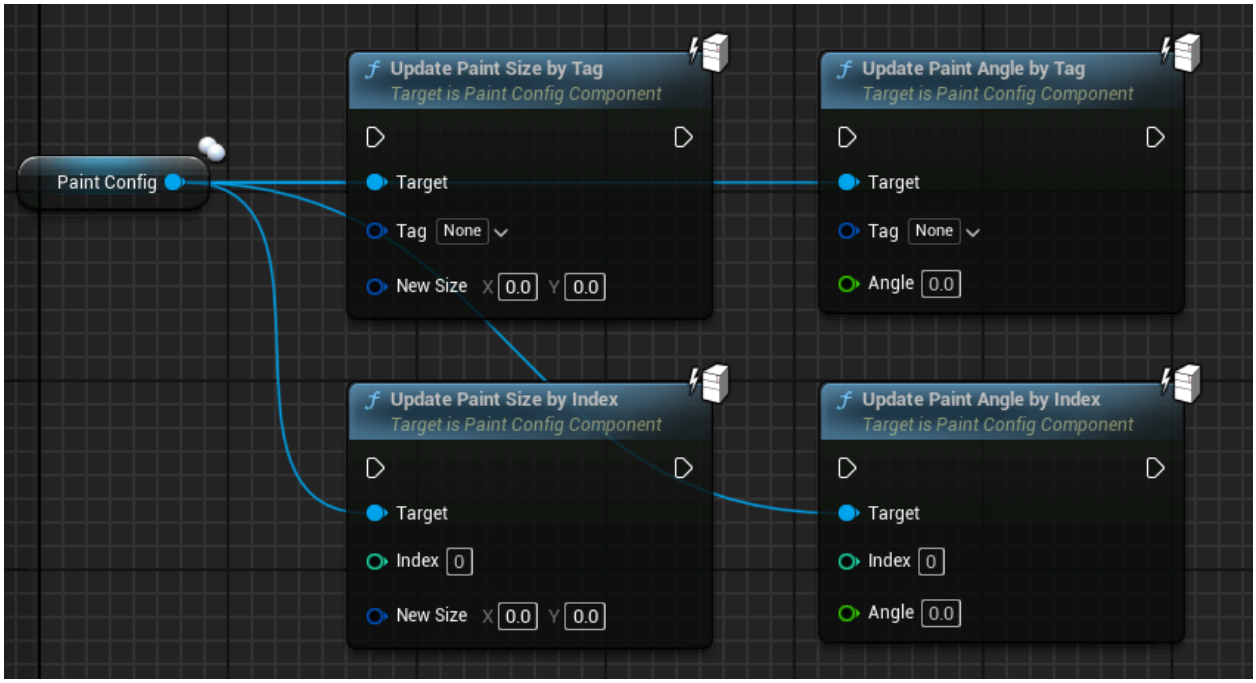


Overview



You can configure an array of **Paint Configs.** These will be the different paints that your actor will be able to apply. These are the same configs that are used to paint a Paintable Component.

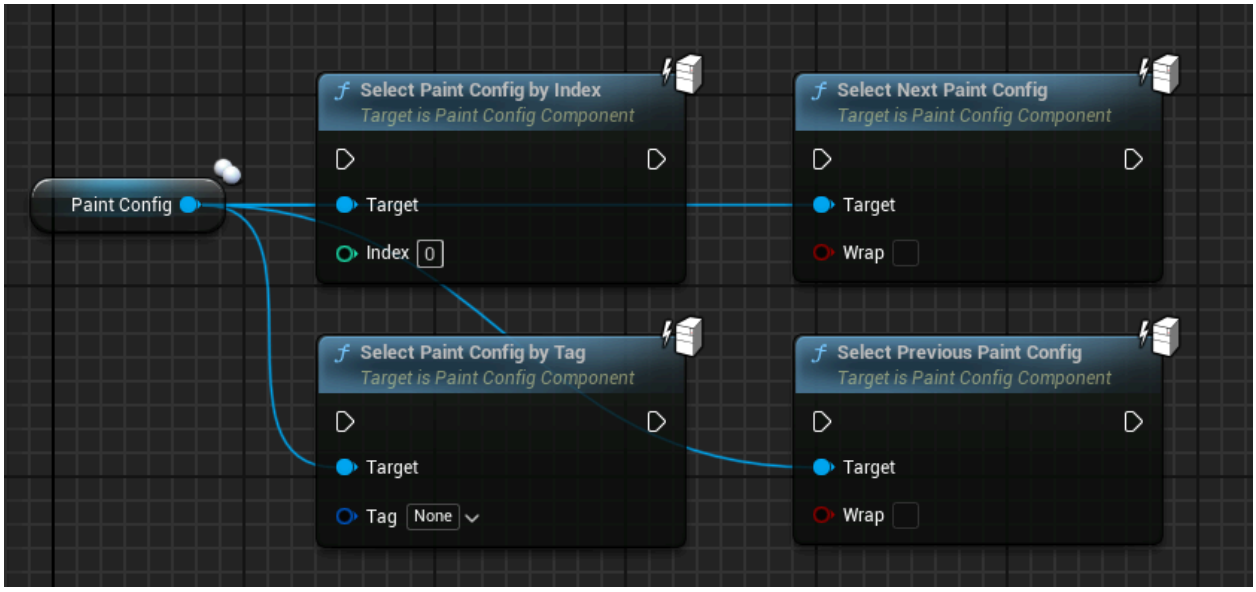
Update Paint Configs



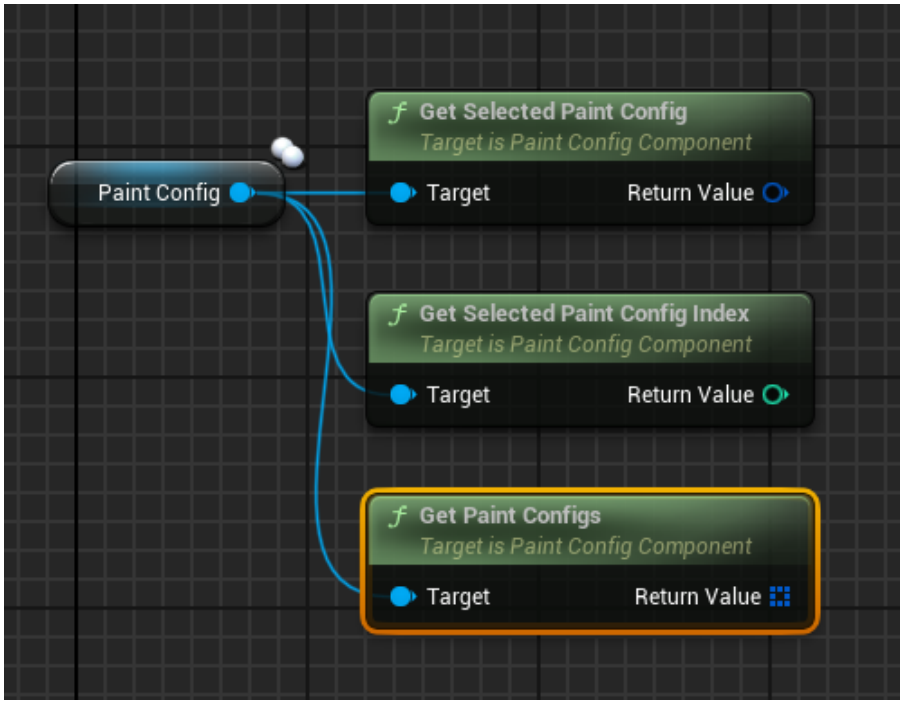
You can dynamically update the paint configs size or angle using their index or their tag.

Select a Paint Config

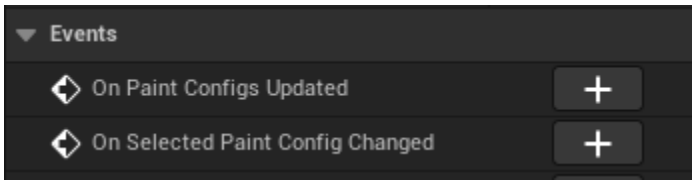
In some cases where you have multiple configs you might want to only have one active at a time. You can use these functions to select a certain config by index or tag.



Then to get the currently select config you can use



Events

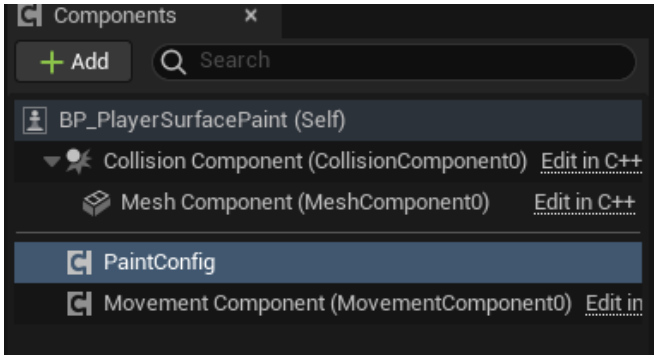


You can listen to these events for when the **Paint Configs are updated**, or when the **Selected Paint Config Changed.**

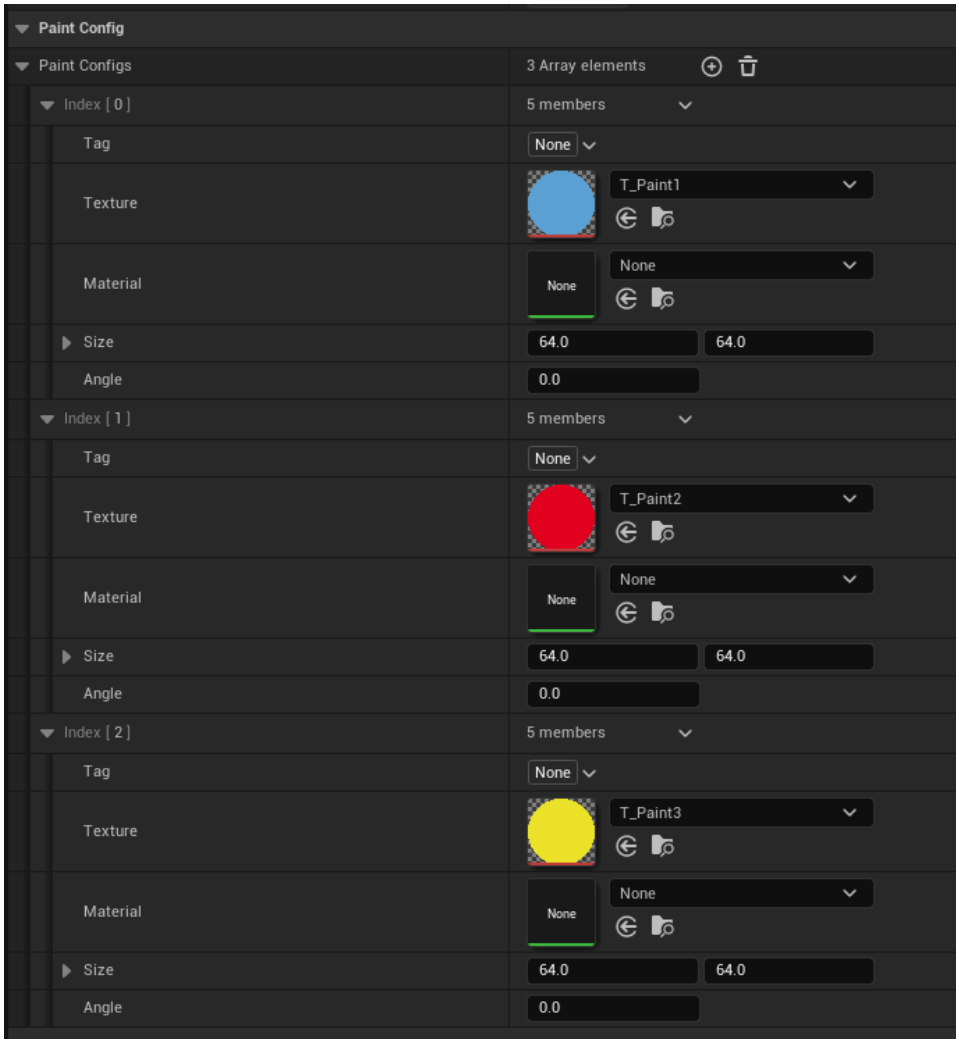
Player Example Setup

The set up for the player is here as an example and can be adapted to your project's needs.

Let's look at BP_PlayerSurfacePaint.



The player has a **Paint Config Component** with three different **Paint Configs**.



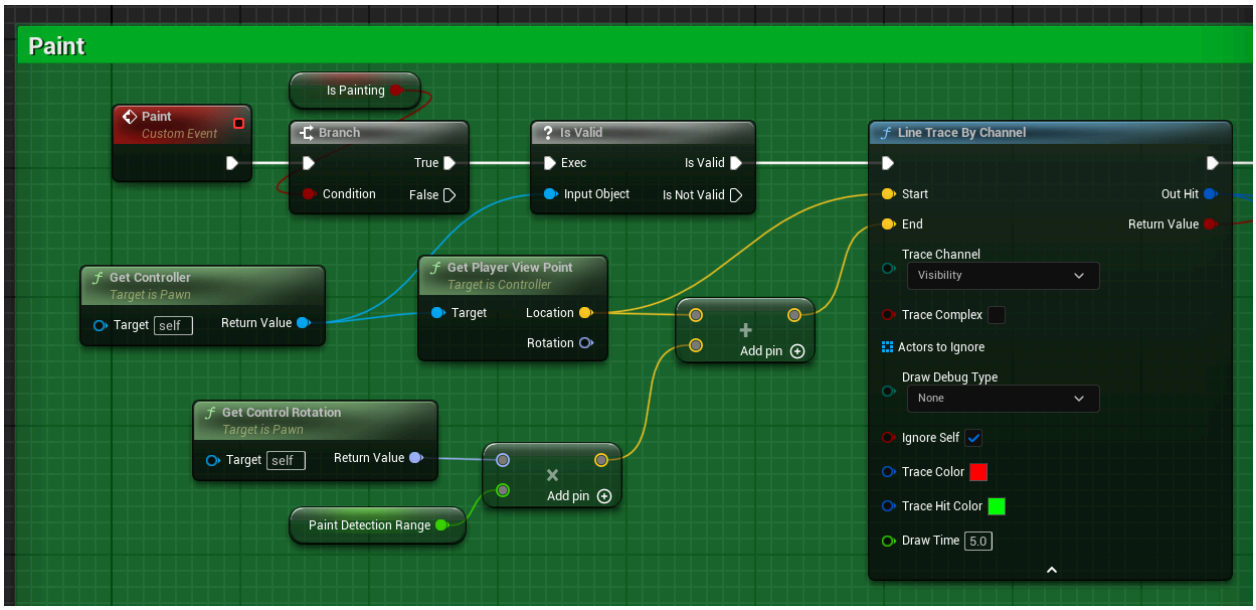
Inputs

The inputs are very simple.

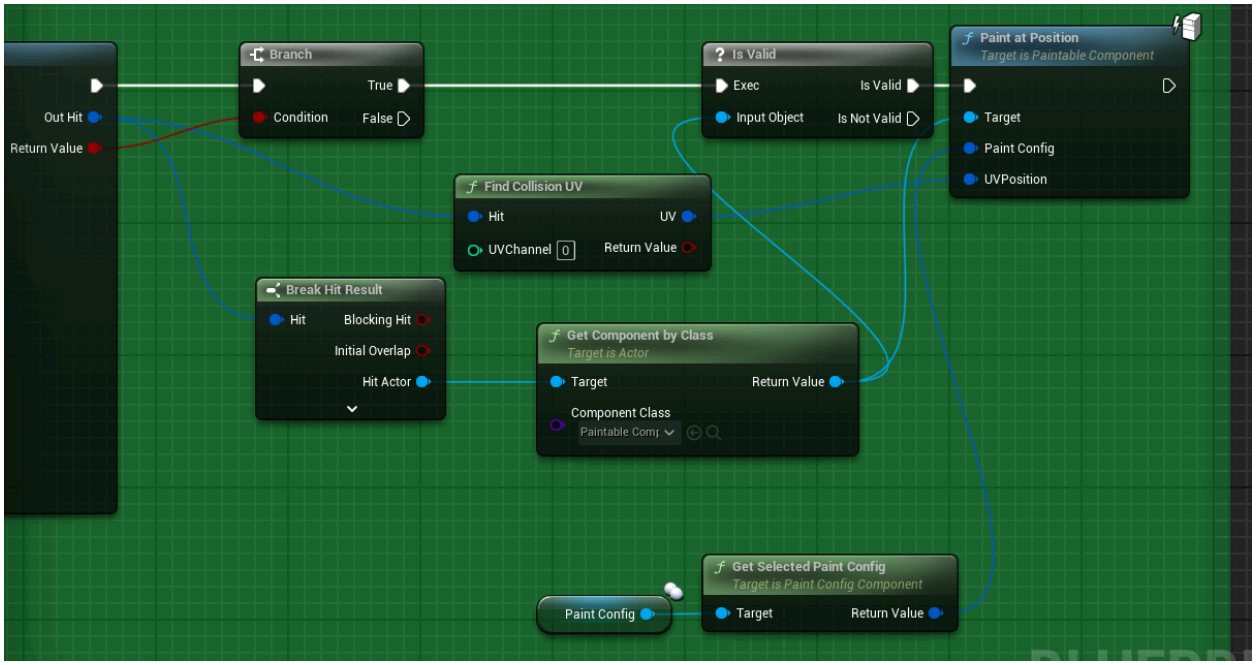
- Left Mouse Button: Set a **IsPainting** boolean with a Server RPC.
- Mouse Wheel Up and Down: Select Next/Previous Paint Config with a Server RPC.
- 1 and 2: Update the size of the selected Paint Config with a Server RPC.

Painting

The actual painting part is done on Tick.

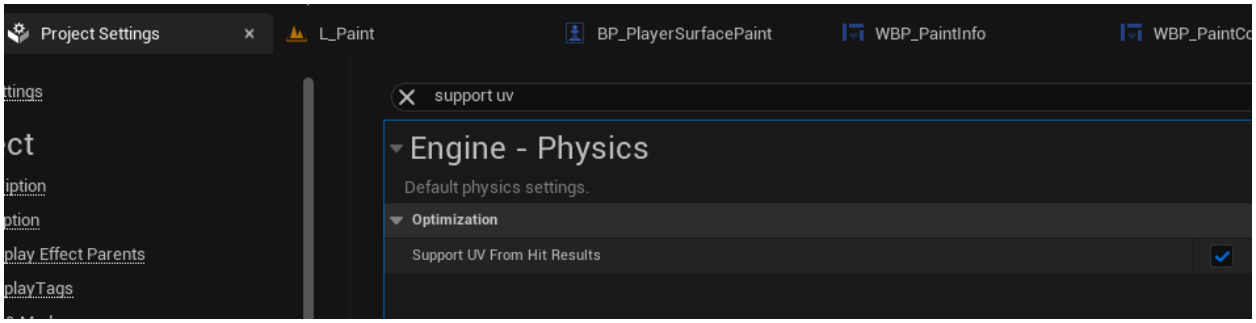


First, if **IsPainting** is true, we do a Line Trace By Channel starting from the player view point and within a range. You need to make sure that **Trace Complex** is ticked, since detection of UV with collision requires more precise collisions.



Then if there is a hit, we take the UV from the collision, and we paint the Paintable Component of the hit actor, at the UV position, using the selected paint config.

The most important part here is **Find Collision UV**. This will allow you to get the UV coordinates (0 - 1) on the hit actor, and apply the paint at the hit position. You have to enable it to access it, going into **Project Settings -> Support UV From Hit Results**



With all that you should have all the information needed to start using the plugin and customize it the way you want. If you feel there is some information missing or you don’t understand some part, feel free to contact me at louisgirard.games@gmail.com.