Introduction to Unreal Engine Blueprints for Beginners

By Chaven R Yenketswamy

Introduction

My first two tutorials covered creating and painting 3D objects for inclusion in your Unreal Project. In this tutorial we will look at how objects can be manipulated and interactions processed to add more variety to your game world. To achieve this we will use blueprints.

If you have a background in object orientated programming then blueprints are in a way very similar to classes that package behaviour and extended functionality. With blueprints we do the same only we don't work directly with coding but with a type of visual scripting very similar to block flow diagrams which illustrate the flow of steps and operations that need to be performed to achieve a specific goal. In Unreal Engine there are several predefined block or nodes which we use to process and build a certain type of behaviour in our game. All we do is chain together nodes referencing our objects and their properties. For example we may change the position coordinates of a ball to create a bouncing animation.

The simplest type of manipulations will therefore involve transforming our objects size, position and rotation in the world. The next stage is looking at how objects interact with each other by checking for overlaps of geometries or collisions. Fortunately with Unreal Engine all these tools are made available to us without writing a single line of code. Once you know how the tools work and how to perform common tasks the rest becomes repetitive and you can focus more on the creative element of game design then struggling with basic concepts.

Outputting a Message to the Screen

Let's see how standard programming tasks translate into Visual Scripts and Blueprints in Unreal Engine. Programmers are all familiar with the traditional "Hello World" type example where this sequence of characters are displayed on the screen. Let's see how something similar is replicated using Visual Scripts.

To start we need to create a variable to store the string "Hello World" followed by a command that outputs the string to the screen. In Visual C# console applications the following code will accomplish this task.

```
class Program
  {
     static void Main(string[] args)
     {
        string s = "Hello World";//Variable of type String
        Console.WriteLine(s);//Command to display our message to  the screen

     }
  }
```

So how do we create a string variable in Unreal Engine and output this to the game user interface? Clearly two things are required a string a placeholder or variable to hold the string content and a command to display the string. Another question one may ask is where is the visual blueprint logic inserted in Unreal Engine. To begin lets open Unreal Engine and create a new project.

Select the project tab that says blueprints and click on a Blank Project. We wouldn't be adding any Starter content so leave everything else at their default settings. Click Ok to create your new project. This opens up the default level of the Unreal Editor. A level is basically a container for all your scene objects including lights, cameras and actors.

Your Content Browser panel is like your Windows Explorer where you manage all your project assets such as Animations, Meshes, Materials and Blueprints etc. We will add a Blueprint class to our content Browser. So right click here and select Blueprint class under the Create Basic Asset category then select Actor which is the base class for objects that are created in our level/scene.

In order to edit our blueprint we need to open the blueprint editor. You can right click on the newly created actor asset and select Edit or you can select the actor asset and in the Details panel, click on Edit Blueprint-OpenBlueprint Editor. You will notice at the top of the editor tab, the name of your asset which easily helps you manage your blueprint editing. In the centre panel you will see above three tabs labelled Viewport, Construction Script and Event Graph. Switch to the Event Graph tab. This is where we add our visual scripts. You will see some disabled nodes.

The Event BeginPlay node will execute when play commences for your Actor. On the node you will see an execution pin on the right hand side. Right click on the execution pin and drag and drop in the empty space outside the node. You will see a menu appear that allows you to select other nodes. For our example we require the Print String node. Ensure context sensitive is checked and type in Print String. Select this node. Your Event Graph tab should now look like Figure 1. (Note - Determining what nodes to select is all dependant on what functionality you intend building into the blueprint.)

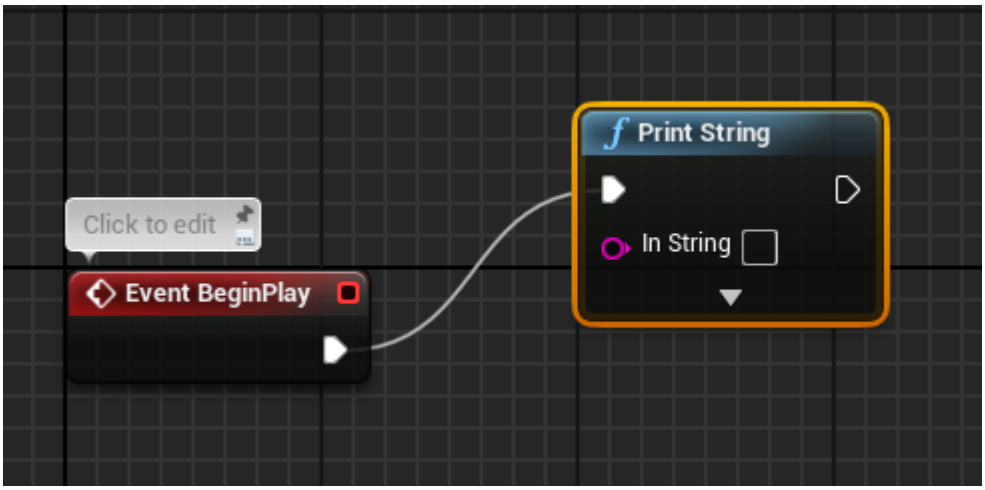
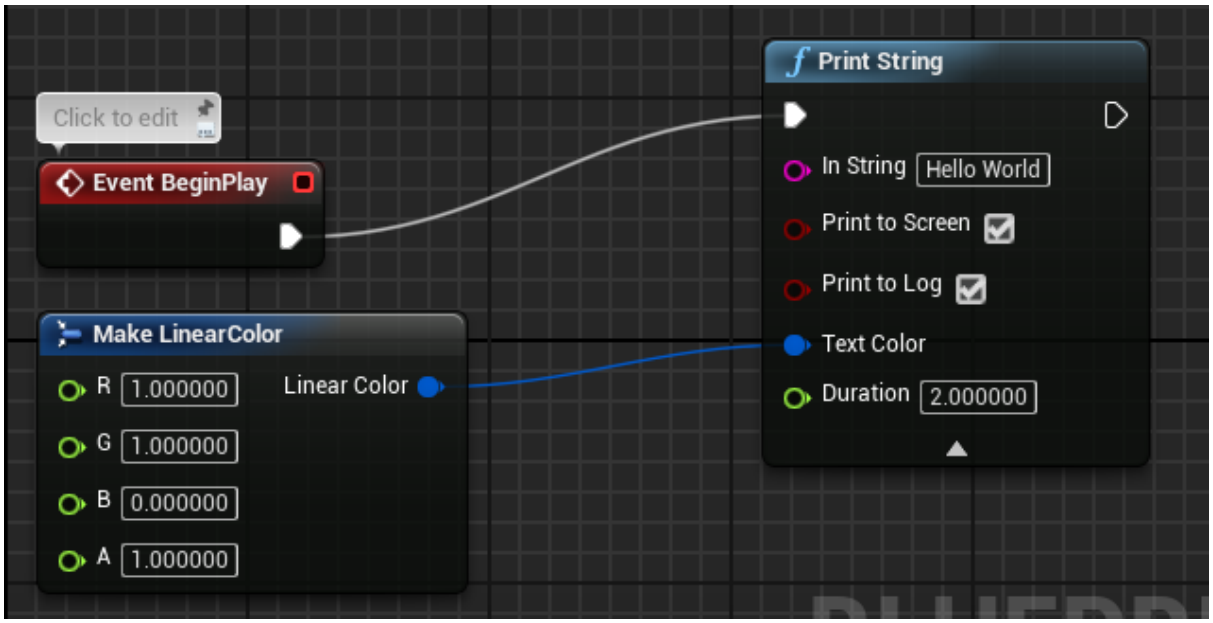


Figure 1: Print String Node

If you look at the Print String node you will see an empty Textbox. Type in the word "Hello World" Then click Compile and then click Save. The compile result is shown at the bottom of the screen in a separate panel. If you have compile errors then you will need to fix your scripting. Minimise the blueprint editor and in the level editor click on the play button. You will see the text Hello World displayed for a few seconds before disappearing. Another observation you would have made is that there is an expandable menu button in the Print String node. If you click this it will show you more connections that allow you to specify things like the colour of your displayed text and the duration the text is displayed on screen.

If you click the Text Color node and drag into the empty area of the Event graph you can add a Make LinearColor node which allows you to specify the R, G, B and transparency values.



Currently our Actor is not visible during play mode purely because we haven't added any 3D mesh components. Under the Components Tab in the blueprint editor for your actor click Add Component, select Cube and press enter. If you go back to the level editor you will see a coloured Cube in your scene. The cube position, rotation and size can be changed by either manually moving the cube with colour handles for each mode or entering numeric values in the details panel under the transformation category.

We will now create a Visual Script that causes the Cube to change colour and play a sound file when the player bumps into the cube. To accomplish this we need a mesh hit trigger. But before we do this lets first add an arbitrary sound file to your Content Browser. We will also create a new material which will be used to colour the Cube.

Creating a Material

In the Content Browser open your materials folder and right click and select Material. Give your material a name then double click to open this in the Materials Editor. In the materials editor you will see a node already present with the same name as the material. Right click anywhere in the material editor and type in and select Vector Parameter. Drag from the top node and connect to the Base Color input node of your new material. Double click on the rectangular box inside the vector parameter node and pick a colour. You can rename this node in the details panel to Colour. If you want your material to have a metallic look, add another node called scalar parameter. Rename this node Metallic and change its value to 1.0. Then connect its output node to the metallic input node of your new material. You will see a change to the appearance of your material in the preview window. Save your new material.

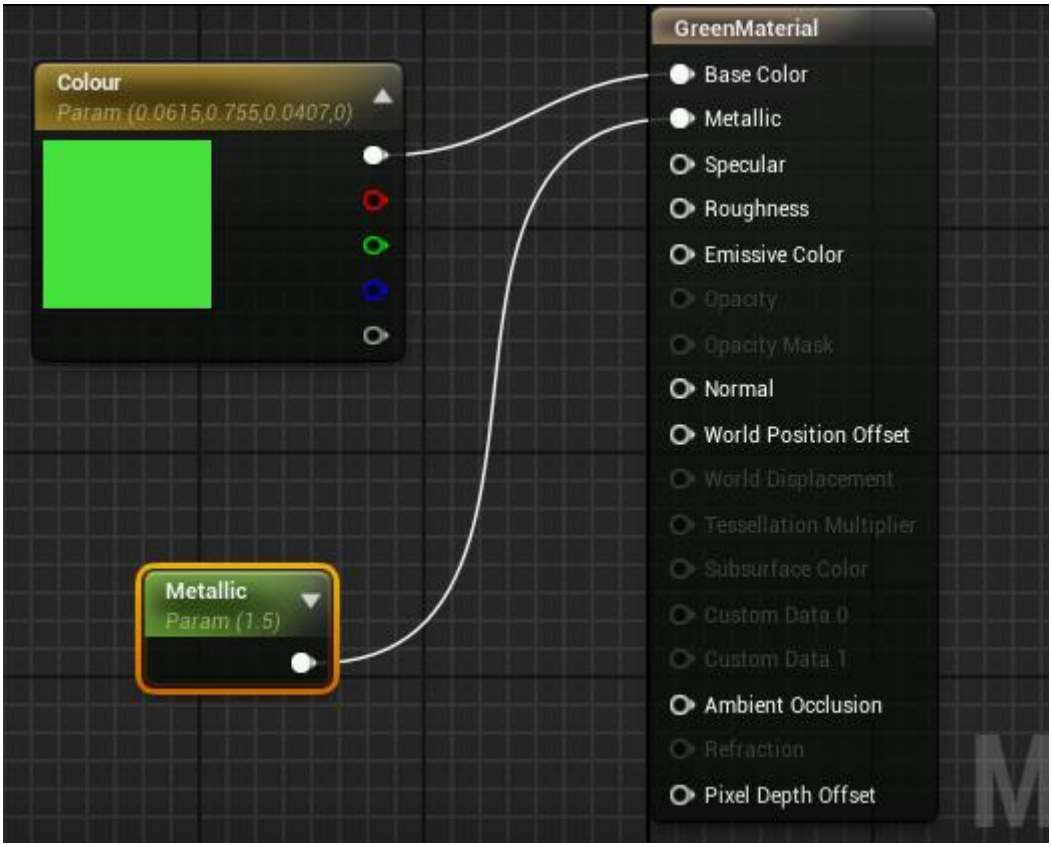


Figure 3: Custom Material in Material Editor (Add a Vector parameter & Scalar parameter nodes)

Adding Scripting Logic to Trigger the Collision

Open the actor blueprint and select the Cube. In the details panel scroll to the section on events. Click the plus sign next to the On Component Hit event and select On Component Hit (Cube). This will add the corresponding node to the Event Graph (Figure 4). Drag from the execution pin and create another node called Play2D Sound. From the sound drop down box select the name of the sound file you added previously to your Content Browser. From the output execution pin of the play sound node add another node called set material. Select the new material that was created above.

Compile and save your script then go back and play your level. Use the arrow keys to move your player in the direction of the Cube and bump into it. You will observe that the cube changes colour and also the sound file is played.

(Note - There is also a play sound at location node that plays the sound based on the location of the actor. In this case you will need to retrieve the location of the actor and pass this into the location node.)

One thing that is not pleasant is that there is attenuation of the sound for each contact the player makes with the Cube. How do we rectify this? Well we can do a check on the cubes material and direct control flow to changing the colour and playing the sound.

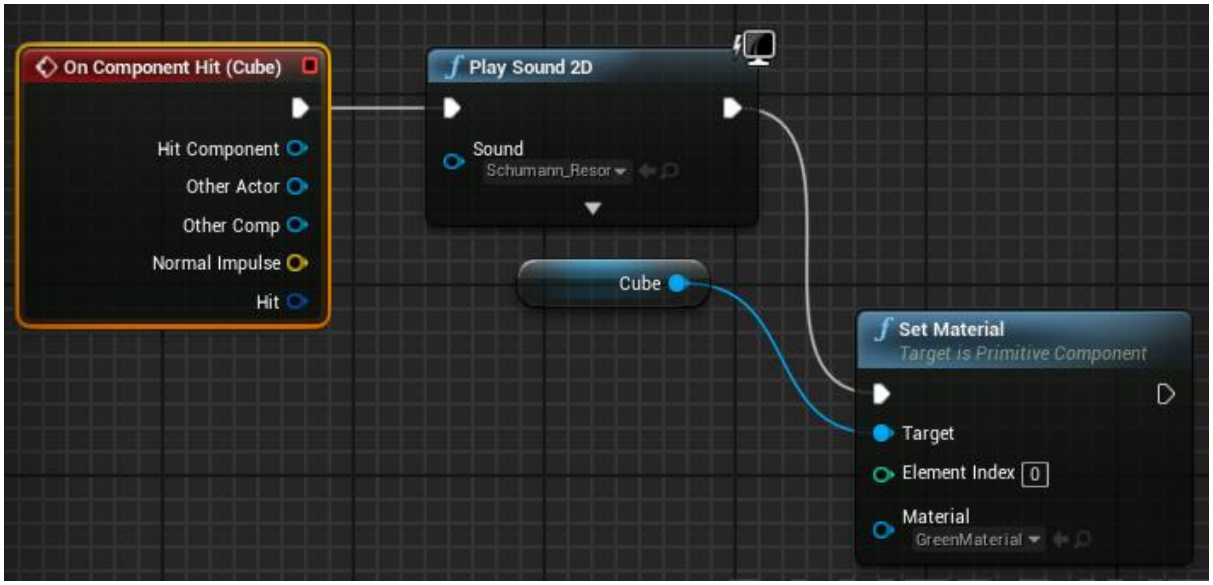


Figure 4: Hit trigger to play sound and change the colour of the actor (Cube)

Using Control flow logic

First we need to create a Newmaterial variable which is of type Material and initialise its value to that of the custom material we created before. Open the actor blueprint and under the My Blueprint tab click Add New and select variable. Rename the variable to Newmaterial in the details panel. In the variable type field scroll down to Object types and select Material. Compile and then change the default value of this material to that of the material we created before. I had called it GreenMaterial.

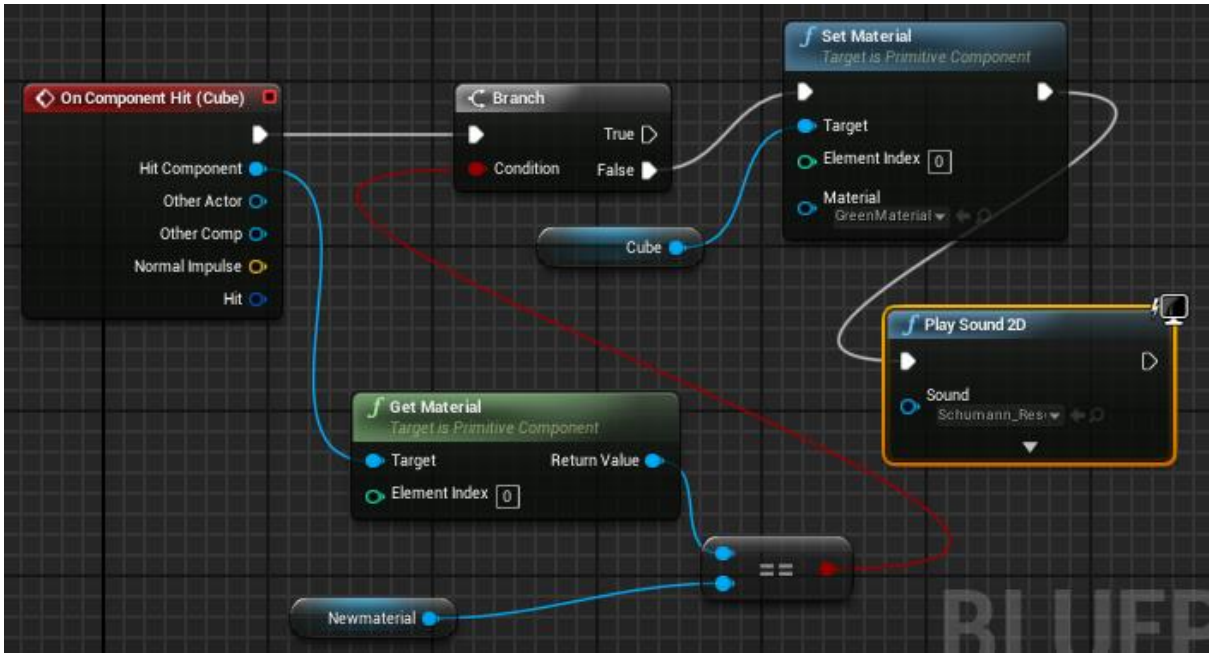


Figure 5: Script to change cube material and play a sound file

In the Graphs category under My Blueprint find and double click on On Component Hit(Cube). This will display the Event Graph for this node which should be the same as that shown in Figure 4. Now we will modify this to resemble Figure 5.

Break the link on the execution pin by pressing the ALT – key and left clicking with your mouse. Then drag from this execution pin and add a branch node. From the My Blueprints panel drag the Newmaterial variable and drop it in the Event Graph. From the hit component node add a Get Material node. Add an equal (Object) node. This will compare the current material of the actor (Cube) with that of the Newmaterial. If the condition evaluates to false then it executes the Set material and Play Sound nodes. If it is true then we don't need to do anything. Now when you play and bump into the cube it will only execute the Set Material and Playsound Nodes once.

Animating our Actor (Cube)

Our actor does not look very exciting so let's add some animation. We will add scripting to make the cube rotate in the x-y plane and around the z axis.

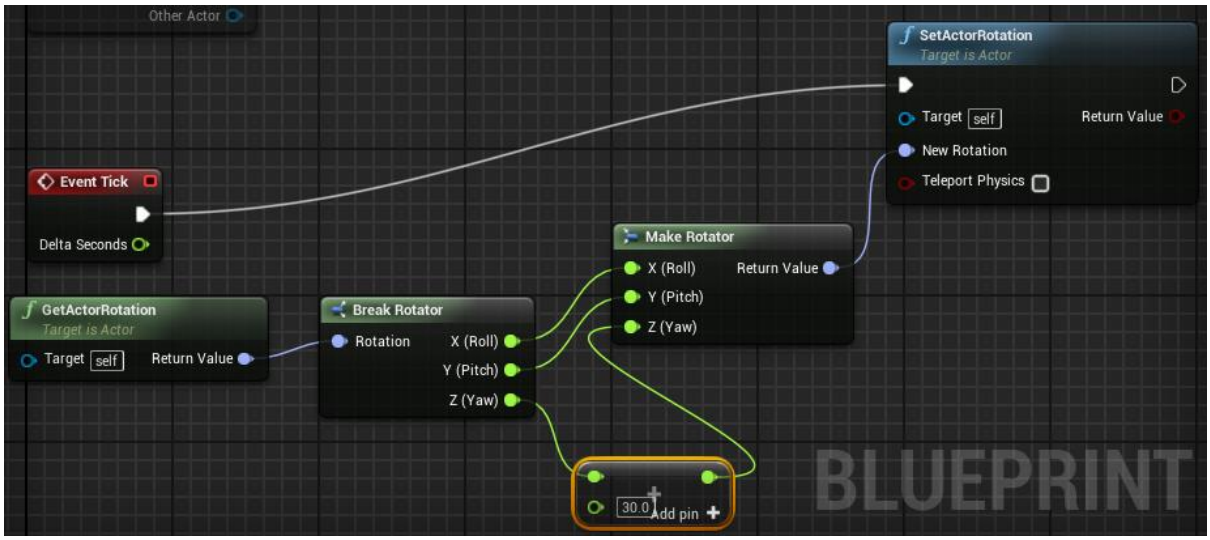


Figure 6: Script to rotate the cube

Open the Event Tick node and add a SetActorRotation Node. This will update the actor's rotation every frame. To achieve this we need to first get the actors current rotation and add a value to the Yaw rotation component. So we break the GetActorRotation rotator value into its x, y and z components. Introduce an addition node to add say 30.0 degrees to the current yaw component. We pass this into a make Rotator which returns its rotator value to SetActorRotation. If we compile this script and play our game you will see the Cube rotating at 30 degrees per frame around its Z axis.(Shouldn't we place a restriction on the upper limit of the yaw value as a float goes up to 3.4x10^38)

Adding instances of our actor during Runtime.

There will be cases where objects are introduced into the game environment during play. To accomplish this one needs to create an instance of our object. Let's say we added an actor to our level and converted it to a blueprint. To create instances of this actor all we need to do is add the Spawn Actor from Class node. Specify the class name as that of our actor and use a Make Transform to initialise the Spawn Transform input node.

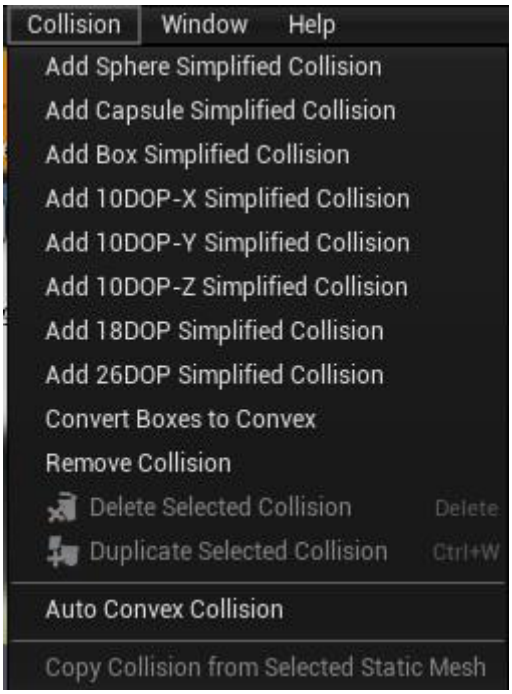
Adding Collisions to a Static Mesh

A static mesh must have a collision set in order for the player not to walk through the mesh and to allow the mesh to simulate physics (When checked). Open up the static mesh in the editor and check that collision mesh is displayed. Then Access the collision menu and select an appropriate mesh. For more complex meshes first remove the current collision meshes and select auto convex mesh.

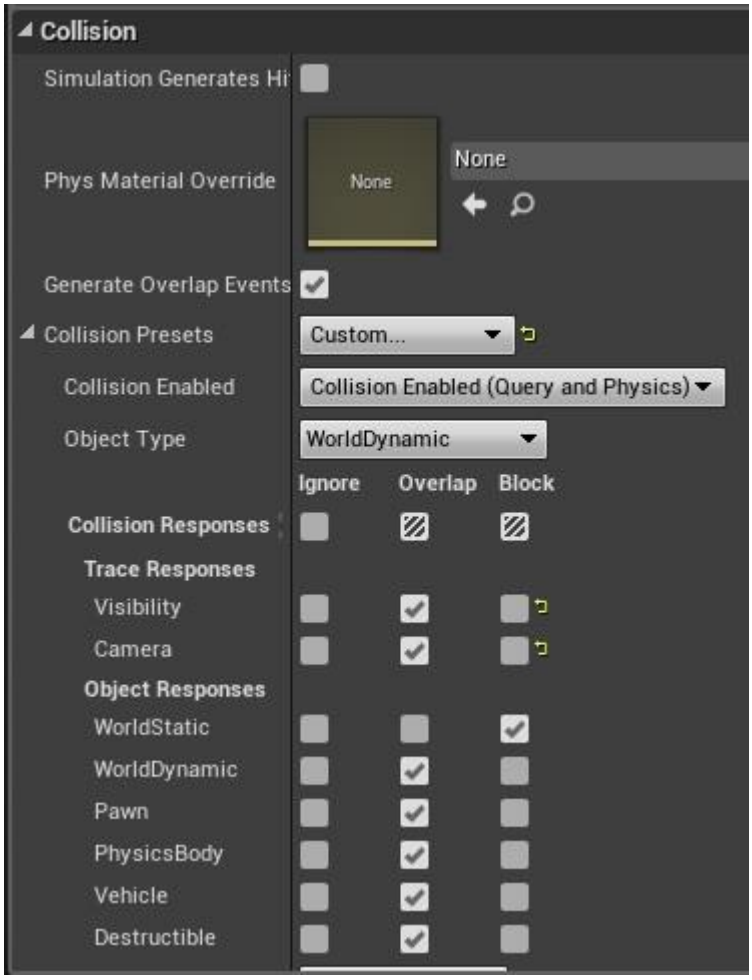
Working with Overlap Events

Let's consider a simple example where we have a *player start* and a static mesh. The objective is to display a user interface menu when the *player start* overlaps the mesh and hide the menu when the player moves away from the mesh. We also do not want the player walking through the mesh nor the mesh falling through the floor. The first step required is to ensure that our mesh has a collision component. To do this open the mesh asset in the mesh editor by double clicking the mesh asset in the *content browser*. From the collision toolbar click the drop down arrow and select simple collision.

Check to see if a collision mesh surrounds your object. If there is none then we need to add a collision from the collision menu.

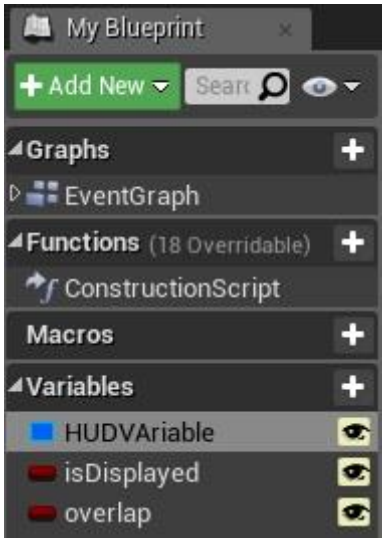


Depending on the geometry of your objects mesh select a box or sphere collision. Save your asset. Then drag and drop your asset into your scene/level. Select your asset in the level and then go to the *details* panel. Note in order for your asset to simulate physics it needs to have a collision. Under the physics category, check *simulate physics*. Under the collision category check *generate overlap events*. Leave the *simulation generates hit events* unchecked. Expand the *collision presets* section and chose the custom preset. Configure all the parameters as shown in the following image. With this configuration we will be able to catch overlap and end overlap events. Furthermore your object will not fall through the fall nor will the *player start* pass through the object.



Next convert your mesh to a *blueprint* and open this in the blueprint editor.

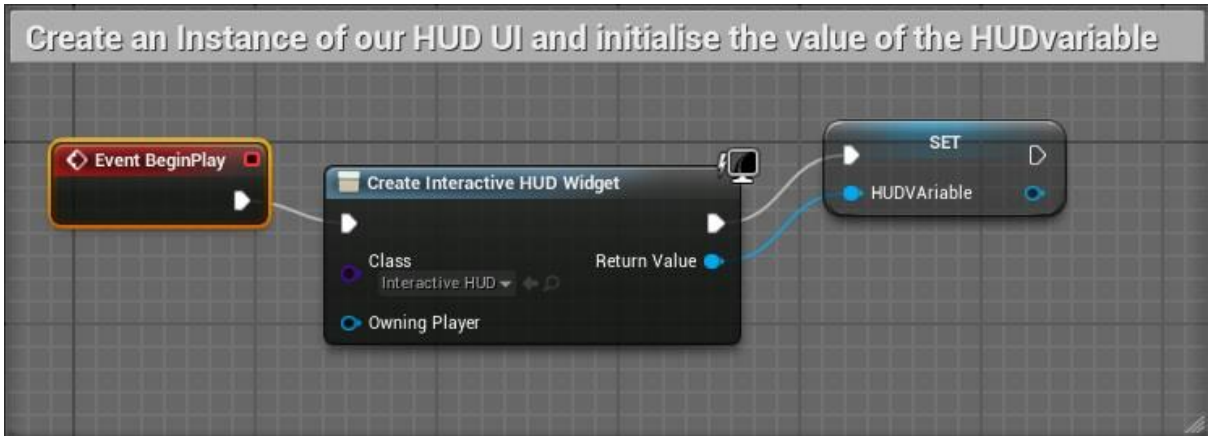
In the blueprint category panel create three variables and name them HUDVAriable, isDisplayed and overlap. Set the type for isDisplayed and overlap to Boolean and that for the HUDVAriable to User Widget. Make sure the *instance editable* field is checked and compile and save your blueprint.



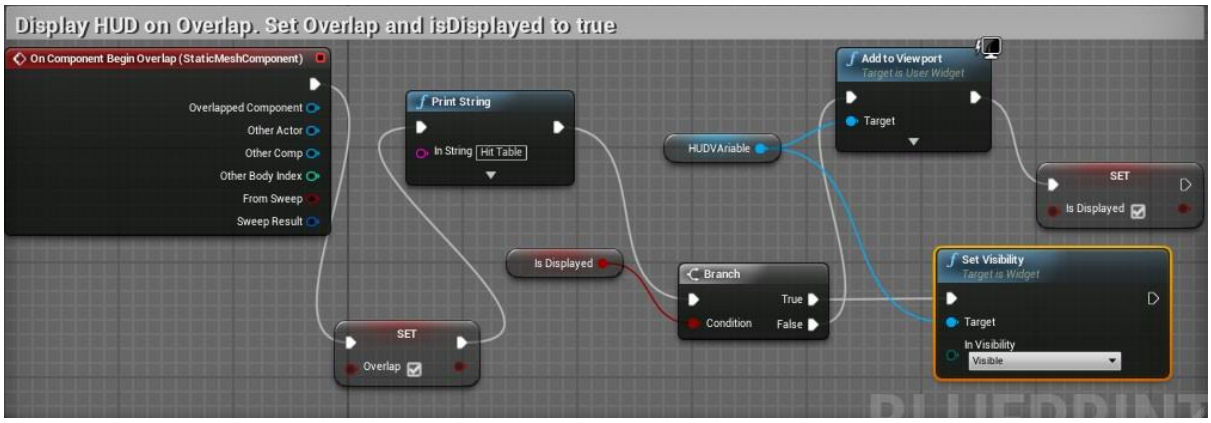
Before we add our logic to our blueprint lets first create our user interface. Go to the *content browser* and right click and select User Interface ->Widget Blueprint. Name this InteractiveHUD and then open

this in the user interface editor. Add some content to your user interface screen like an image or button. Compile and save.

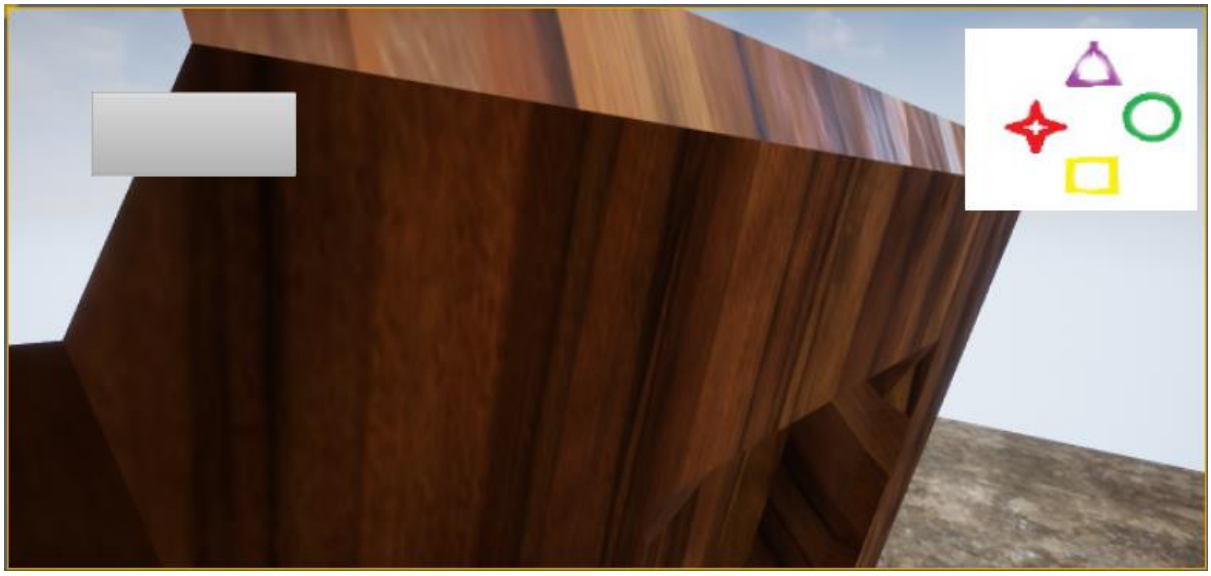
Now go back to our asset blueprint and create the following visual scripts. The first script creates an instance of our user interface and takes its return value and uses this to set the value of our HUDVAriable we created previously.



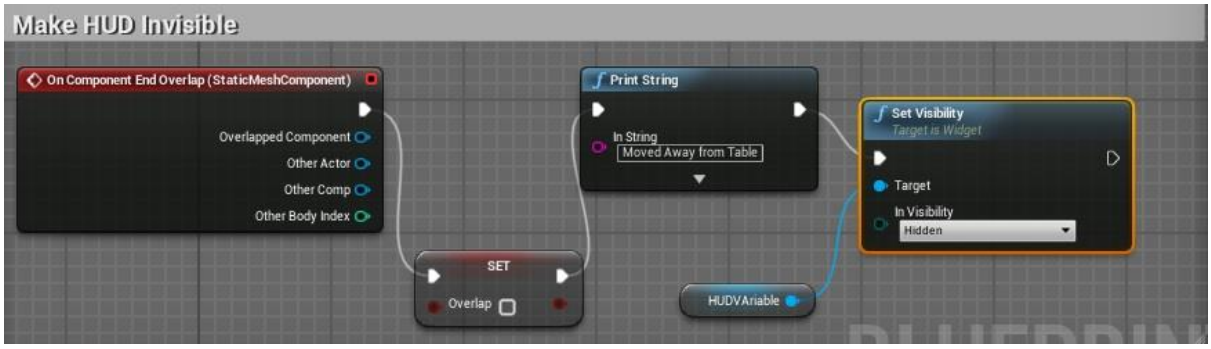
In the next script we add an OnComponent Begin Overlap node for our Static mesh component. We set the overlap variable to true. Then we check if our user interface is displayed based on the value of the variable isDisplayed. If it is false then we add our user interface to the viewport and set isDisplayed to true. If isDisplayed is true then we set the visibility of our user interface to visible.



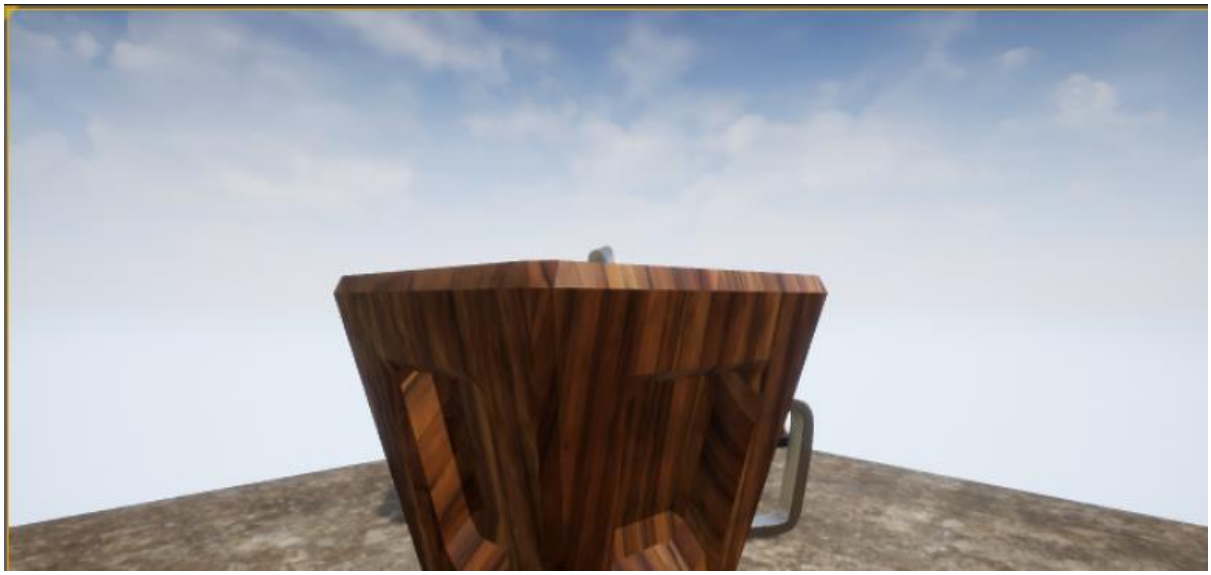
Finally we add an OnComponent End Overlap node, set overlap to false and set the visibility of our user interface to hidden. Now go to play mode and move your player so it overlaps your mesh object. You we observe that your HUD will become visible when there is overlap and becomes invisible when overlap ends. You HUD display may have images depicting keyboard or controller inputs the player can press like an X key or square button on a joystick.



UI is Shown when player overlaps the interacting object



Script to make UI invisible when player moves away from interacting object



User Interface is Hidden when player moves away from the interacting object

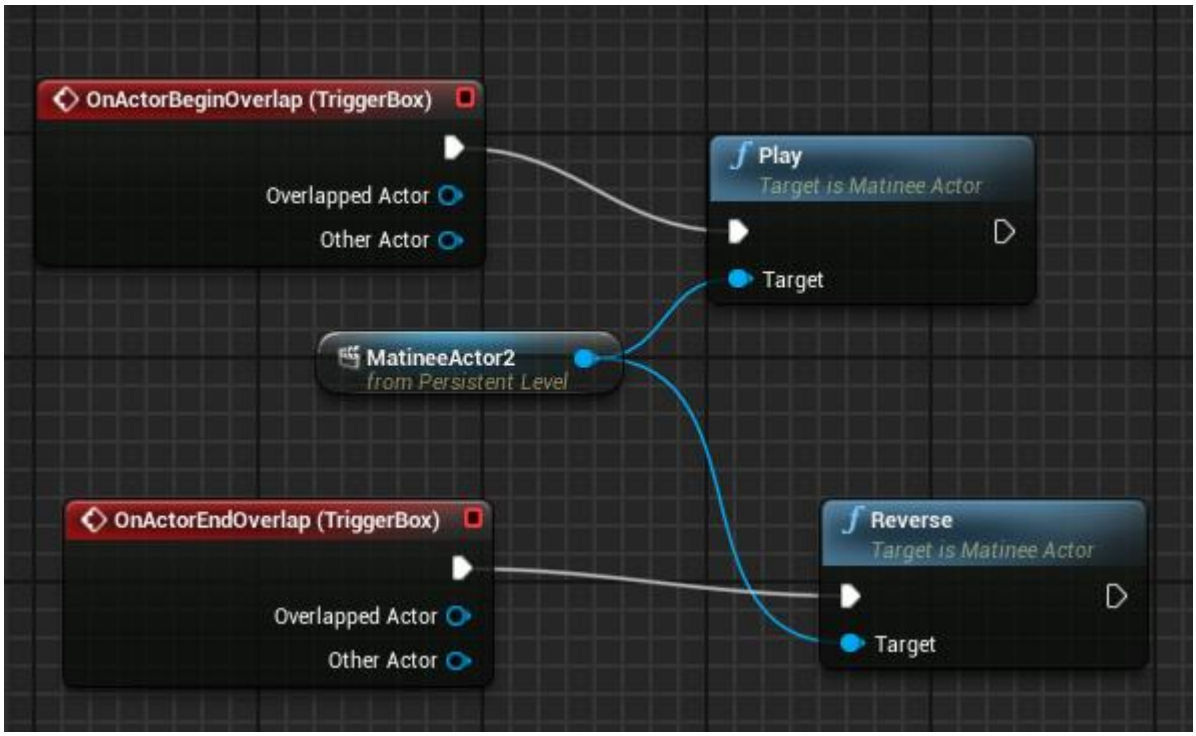
Using Matinee for Animations & Cinematics

The Matinee tool editor can be used to create animations and cinematics. The basic operation involves creation of keyframes for an actor at various time intervals along a linear track very similar to Blender. The same can be applied with a camera to make cutscenes. With skeletal meshes for animated characters you can use Matinee to create predefined animation paths by inserting movement tracks and playing back unique animations.

To create Matinee add a Matinee Legacy Actor from the Blueprints toolbar in the main level editor. In the Matinee editor under the Tracks tab, Add a new Empty Group and give it a name that you can identify with your level actor. Then right click on this group and select Actors->Add Selected Actors (Make sure your actor is selected in the Level).

Then right click on your group and select Add Movement Track. Note a key frame will be added at time 0. Now move the black slider time bar to the next key frame by clicking in the time line. Go back to your level and select your actor and apply the necessary transformations (Scale, Translate, Rotate). In Matinee click on Add Key Frame(Or press ENTER key). Repeat for other key frames. Then finally move your end track to the timeline position of your last keyframe. Save.

To use the Matinee one needs to reference it in a blue print level and apply the play or Reverse node commands. Usually a Box or sphereTrigger actor will be used to get the Overlap and end overlap which will then play the matinee. The example of the Door opening and closing is well described in the Unreal Engine Documentation. Refer to this example.

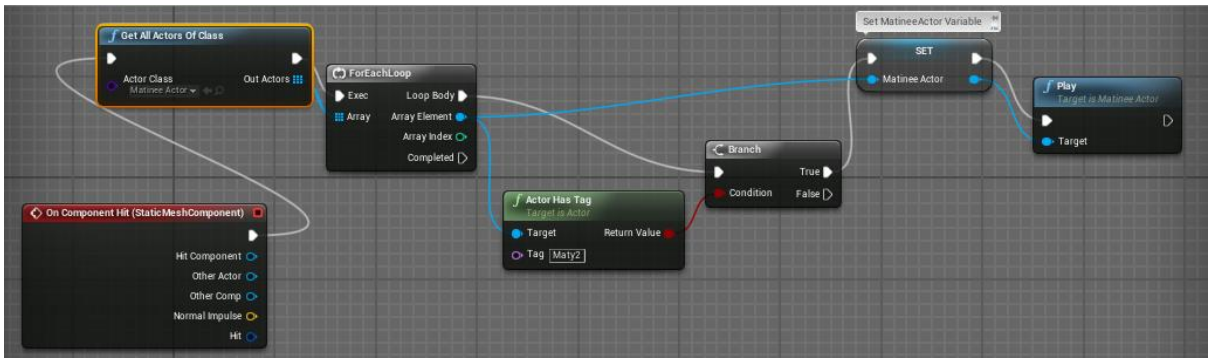


Accessing and playing Matinees from blueprints

Many new users have encountered this issue of having several matinees and being unable to get references within blueprints to play these matinees based on player interaction.

When Matinees are added they are placed in the level. Hence they are not directly visible from inside a Blueprint except for the Level Blueprint. To play a specific Matinee from inside a blueprint, one first needs to get an array of references to all the Matinee actors(Get All Actors of Class). One uses a for-each-loop to iterate over the array and query a specific actor variable such as a tag(Actor Has Tag). Then return the instance of this reference and store it's state in a variable of type Matinee inside the blue print(So you need to create a variable in your blueprint of type Matinee). One then calls the play node to play the Matinee (Play – Target is Matinee Actor).

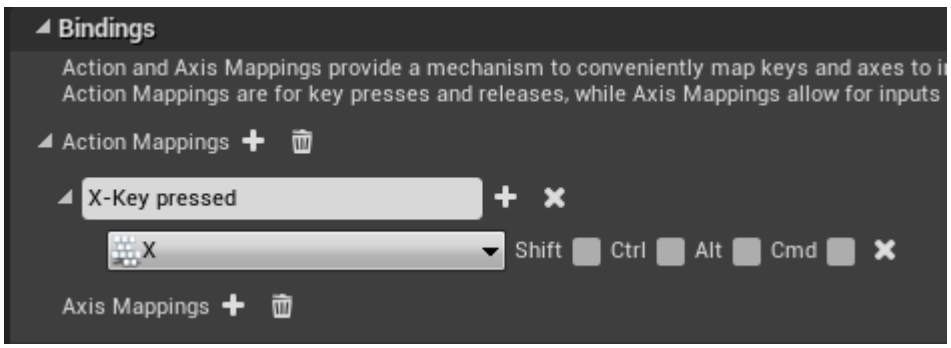
In the following Visual script, the Matinee is played following contact of the player with a specific actor in the level.

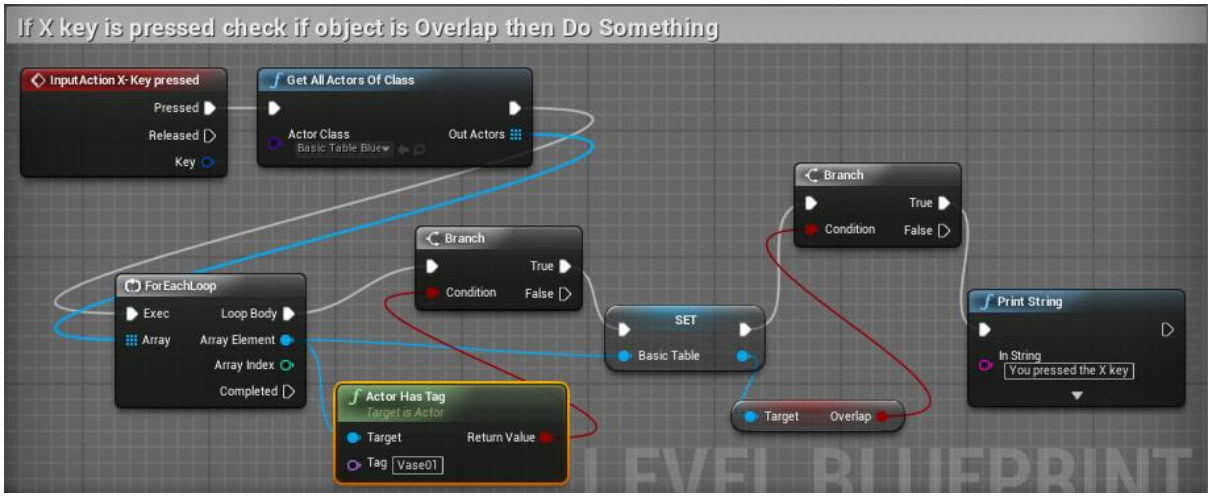


NB If two or more Matinee Actors are assigned the same Tag then these Matinees will all play in sequence as they are positioned in the array reference returned from Get All Actors of Class.

Checking for a Key press

Earlier on we made a UI pop up when the player overlaped with a game Actor. The following script processes a key press in the level and then checks whether there is overlap with the actor before doing something. Before we complete the script we first need to add a binding for the X key in the Project settings under Engine-Input to process the key input.





NB if you had rather place the InputAction node in the Blueprint for the actor it would not work as the Keypresses are processed in the level.

That's about it. These are just some of the basics to get you on your feet.