Creating a 360-Panorama VR Viewing App in Unreal Engine for Android devices

This tutorial will cover the creation of a simple VR app (aimed for a suitable Android mobile device within Google Cardboard) and providing gaze interaction viewing of several 360-degree panorama images.

While Google Cardboard is often derided by the more serious VR developers and users, it is nevertheless the least expensive, pervasive and the most likely platform for the majority of people to experience VR for the first time. According to Google (2017), the company has shipped 10 million cardboard VR sets and seen 160 million downloads of its free template.
Refer: https://techcrunch.com/2017/02/28/google-has-shipped-10m-cardboard-vr-viewers-160m-cardboard-app-downloads/

The most common VR app downloaded for mobile device viewing in Google Cardboard are typically demos that support content for tourism, education and games, and online 360-degree content – e.g. YouTube. The following links list the most popular VR apps for Android/iOS devices and Google Cardboard:

http://www.androidauthority.com/best-google-cardboard-vr-apps-622766/
http://www.makeuseof.com/tag/best-vr-apps-google-cardboard/
https://vrtodaymagazine.com/free-iphone-vr/

A cardboard template and instructions can be downloaded here:
http://www.instructables.com/id/Google-Cardboard-20/

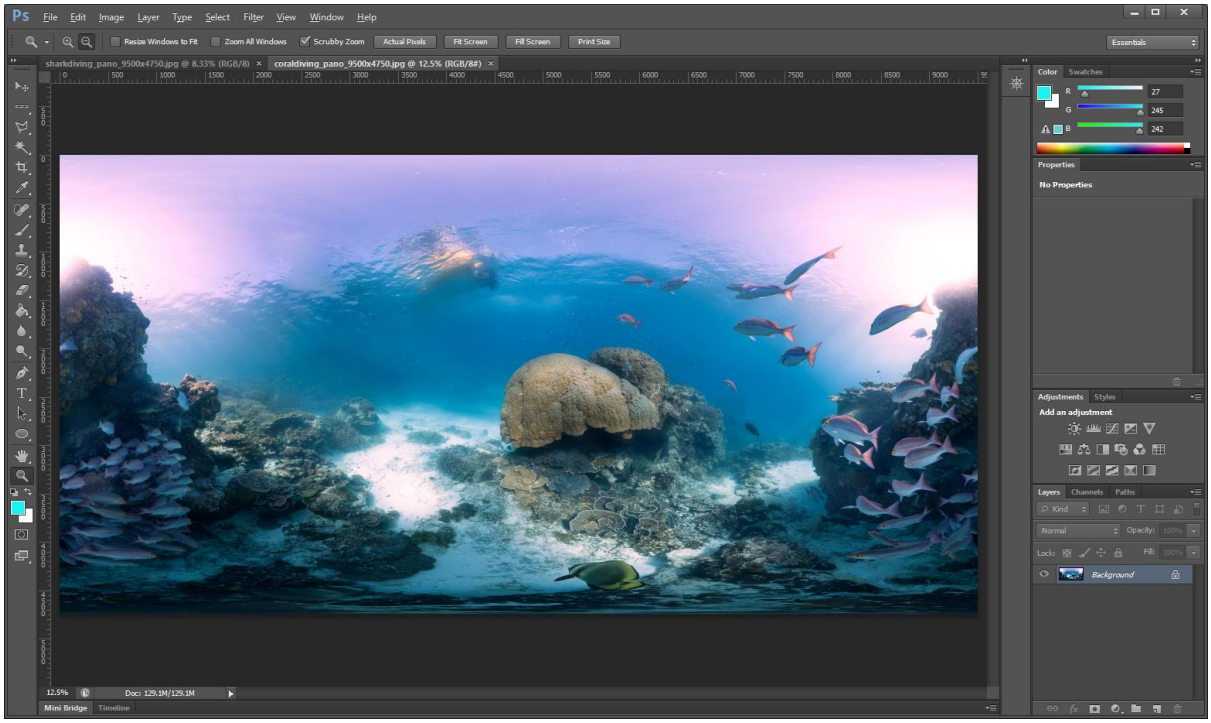
The tutorial covers 10 main steps:

1. Creating a cubemap using Photoshop and Blender
2. Setting up Unreal Engine for Android deployment
3. Setting up a VR camera for basic gaze tracking
4. Setting up a cubemap for Unreal Engine
5. Adding background music
6. Adding interaction via interfaces
7. Creating a gaze reticle using a UI widget
8. Working outside the Persistent Level
9. Creating a second interface and additional cubemaps
10. Tutorial videos and documentation links

1. Creating a cubemap (ready to import into Unreal Engine)

A cubemap is a cube which contains a mapping of 6 separate images (each image is mapped to a single cube face internally, and set up to display a seamless 360-degree image for VR projects using Unreal Engine.

Most panoramic images are in equirectilinear projection – typically with a width to height ratio of 2:1. Unreal can map these on to a sphere, but if the lighting and environment is to be matched in the best possible way, then a cubemap is required. This can easily be made using Blender and Photoshop.



360-degree image in equirectilinear projection

Image layout for a cubemap – with 6 sides specifically set up for importation into Unreal Engine.

Image 1: Positive X-Axis
Image 2: Negative X-Axis
Image 3: Positive Y-Axis
Image 4: Negative Y-Axis
Image 5: Positive Z-Axis
Image 6: Negative Z-Axis

NOTE: Details of this process was taken from:

https://evermotion.org/tutorials/show/10738/creating-custom-sky-from-hdr-image-in-unreal-engine

First – get your images!

Choose an image – either a 360-degree photograph taken with a 360-degree camera, or a rendered image from a 3D software program or a downloaded one. The image should be at least 4096 x 2048 pixels (width: height - 2:1 size ratio) or higher. Often these will be in the compressed .JPG format (others may be in the uncompressed .TIFF format).

Second, open the images in Photoshop

Open the panorama image in Photoshop and convert to RGB 32-bit mode IMAGE -> MODE and check both RGB and 32-BIT. Next, resize the image to 4096 x 2048 pixels (resolution 72-150ppi is fine), if not already done so. Finally, go to FILE -> SAVE AS and select the RADIANCE .HDR format.

Note: You will need the NVIDIA texture tool plugin for Photoshop to be able to save as .HDR.

NVIDIA Texture Tools for Photoshop plug-in available here:
https://developer.nvidia.com/gameworksdownload#?dn=texture-tools-for-adobe-photoshop-8-55

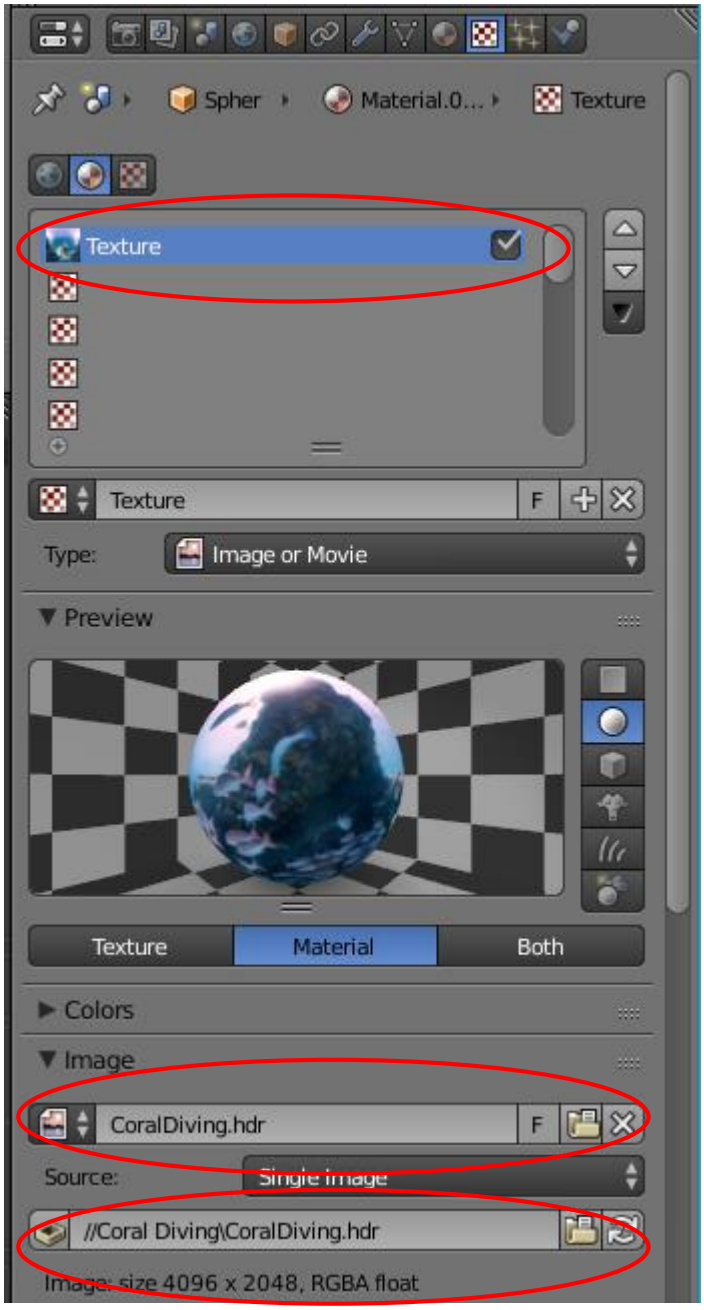
Information regarding this plugin available here:
https://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop

Third, Blender

Open the file “CubeMap_Scene.blend” in Blender and select the sphere object in the OUTLINER. Select the TEXTURE editor and you should see that an image is already mapped as a texture to the sphere. Delete the image and import another – your .HDR panorama image.

Blender file is available here: https://evermotion.org/files/upload/CubeMap_Blender_Scene.rar

Blender 3-D software is a free download available here: https://www.blender.org/download/



Texture settings for each HDR image for the sphere (note: you must have the sphere selected from the outliner window)

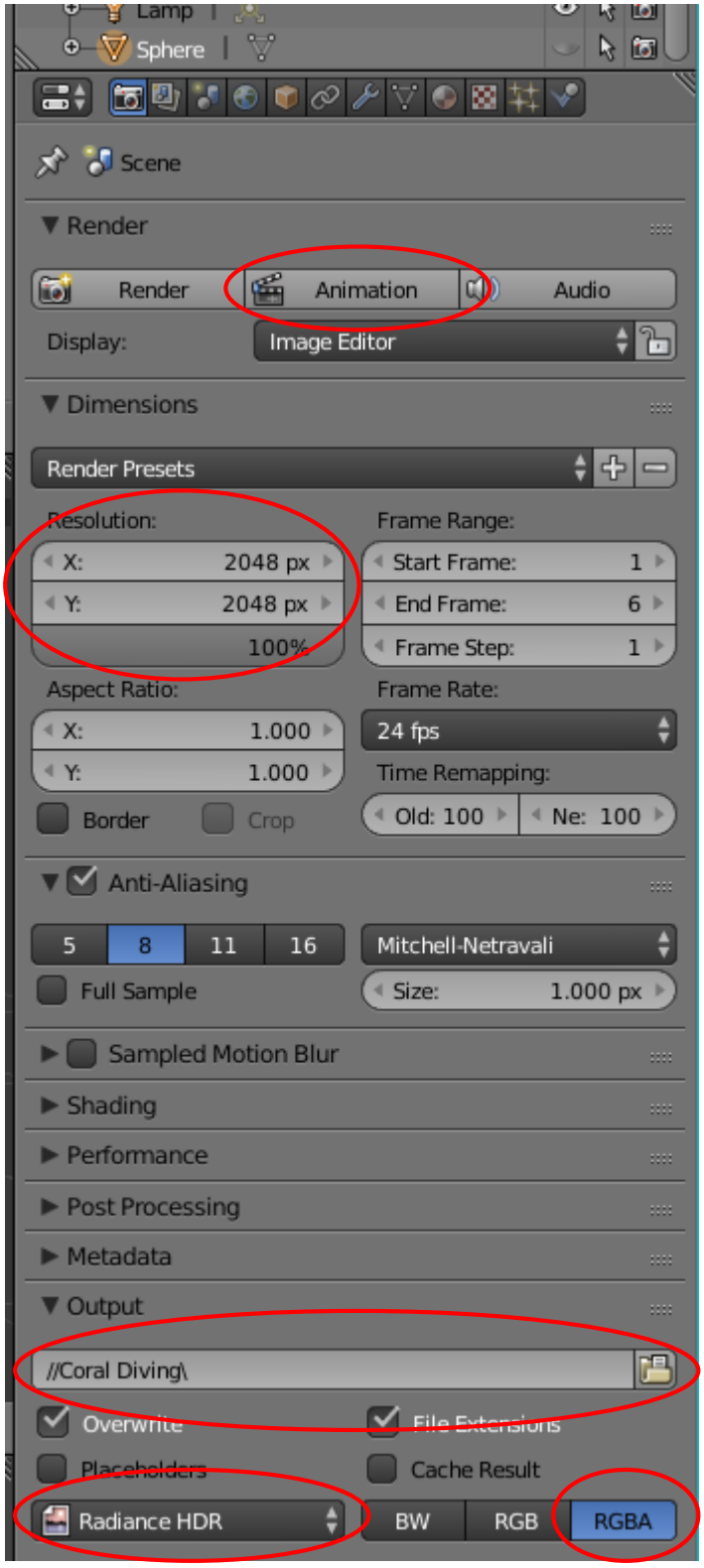
Next, select the RENDER editor. Change the RESOLUTION to 2048 x 2048 pixels (the size needed for each of the 6 images) at 100%.

Go to the OUTPUT and select the folder where you want the resulting renders.

Select the RADIANCE HDR format as the file format and choose RGBA as the output.

Finally, select the ANIMATION button and Blender will render out 6 separate images for each of the 6 frames in the animation.

The output will be 6 images in the precise orientation needed for the Unreal Engine cubemap.

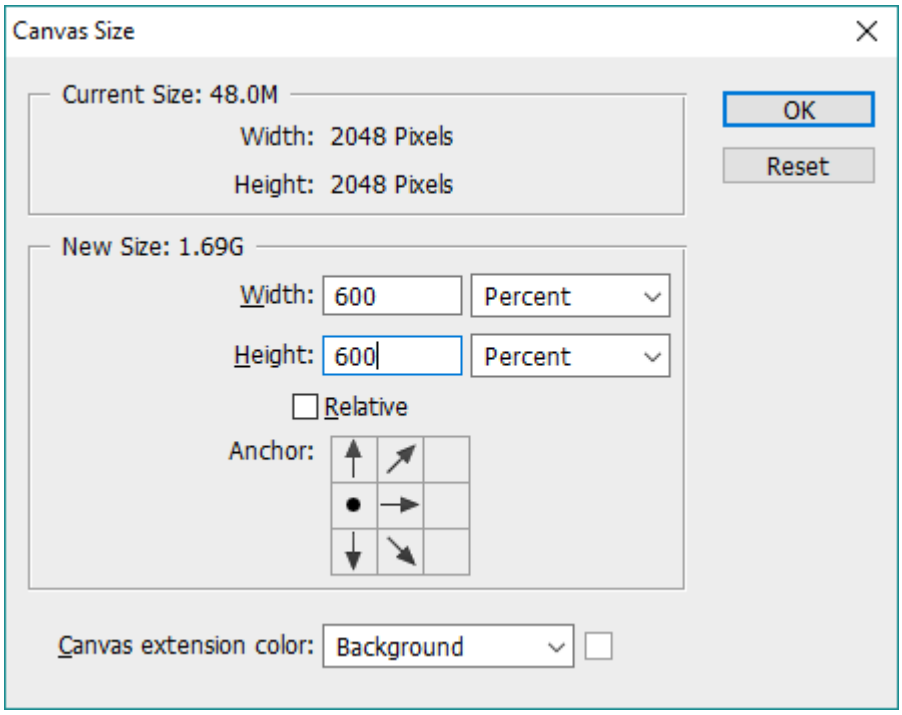


Render settings needed to output the six RADIANCE HDR (RGBA) images – click the ANIMATION button to start the renders.

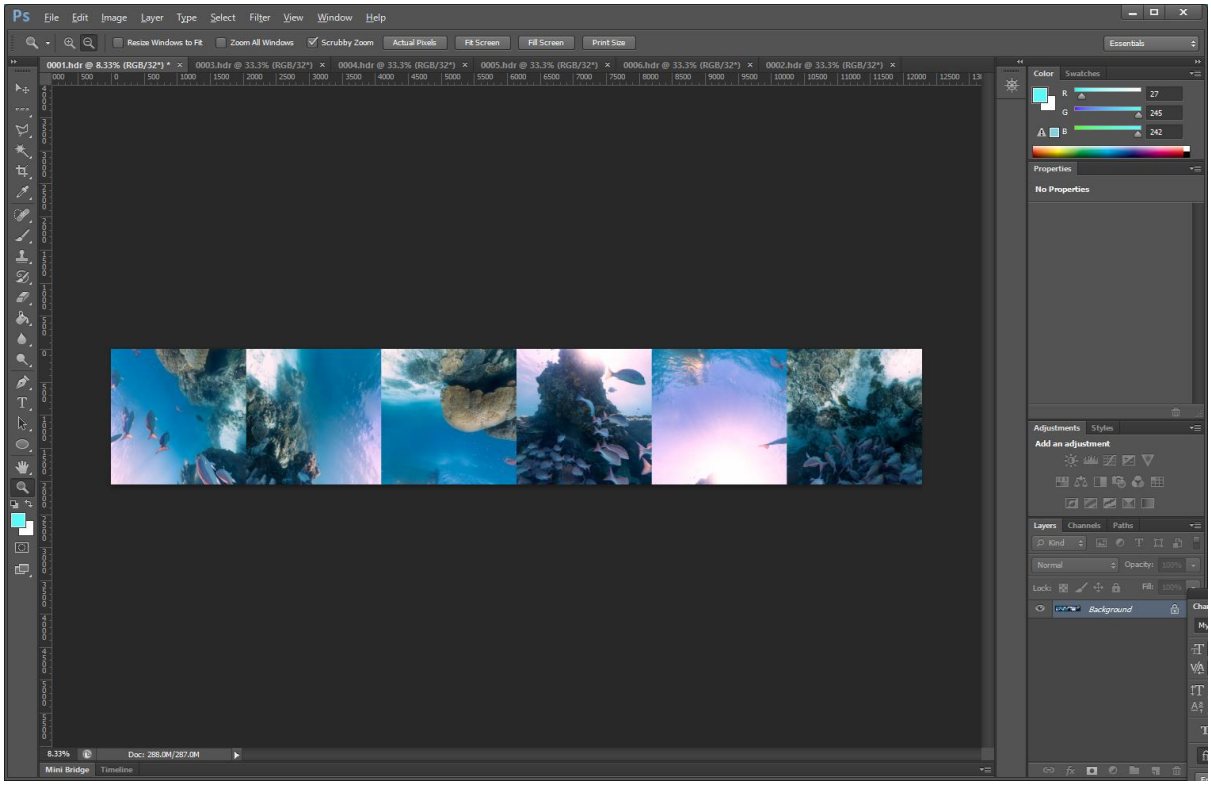
Four - back to Photoshop

Drag and drop all 6 .HDR images into Photoshop – they will open each in their separate window.

Go to the first image, and use the CANVAS tool (IMAGE -> CANVAS SIZE) – resize by 600% and click the left arrow button.



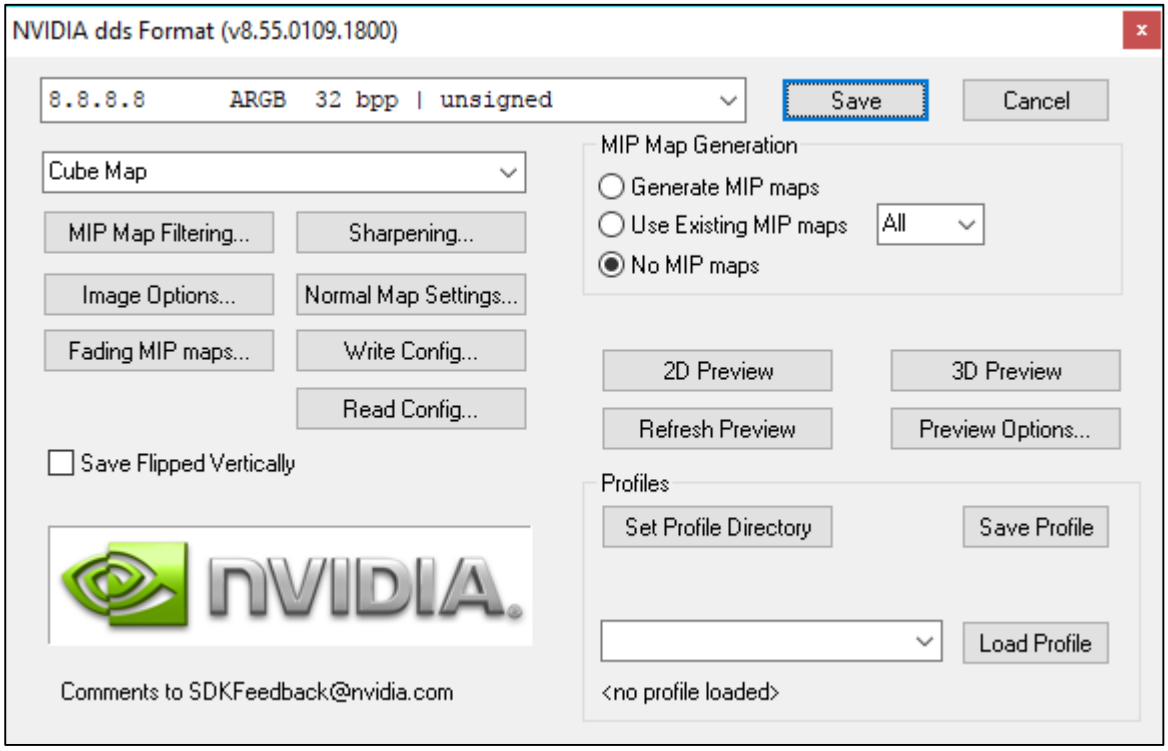
Collate each image in order as per example screen shot below (cropping will be required)



After cropping, the image size should be 12288 x 2048 pixels and the image mode should be RGB 32-BIT – check this before proceeding.

Save the file using FILE -> SAVE AS and select the D3D/DDS format option.

From the NVIDIA plug-in window:
- Choose 8.8.8.8 ARGB 32bpp | unsigned
- Cube Map from the Drop-Down menu of options
- No MIP maps



You should now have your image(s) (.DDS format) ready to import into Unreal Engine as a cubemap.

2. Setting up Unreal Engine for Android deployment

NOTE: This project targets the ANDROID Mobile Device using the Google Cardboard system. To deploy an Android .apk file (the mobile executable format), you will need to install the required NVIDIA CodeWorks for Android.

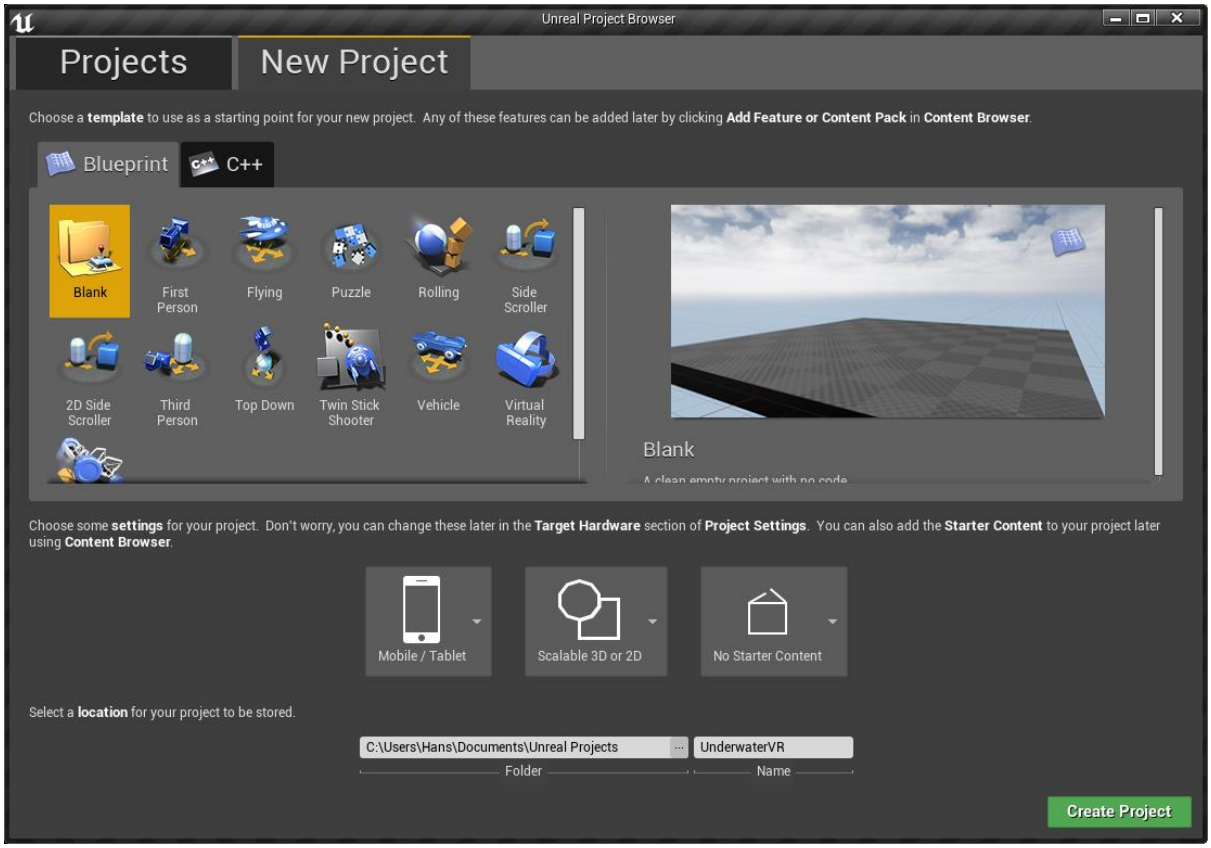
Once this is installed, you will then need to run the Android SDK Manager and install the required packages, including that for Android API 19 (ensure that this is checked for the download) --- why? This API version will be the most useful as it covers most Android user's smartphones that utilise Android 4.4 'Kit Kat'.

Step-by-step guide is available here:
https://docs.unrealengine.com/latest/INT/Platforms/Android/GettingStarted/

Start with Part 1: Required Android Setup
https://docs.unrealengine.com/latest/INT/Platforms/Android/GettingStarted/1/index.html

Some content was taken from youtube videos by Stigifo, available here:
https://www.youtube.com/watch?v=zLWV866RDvo [tutorial 1 – Android Project setup]
https://www.youtube.com/watch?v=_AEE7v164Ns [tutorial 2 – mobile VR build]
https://www.youtube.com/watch?v=p3LYkg8WiFo [tutorial 3 – packaging issues]

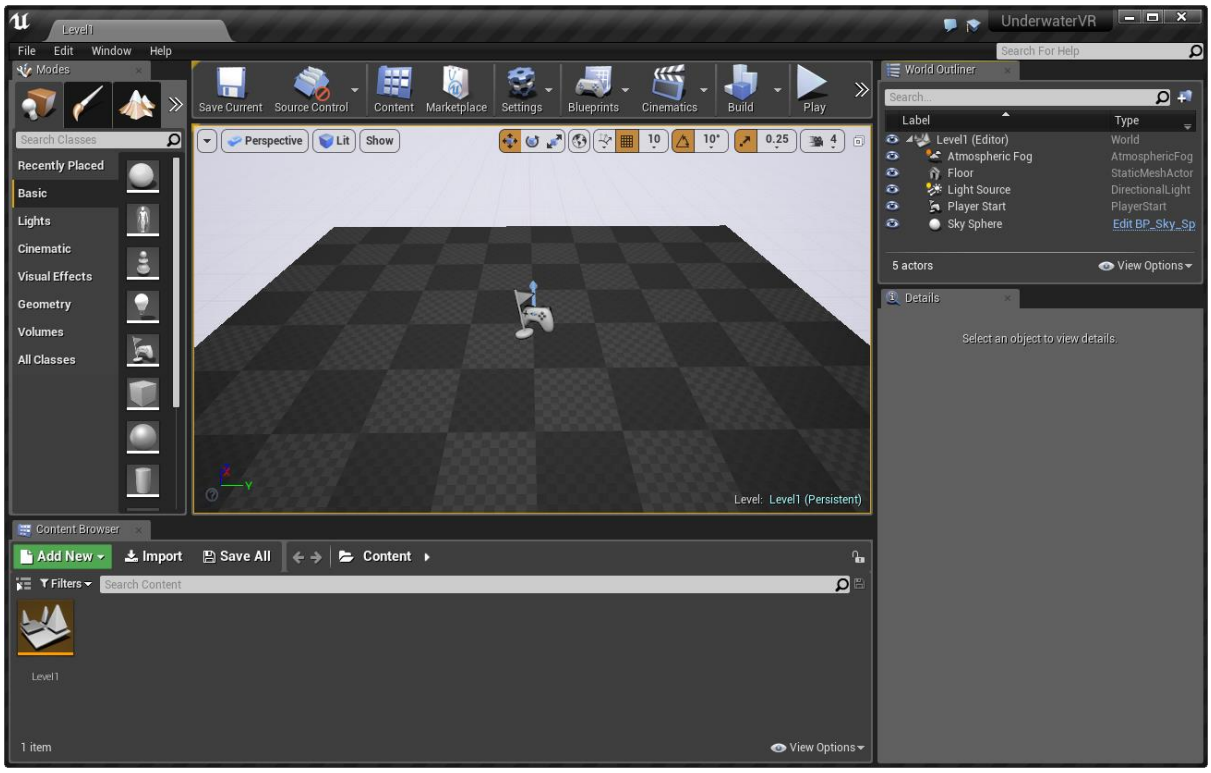
Start a new project in Unreal Engine.
In the settings, choose MOBILE/TABLET for the target platform, select SCALABLE 3D or 2D for the target's graphical level and NO STARTER CONTENT. Name the project accordingly and select the folder target and then finally click the CREATE PROJECT button.



You should have a default scene (untitled at this point) which contains the following components (known as ‘Actors’):

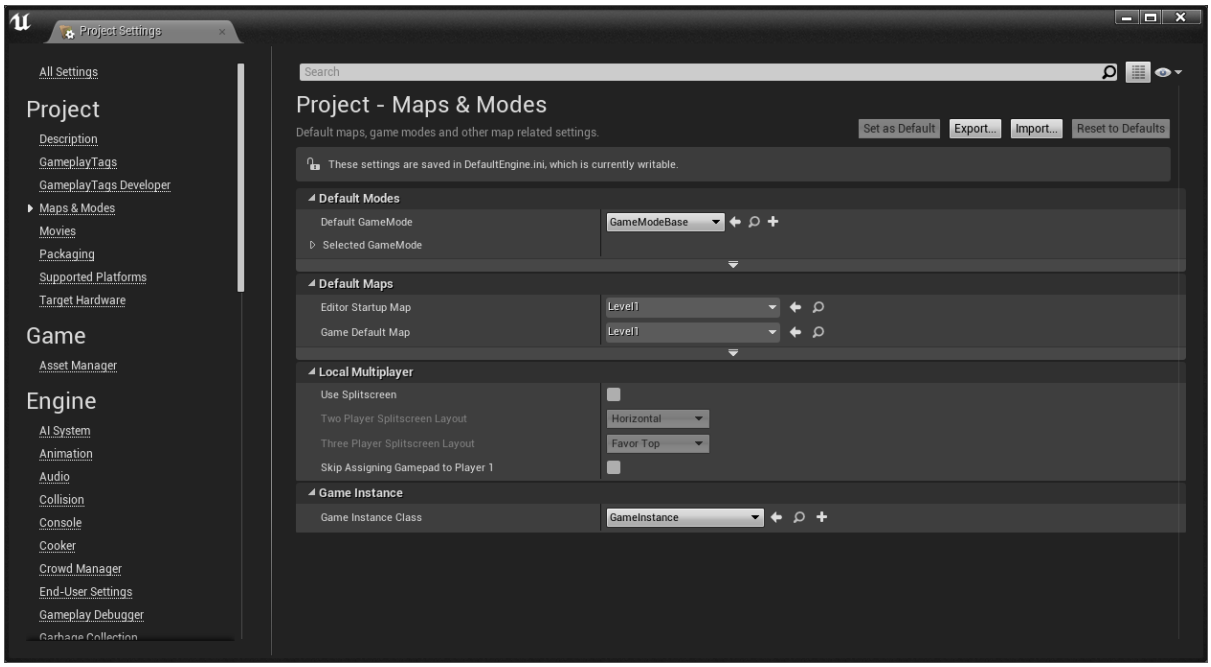
- Atmospheric Fog
- Floor
- Light Source
- Player Start
- Sky Sphere

Save this current scene using: FILE -> SAVE CURRENT AS (and name it “Level1”). The “Level1” scene will appear in the CONTENT BROWSER (bottom left).



In order to organise the Android target settings, go to EDIT -> PROJECT SETTINGS.

Under PROJECT -> MAPS & MODES -> DEFAULT MAPS, select the scene “Level1” for both the Editor Startup Map and the Game Default Map.

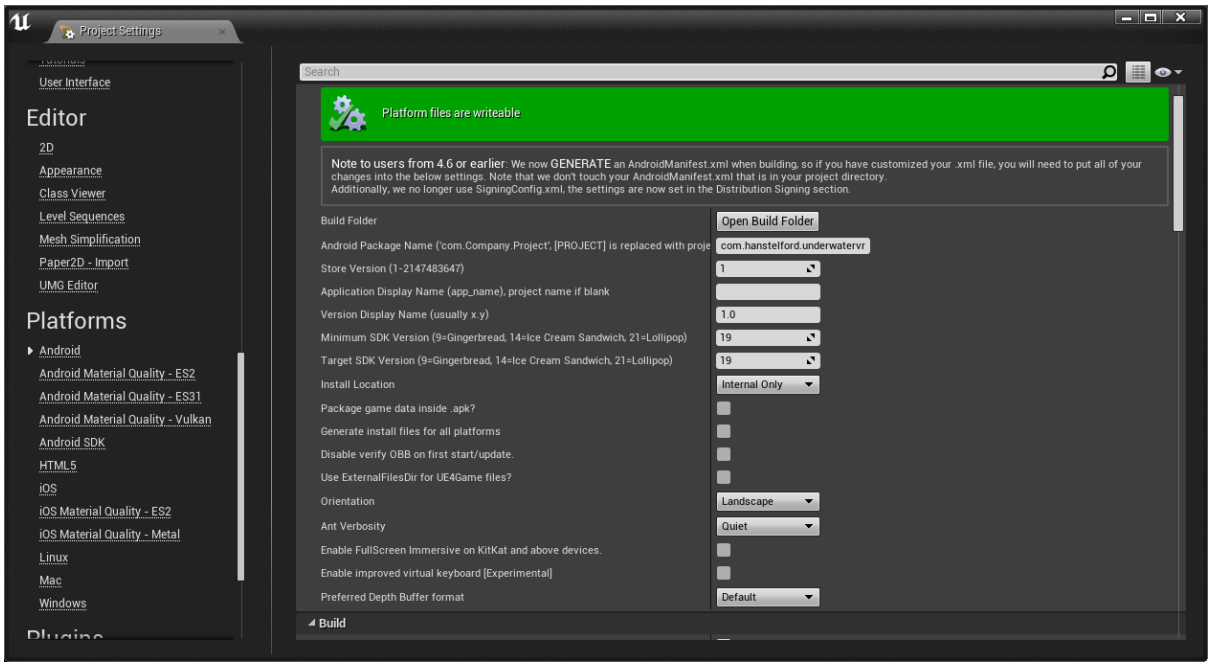


Under PROJECT -> ANDROID, click the Configure Now buttons (there are two here, both within a red box). When they are clicked, the box colour turns green. This ensures that the project is Android-ready.

At the top section, under APK Packaging, go to the Android Package Name, provide a suitable name in the following format (all lower case with no spaces): com.mycompanyname.projectname

Go to the Minimum SDK Version and enter 19.

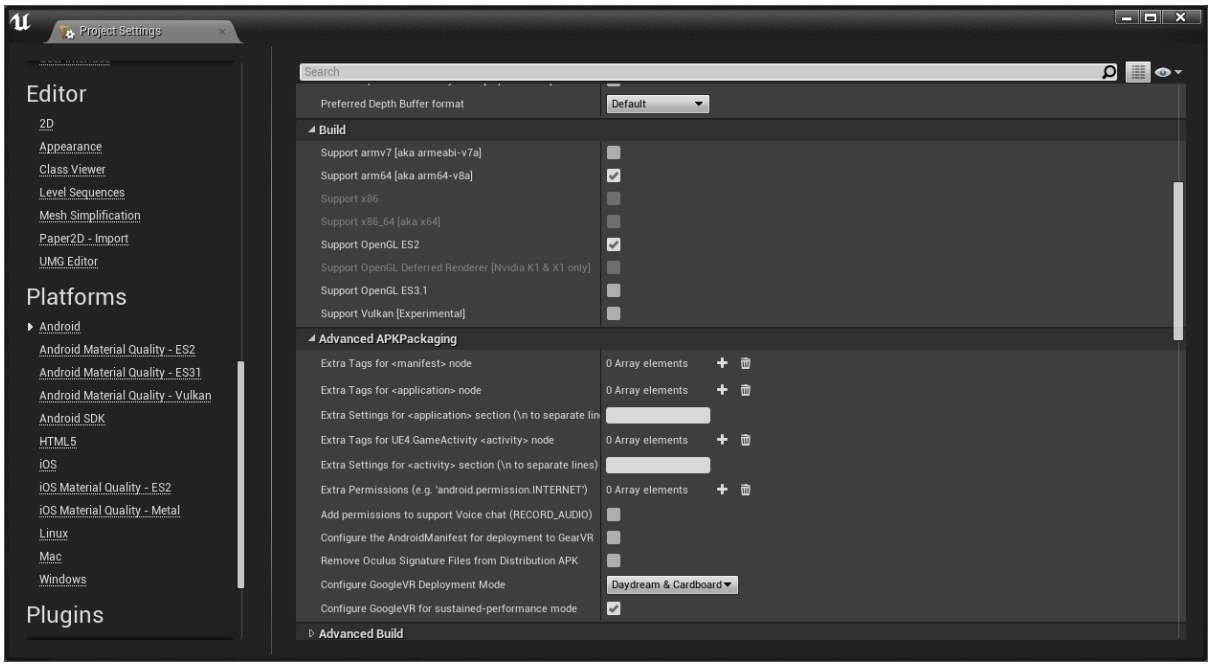
Go to the Target SDK Version and enter 19.



In the Build section below it, ensure that the Support armv7 is checked, as well as Support OpenGL ES2. The armv7 refers to 32-bit devices. If you are targeting 64-bit devices (e.g. Samsung S7/S8 and Google Pixel), you may wish to uncheck the armv7 and instead, check Support arm64. DO NOT check both options.

In the Advanced APK Packaging, go to CONFIGURE GOOGLE VR DEPLOYMENT MODE and select Daydream & Cardboard (the system will auto-detect in this case) --- or you can simply select either DAYDREAM or CARDBOARD depending upon what device you are using with the mobile device.

Check the Configure Google VR for sustained-performance mode.



Under PLATFORMS -> ANDROID SDK, enter the folder locations needed for Unreal Engine to work with the necessary Android SDK, NDK, Apache-Ant and Java components that were previously installed.

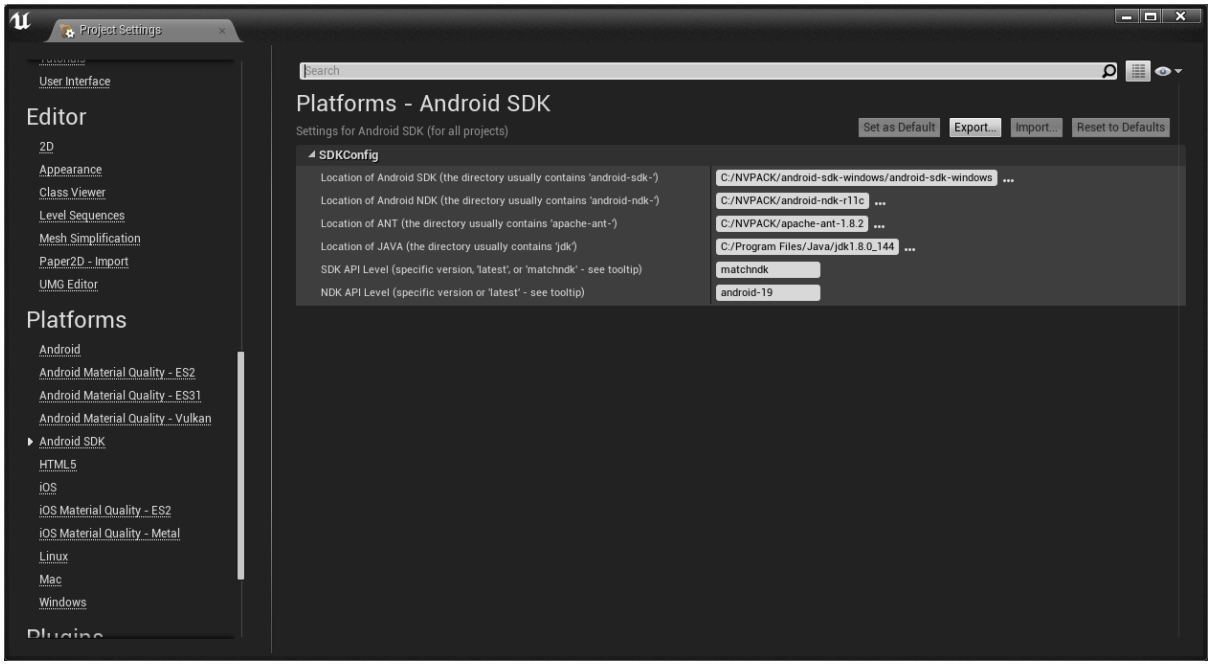
Installations are usually made in the C:/NVPACK directory in Windows.

The Java JDK or Java Development Kit installation may be located at C:/Program Files/Java/jdk1.8.0_144 --- ensure that you have the latest version installed.

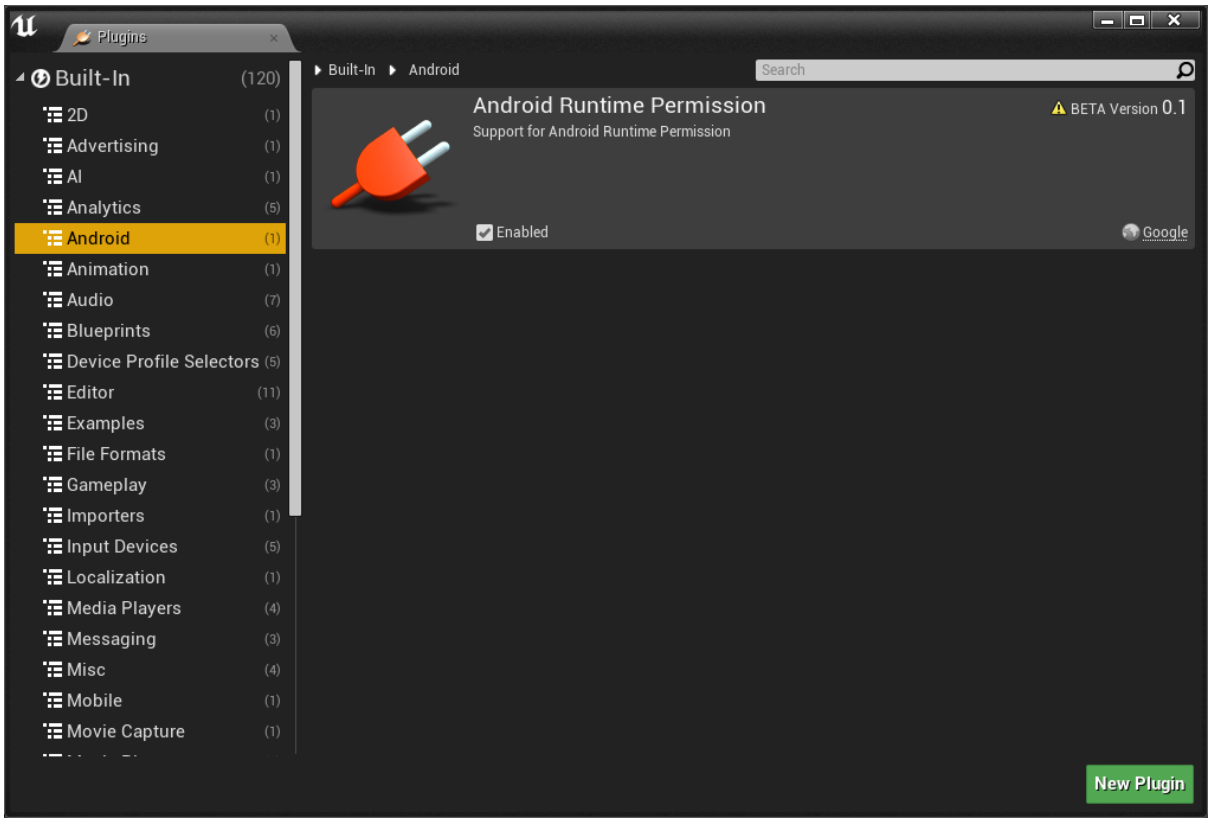
JDK installations can be downloaded from the Oracle website: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

The SDK API Level refers to the target SDK API (which is android-19), but it may be suitable to utilise 'matchndk' at this point. If you have the SDK API-19 installed, then "android-19" can be entered.

Next, enter "android-19" for the NDK API Level.

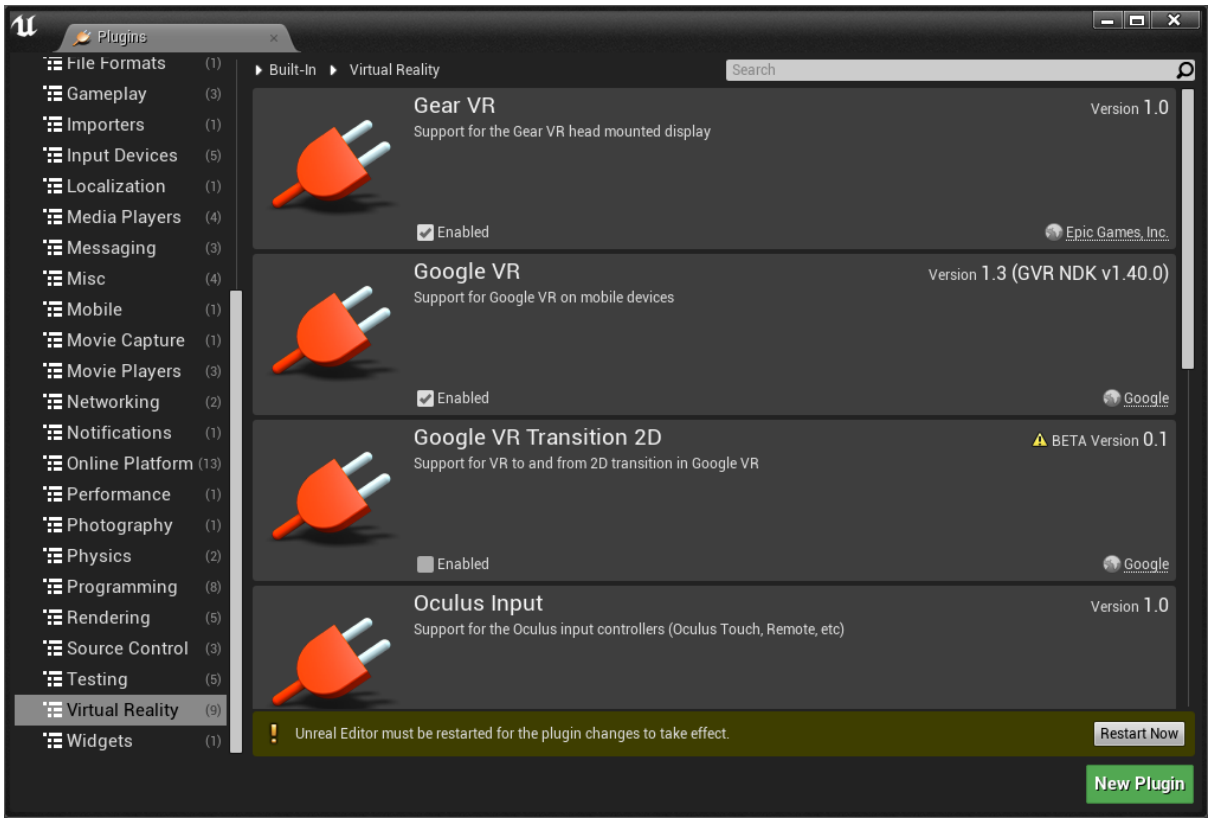


Go to EDIT -> PLUGINS and select ANDROID. You should see that the Android Runtime Permission is enabled.



Next, go to the VIRTUAL REALITY plugins and ensure that GOOGLE VR is enabled. This is needed also for the Google Daydream device – NOTE: The Google Daydream device only supports the Google Pixel, Samsung Galaxy S8 and S8+, Motorola Moto Z, Huawei Mate 9 Pro, ZTE Axon 7 devices.

Restart Unreal Engine and re-check that you have these plugins enabled.



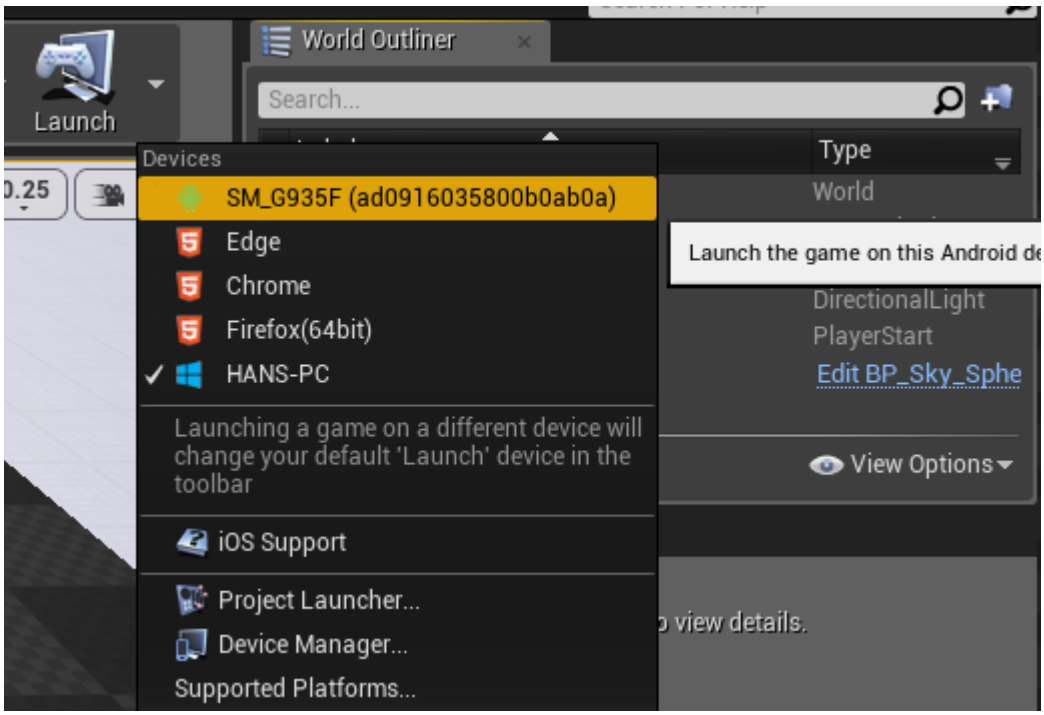
Go to INPUT DEVICES. If you are using the Google Daydream device (which has a controller), you will need to select ENABLED for GOOGLE VR MOTION CONTROLLER. If you are only using Google Cardboard, then leave this alone.

Run your Android device, and ensure that it is set up and responds to the installation of your app generated from Unreal Engine.

For more information on this aspect, go to:
https://docs.unrealengine.com/latest/INT/Platforms/Android/GettingStarted/2/index.html

Connect your Android device and then go back to Unreal Engine. You should be able to see your Android device listed in the LAUNCH menu at the top (refer screen shot example below – note: this example refers to a Samsung S7 smartphone using Android). This will launch the build and create the app directly in the mobile device.

For the first launch, this may take a several minutes.



An alternative method of building the app is to go to FILE -> PACKAGE PROJECT -> ANDROID -> ANDROID (ETC1). This will create the package in a folder in the directory of your choosing (local machine). Developers often use the “Build” folder to store various build versions of the project. Project folders are typically located at C:/Users/<UserName>/Documents/Unreal Projects/.

Once completed, go inside the folder to locate the install .bat file and double-click to run (you will need the Android device attached).

Refer:
https://docs.unrealengine.com/latest/INT/Platforms/Android/GettingStarted/5/index.html

Test your app in the mobile device to ensure that you are getting a basic plane and default sky. There should be a box message indicating “Running <PROJECT NAME> on <MOBILE DEVICE NUMBER>. Click the CANCEL button when finished testing. NOTE: The default project contains a set of two touch interfaces only (this will be changed later).

You should now have a very basic app installed from Unreal Engine and running on your target Android device.

If for some reason, you are not getting the app installed, check that you have the Android SDK API's installed correctly. Run the Android SDK Manager and check what API's are installed – it needs to match what target you have set in Unreal Engine. You should have API-19 installed for Android 4.4.2 (the lowest target as set up in Unreal's project settings) if you are sticking to the settings described in this tutorial.

In addition, check the SDK configuration (locations of libraries) in PLATFORMS -> ANDROID SDK.

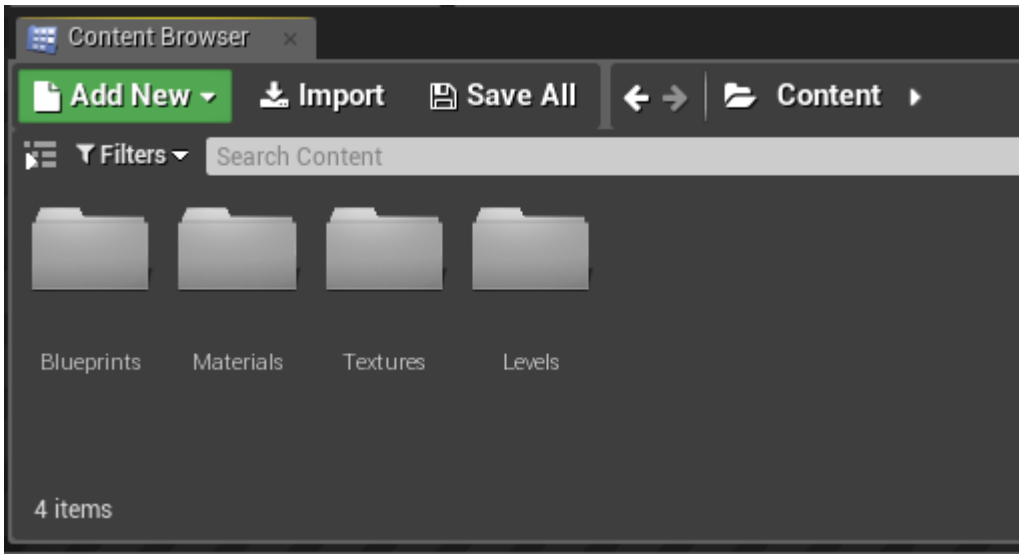
If you are using an Android device with a later version of Android, AND you want to specifically target this device, then you will need the relevant API's installed for it. For example, if you are using an Android device installed with Android 6.0 (Marshmellow), you will need API 23. If you only want to target this device, then set the Unreal settings to "API-23". These target settings are in the PROJECT SETTINGS – SDK API LEVEL and NDK API LEVEL. For the above example, you will need to make them both "API-23".

Organising folders in Unreal Engine

Ensure that you have the CONTENT folder open. RMB-click inside the CONTENT folder and create 4 folders:

- Textures
- Materials
- Blueprints
- Levels (sometimes named "Maps" or "Scenes")

You may wish to add a 5th folder for "Meshes" (or "Geometry"), however this project will not require imported 3D assets.



Additional Project Settings and Plugins

Go to the PROJECT SETTINGS again, and select ENGINE -> INPUT.

Change the MOBILE - DEFAULT TOUCH INTERFACE to CLEAR (this will set this to NONE). This will change the means of interacting in the environment.

NOTE: When packaging for Daydream devices, it is best to use:
FILE -> PACKAGE PROJECT -> ANDROID -> ANDROID (ATC), otherwise for Google Cardboard, use Android (ETC1).

NOTE: If you are moving “Level1” inside the Levels folder, you will need to reset the DEFAULT MAPS in PROJECT -> MAPS & MODES. You may need to use SAVE AS again for the current level and overwrite the moved “Level1”.

3. Setting up a VR camera for basic gaze tracking

NOTE: Majority of this content comes from youtube tutorial (from Strigifo): https://www.youtube.com/watch?v=TeOaThzS51w

Gaze tracking is a means of tracking the viewer's gaze and determining whether an object is being looked at within the VR environment. This is done by tracing out lines from the start point (i.e. the camera that the user is using to see with) to an end point within the 3D space. Unreal Engine uses the Line Trace by Channel and the Line Trace for Objects nodes to help developers work with this.

Open the Blueprints folder (inside Contents) and RMB-click to select BLUEPRINT CLASS -> PAWN.BP. Name it "BP_Pawn".

RMB-click again and select BLUEPRINT CLASS -> ACTOR and name the object "BP_RedCube".

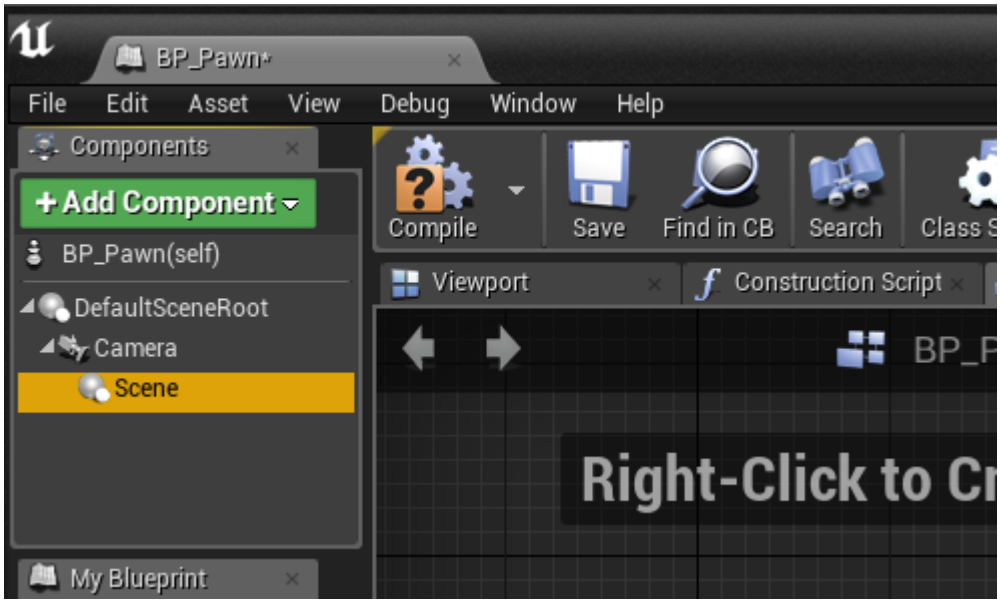
Double-click the BP_Pawn object to launch its editing window and click the EVENT GRAPH tab.

Marque-select the three default nodes and delete.

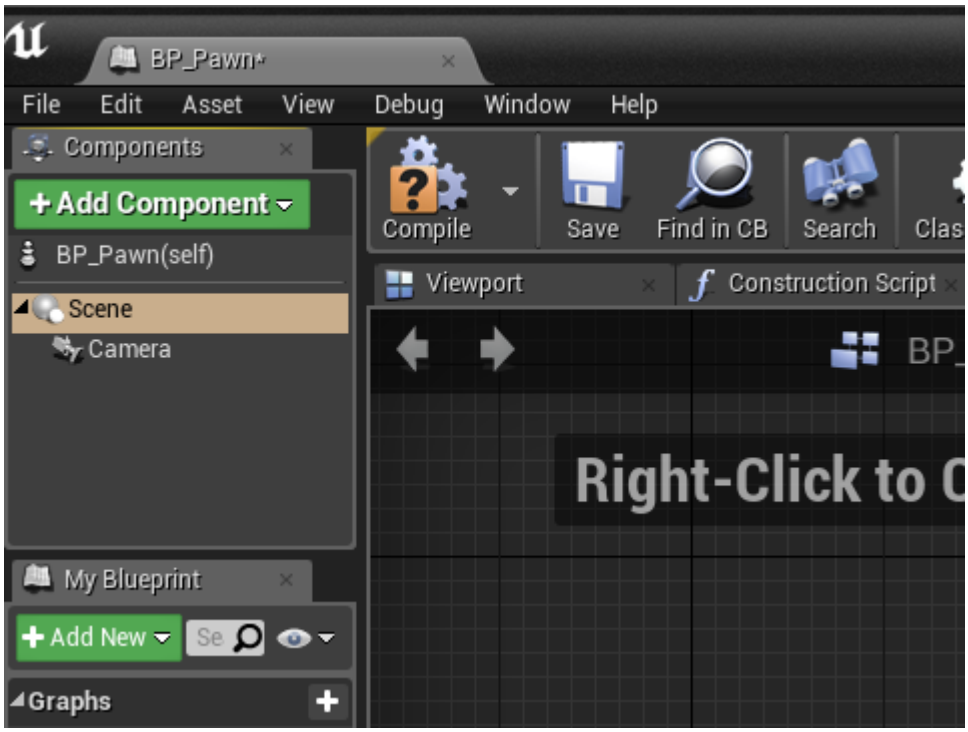
Go to the green + ADD COMPONENT button and in the search field, enter "Camera". Select the resulting "Camera" object.

Click the + ADD COMPONENT button again and in the search field, enter "Scene" and select the utility "Scene" object.

You should have something looking like the screen shot below.



Change the hierarchy to resemble the following screen shot (make the new Scene the new root with the camera a ‘child’):



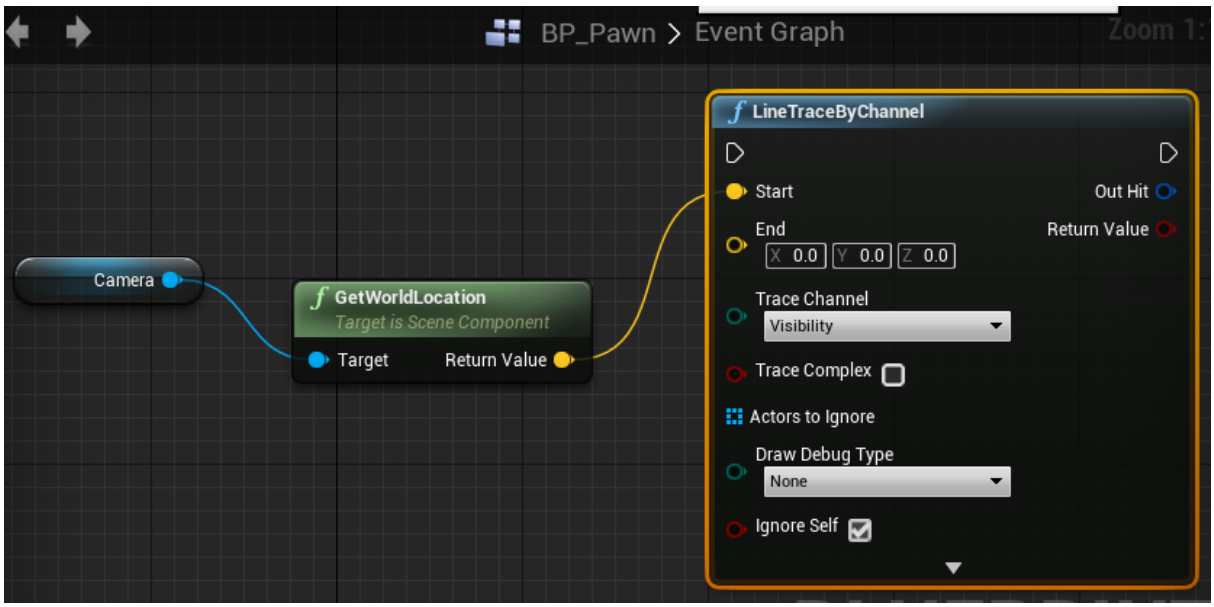
Drag the camera into the event graph window – it will become a node.

Next, RMB-click and search for “LineTraceByChannel” and select. This will create a larger node with several components (Start, End, Trace Channel, Trace Complex, Draw Debug Type etc).

Select the only pin in the Camera node (the circle on the right) and drag it out to an empty space to the right of it. Next enter the search for “GetWorldLocation” and select this node.

Finally drag the RETURN VALUE pin out to the right of the “GetWorldLocation” node and move it to the left of the “LineTraceByChannel” node at the Start pin.

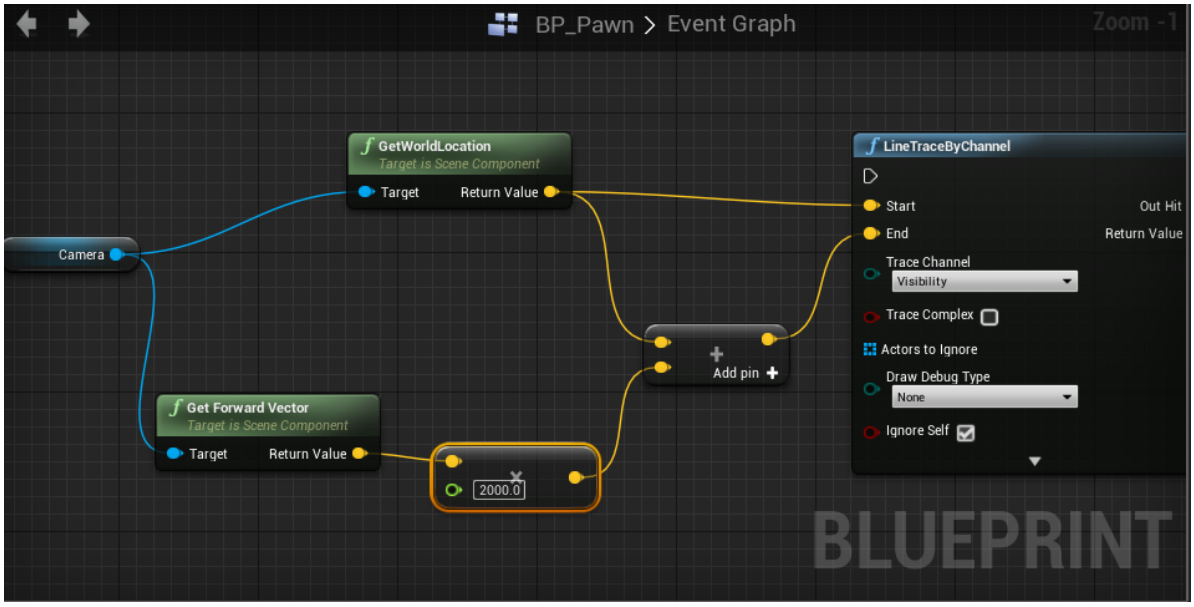
Refer screen shot example below.



Click the IMPORT button and select your 6-image .DDS file.

Next, drag out from the Camera pin to a space on the right. Search for “GetForwardVector” and select. From the Return Value pin, drag out and search for “vector * float” and select. Replace the default zero value with “2000.0”. This value is how far the trace ‘line of sight’ runs out from the camera.

Next, drag out the right pin from “vector * float” and enter “vector + vector” and select. Pull out the RETURN VALUE pin from “GetWorldLocation” node and drag it to the first pin of the “vector + vector” node. Next, take the right output pin from the “vector * vector” node and drag to the second pin on the left of the “vector + vector” node. Finally, drag the only output right pin in the “vector + vector” node to the END pin of the “LineTraceByChannel” node. Everytime you make these connections, you should see a small green tick icon to convey that the connection is made successfully. Refer screen shot example below.



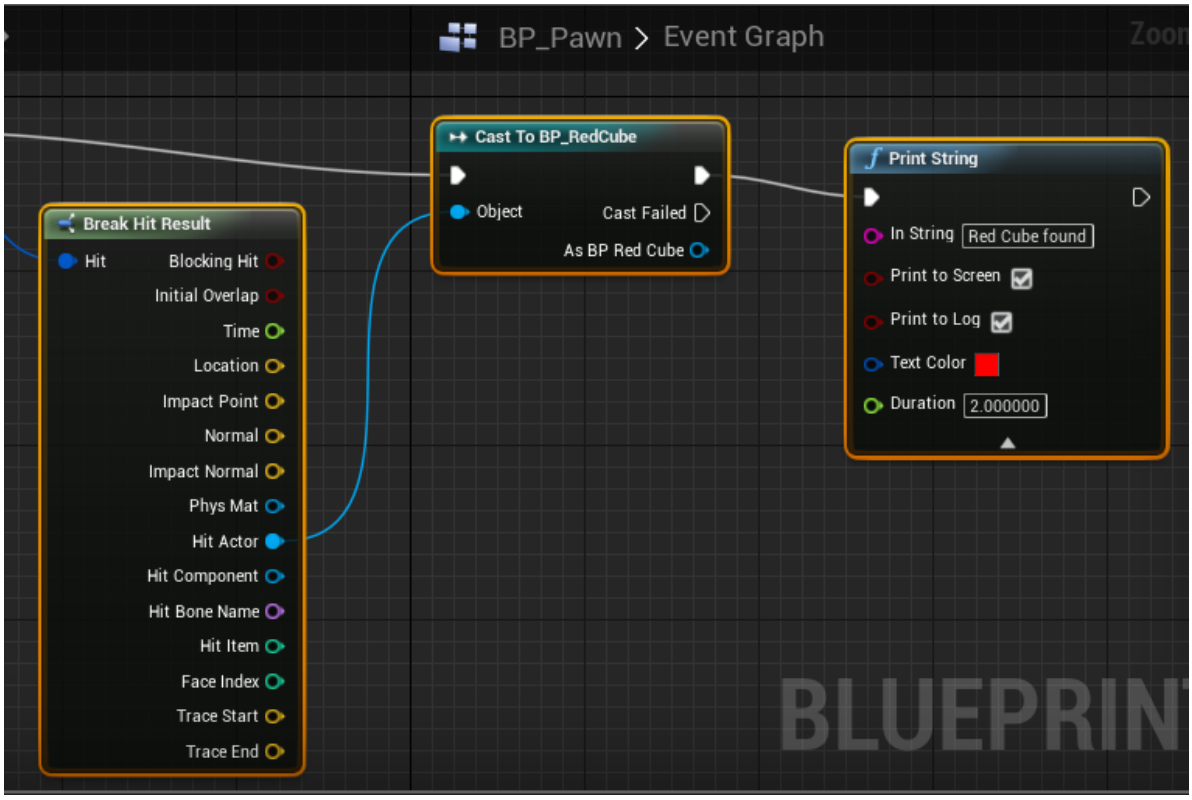
Drag out the OUT HIT pin (right output side of the “LineTraceByChannel” node) and move to an empty space. Enter the search “Break Hit Result” and select.

Drag out the HIT ACTOR pin (right output side of the “Break Hit Result” node) and move to an empty space. Enter the search “Cast To BP_RedCube” and select.

Next, drag out the white arrow pin (top right in the “LineTraceByChannel” node) and move over the top of the pin located at top left of the “Cast To BP_RedCube” node.

Drag out the white arrow (top-right pin) of the “Cast To BP_RedCube” node and search for “Print String” and select. Enter the value “Red Cube found” in the In String text field. Expand this node and in the Duration setting, change to 2.0 (number of seconds the message lasts for). Change the text colour to red.

Click the COMPILE button (top left). You should now have the following added nodes and pins:

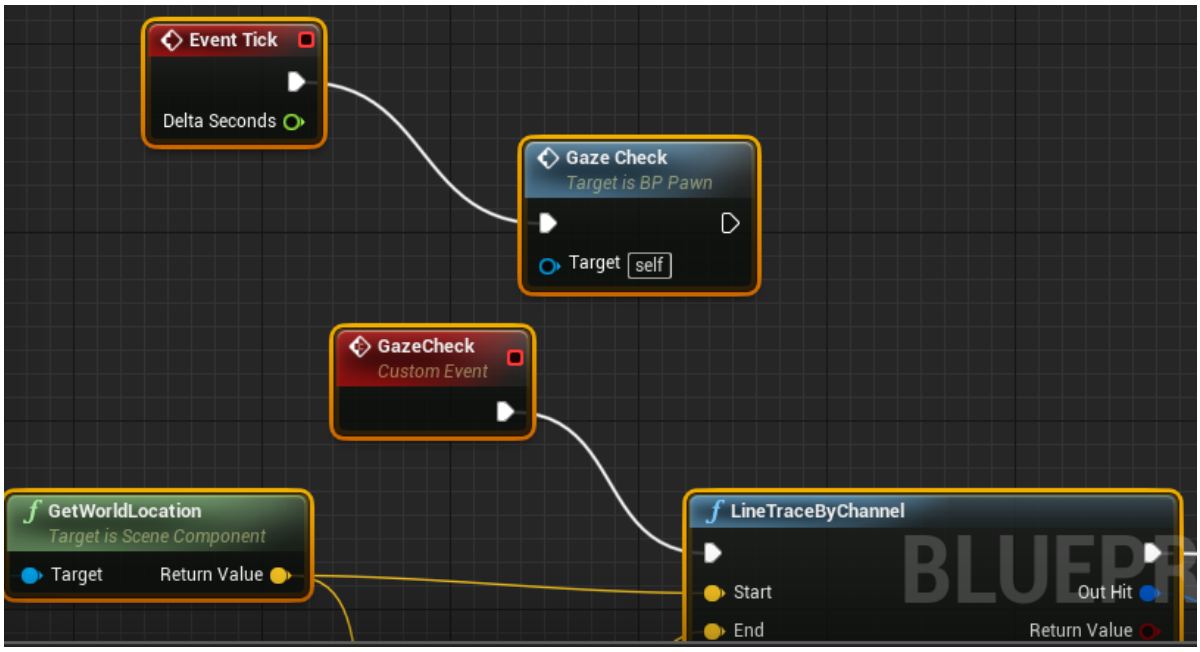


To run this constantly, RMB-click in an empty space above, search for “EVENT TICK” and select. This will be an event which is called in every frame.

RMB-click again in an empty space and search for “Add Custom Event” and select. Change the node’s name to “GazeCheck” and connect the white arrow output pin into the white input pin of the node “LineTraceByChannel”.

Finally, pull the top-right white pin in “EVENT TICK” node and search for “GazeCheck” and select.

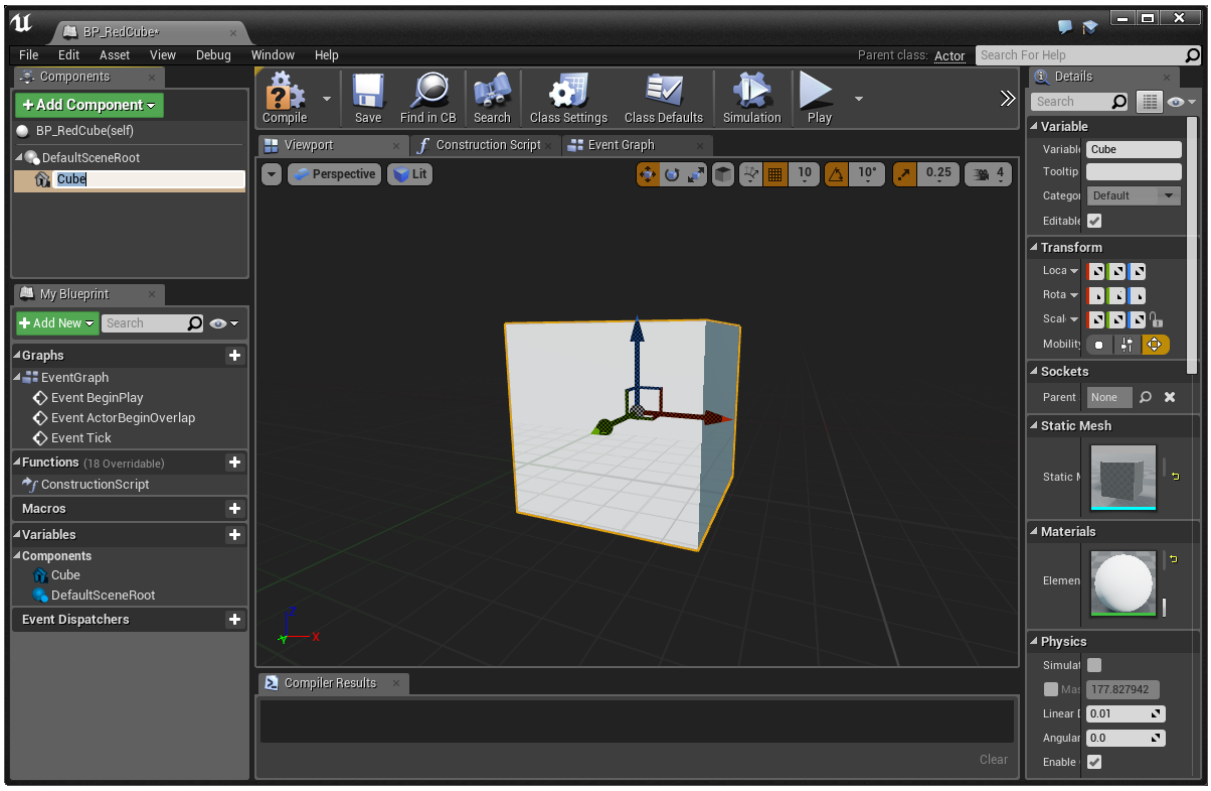
You should now have these added nodes and pin connections similar to the example screen shot below.



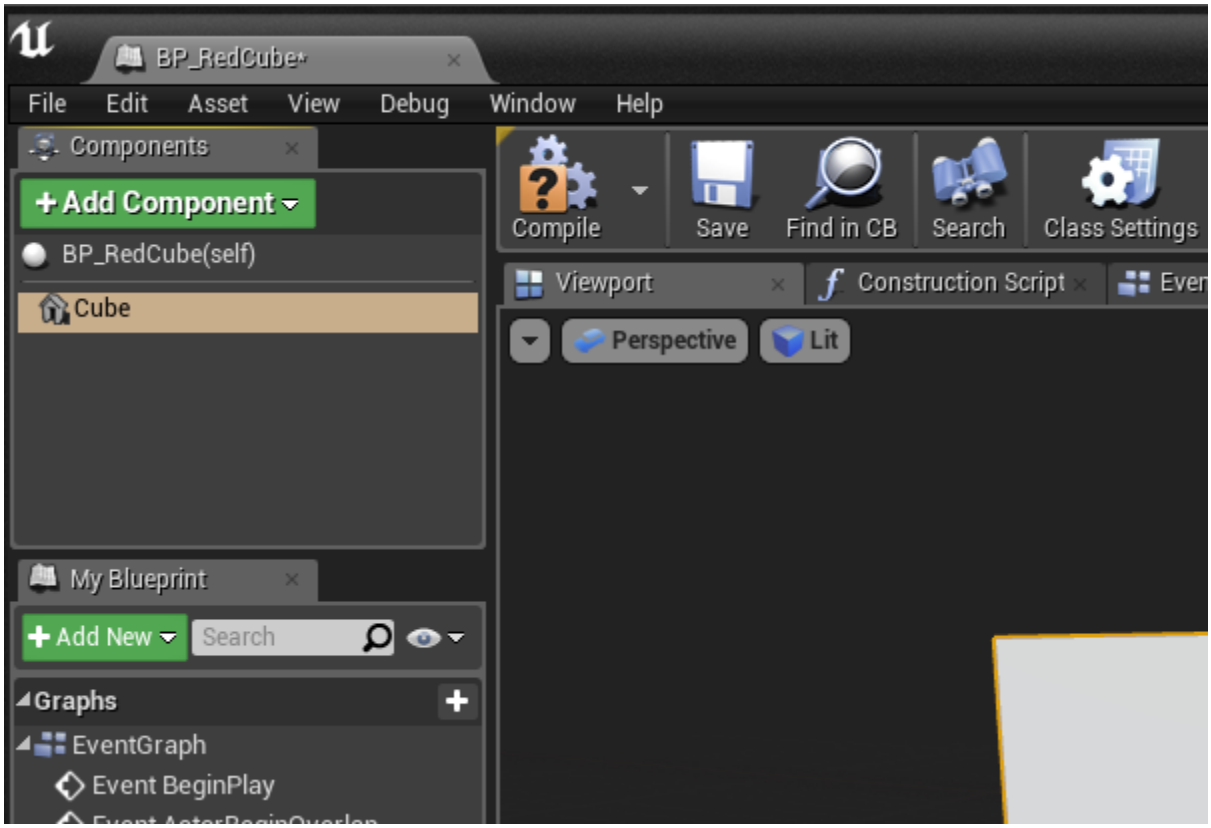
Click COMPILE and SAVE.

Close the BP_Pawn editor window.

Double-click the BP_RedCube blueprint object to open the editor window. Click the green + ADD COMPONENT drop-down menu and select CUBE.

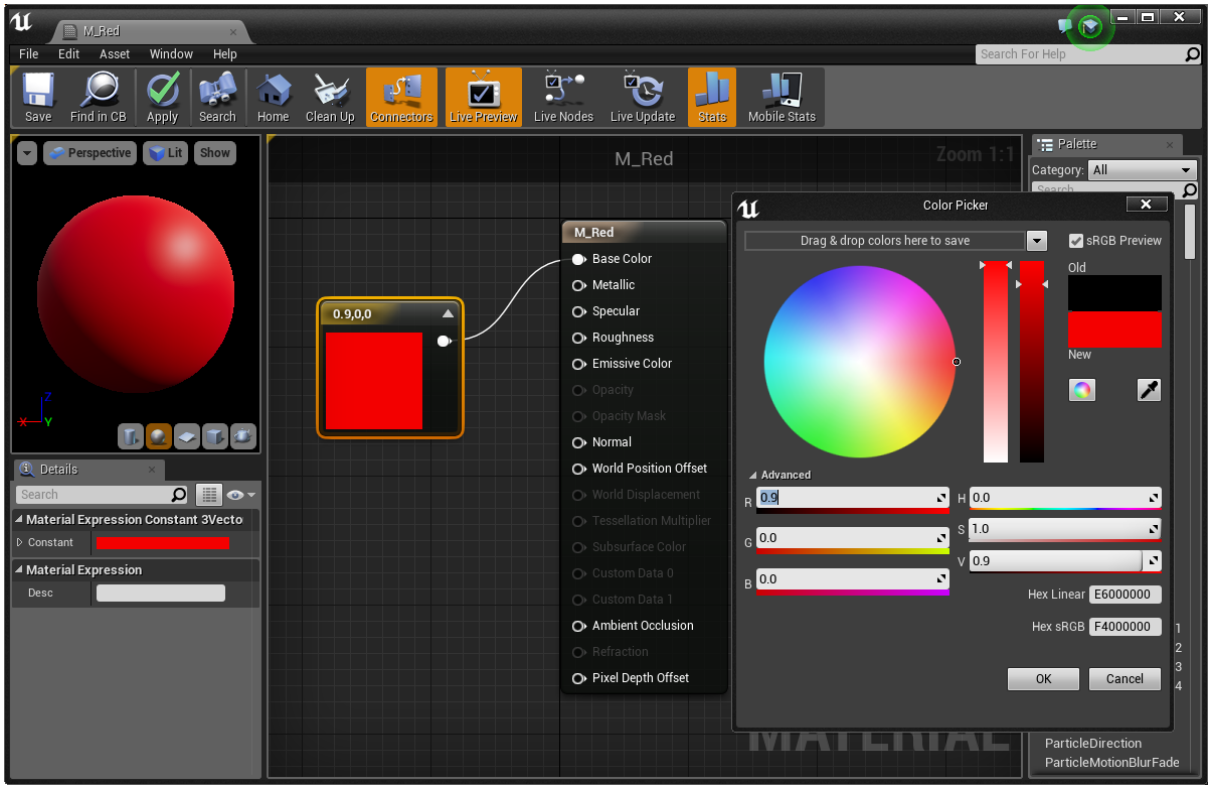


Make the cube the default scene root (i.e. move the cube over the top of the Default Scene Root to remove it --- this appears under the ADD COMPONENT button). Compile and Save.



Close the editor window for the BP_RedCube and open the Materials folder. RMB-Click and select “Material”. Name the new material object “M_Red”. Double-click it to open the editor window.

RMB-click an empty space to the left of the M_Red node and search/select for Constant3Vector. Connect the output pin of this node to the Base Colour input pin of the M_Red node. Double-click inside the space of the Constant3Vector node to bring up the colour picker tool. Select a red colour and click the OK button to accept.

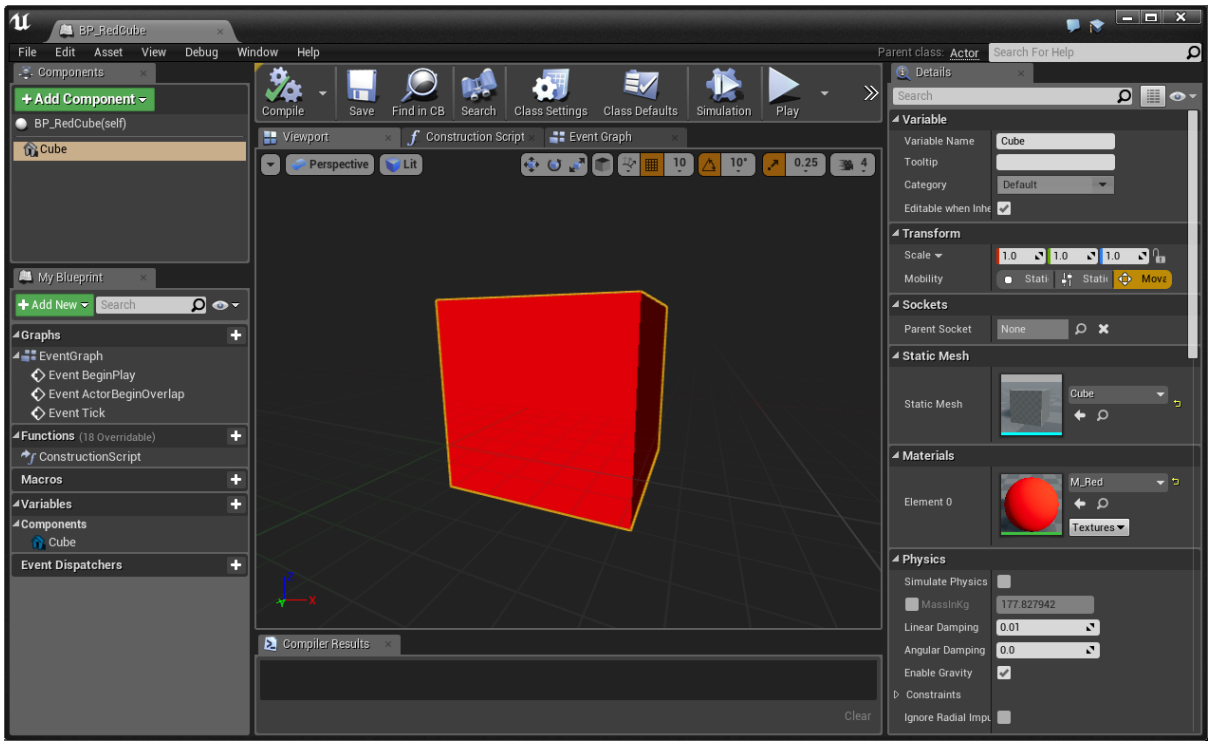


Next, RMB-click an empty space to the left of the M_Red node again – and search/select for Constant. Do this three times and connect one constant node output to the Metallic input of M_Red. Next, connect the second constant node output to the Specular input of M_Red, and finally, connect the third constant node output to the Roughness input of M_Red. This forms the basis of a physically based rendering (PBR) system, which allows more accurate representation of how light interacts with surfaces. For now, leave these constant values at 0. Click APPLY and SAVE. Close the editor window.

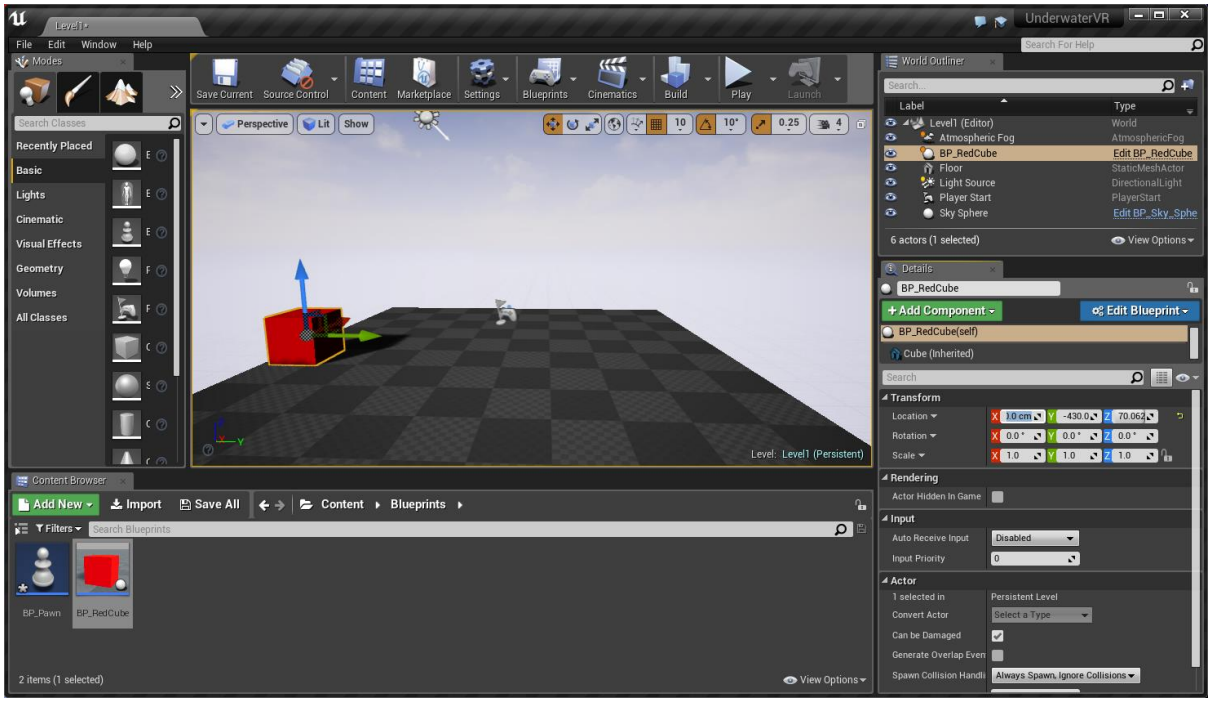
For more information about PBR systems in Unreal Engine, go to:
https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/PhysicallyBased/

Double-click the BP_RedCube blueprint object to open the editor window.

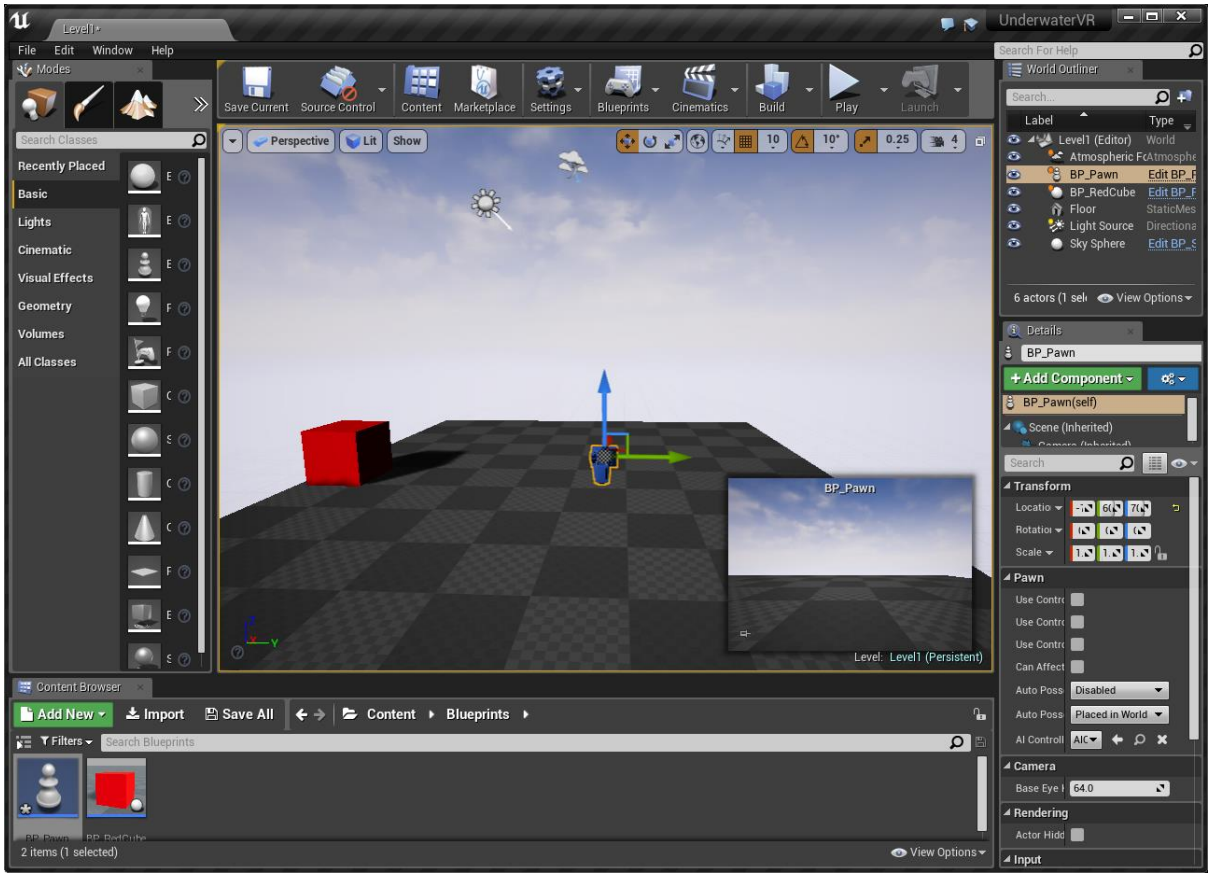
Select the cube and in the Materials section (right side of the window), select the M_Red materials. Compile and Save. Close the editor window.



Next, drag the BP_RedCube on to the scene and position it somewhere on the left of the plane.



Next, select the PLAYER START object in the scene and delete it. Drag the BP_Pawn object to the screen and position it just above the plane – for example: (x, y, z) (-70, 60, 70) – check the small view screen presented when this is selected.

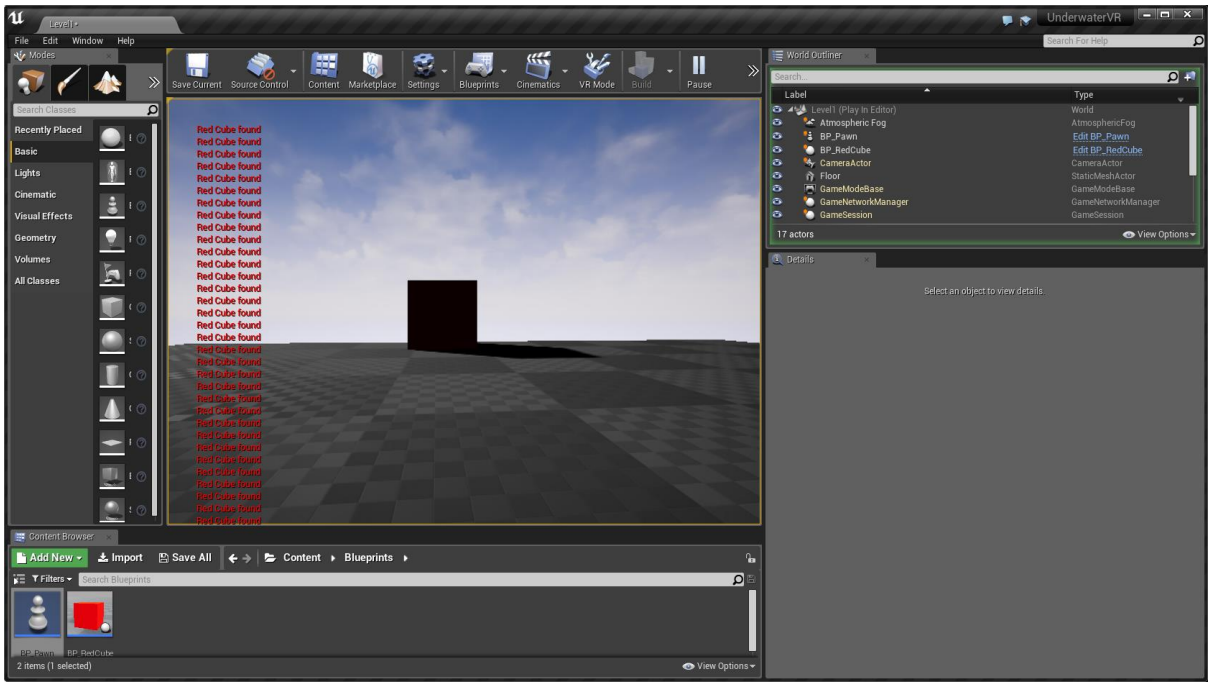


With the BP_Pawn (camera) selected, go to the settings on the right and under the TRANSFORM settings, located the PAWN settings.

Change AUTO POSSESS PLAYER from DISABLED to PLAYER 0.

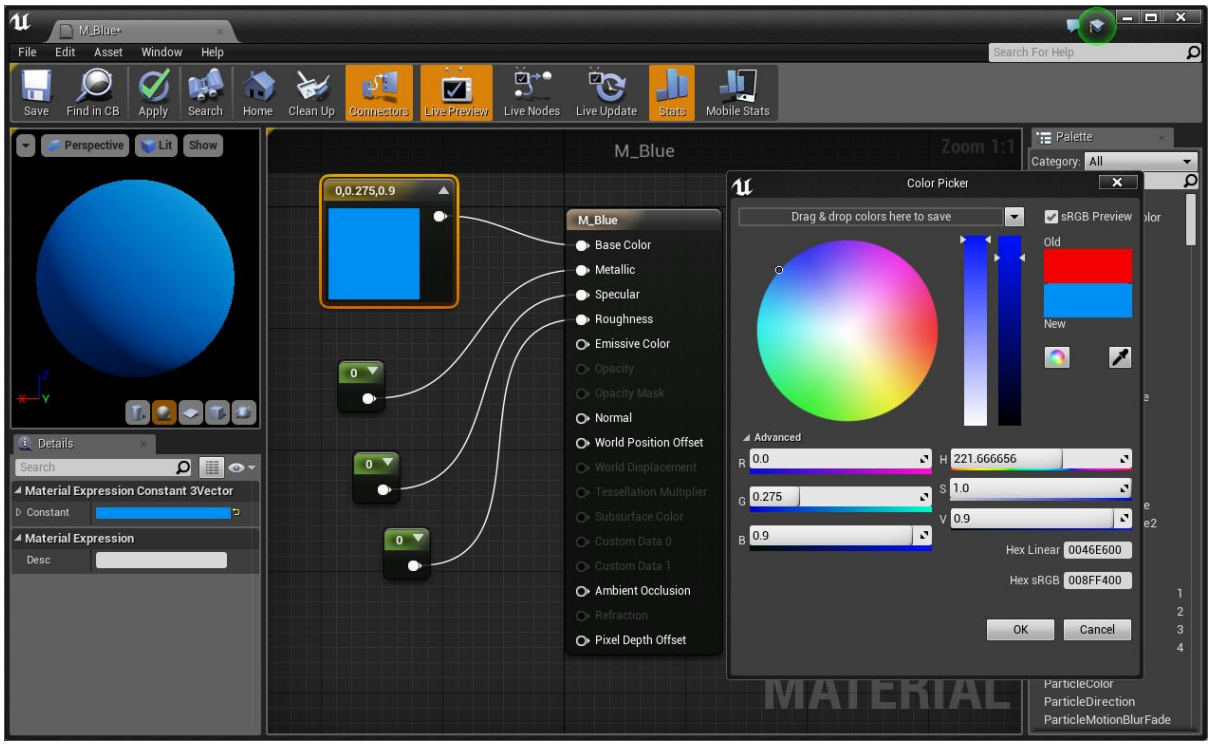
Save the current scene (level) and click the PLAY button to test.

You should have a playable scene where the mouse movement will move the camera. Move the camera so that the red cube (showing as a dark cube as no light is acting on it) appears in the centre of the screen. You should see the red text appear: “Red Cube Found”.

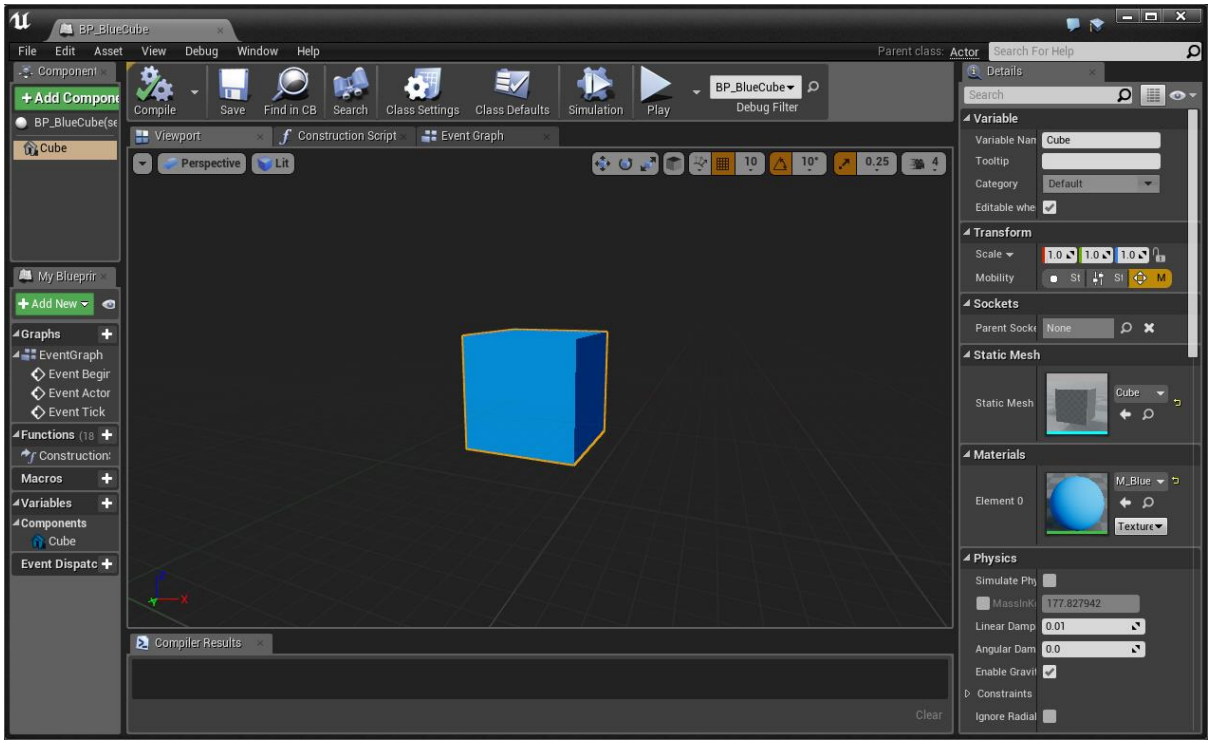


Go to the Blueprints folder in the CONTENT BROWSER and RMB-click the BP_RedCube blueprint and select DUPLICATE. Name the duplicate “BP_BlueCube”.

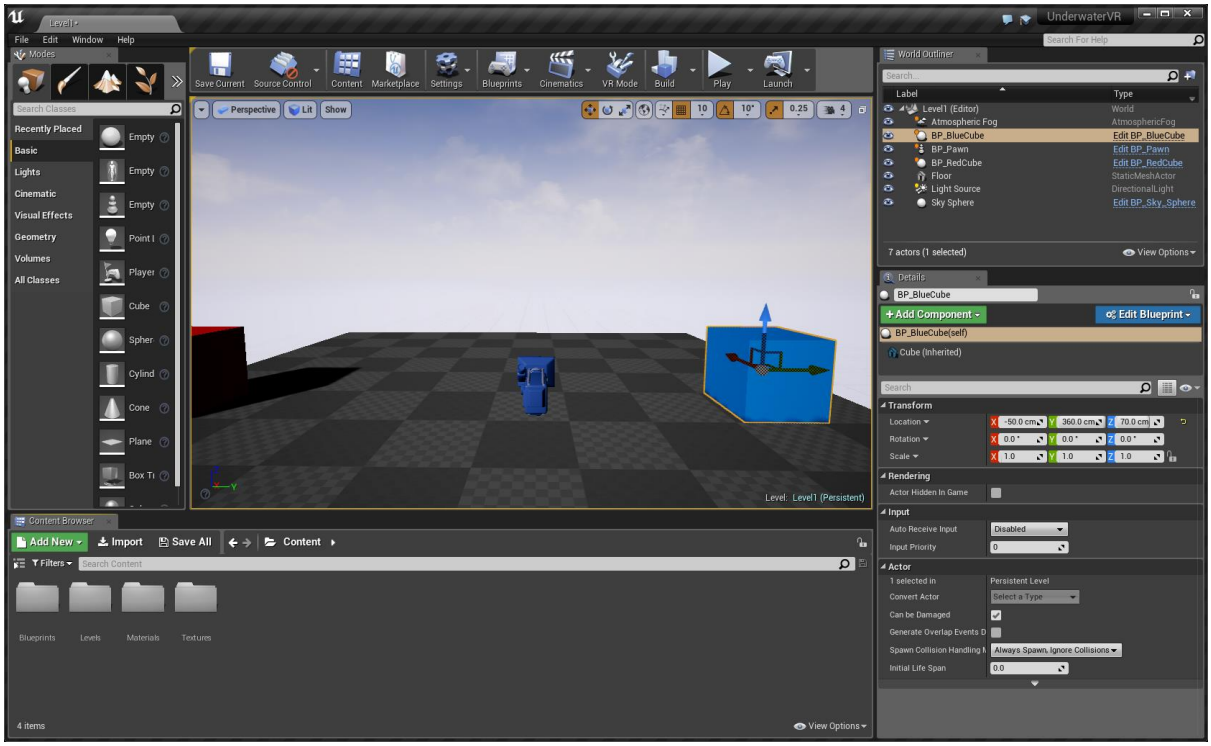
Next, go to the Materials folder and duplicate the M_Red material and name the duplicate “M_Blue”. Change the red base colour to a blue colour. Click the APPLY and SAVE.



Double-click the BP_BlueCube blueprint to open the editor and select the cube in the viewport. Over to the right side, select the M_Blue material for this cube. Click COMPILE and SAVE.



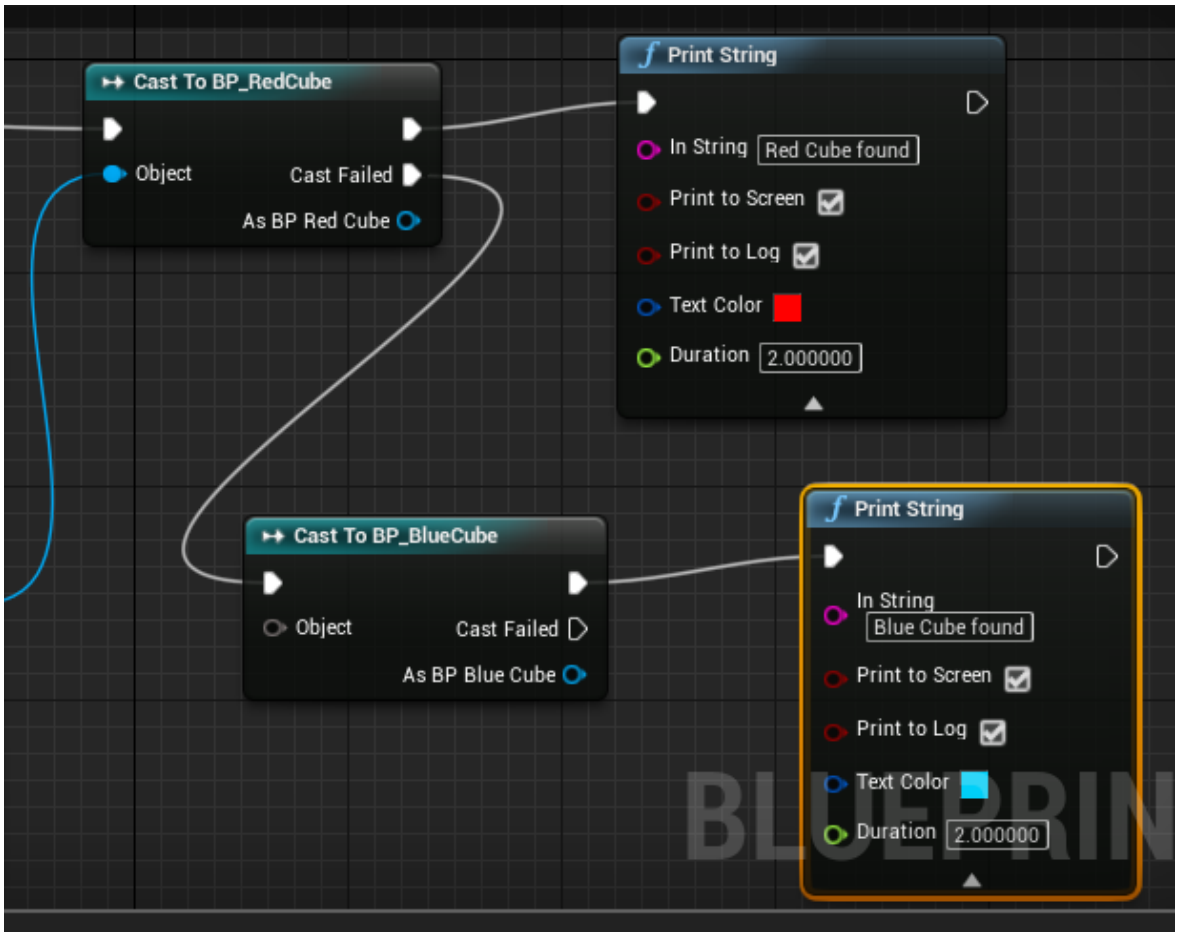
Drag the BP_BlueCube into the scene and position it to the right of the BP_Pawn camera.



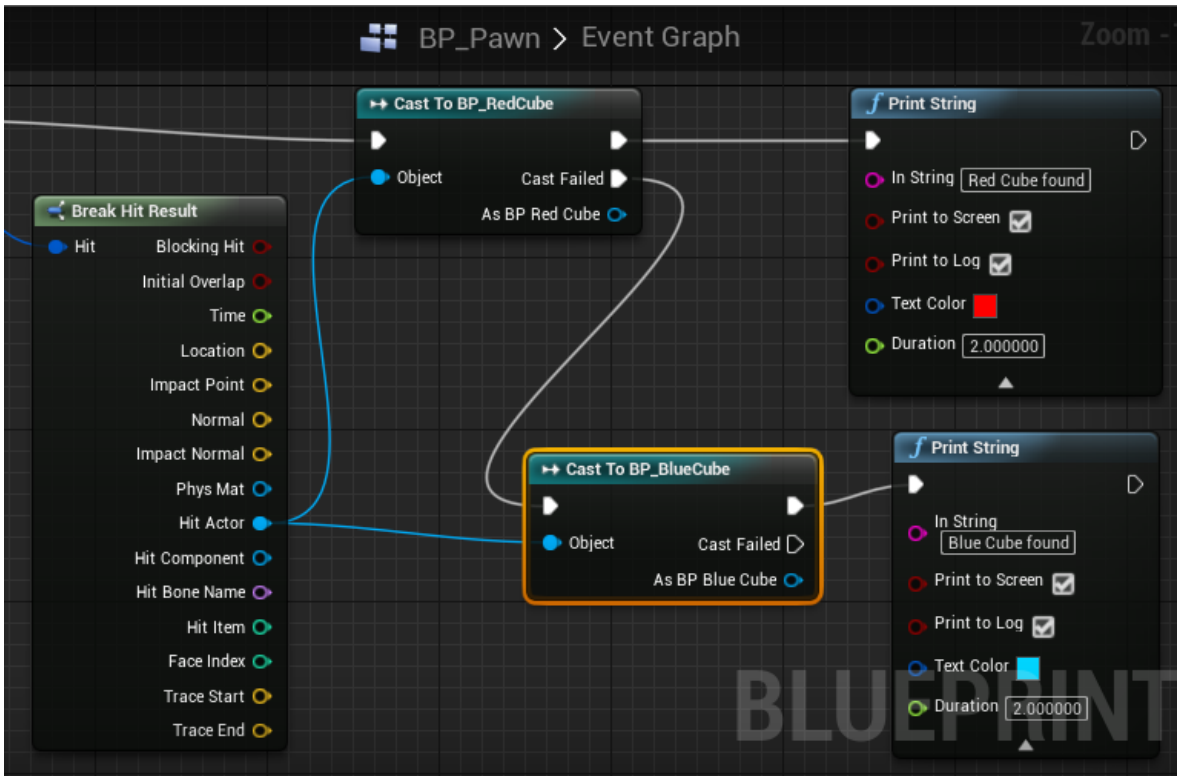
Double-click the BP_Pawn blueprint to open the editor. In the Event Graph, select the CAST FAILED output pin (in the Cast to BP_RedCube node) and pull out to an empty space on its right side.

Search and select CAST TO BP_BLUECUBE.

From the top output pin of the Cast to BP_BlueCube node, extend out and search/select PRINT STRING. Enter for the IN STRING value: “Blue Cube found” and change the text colour to blue.

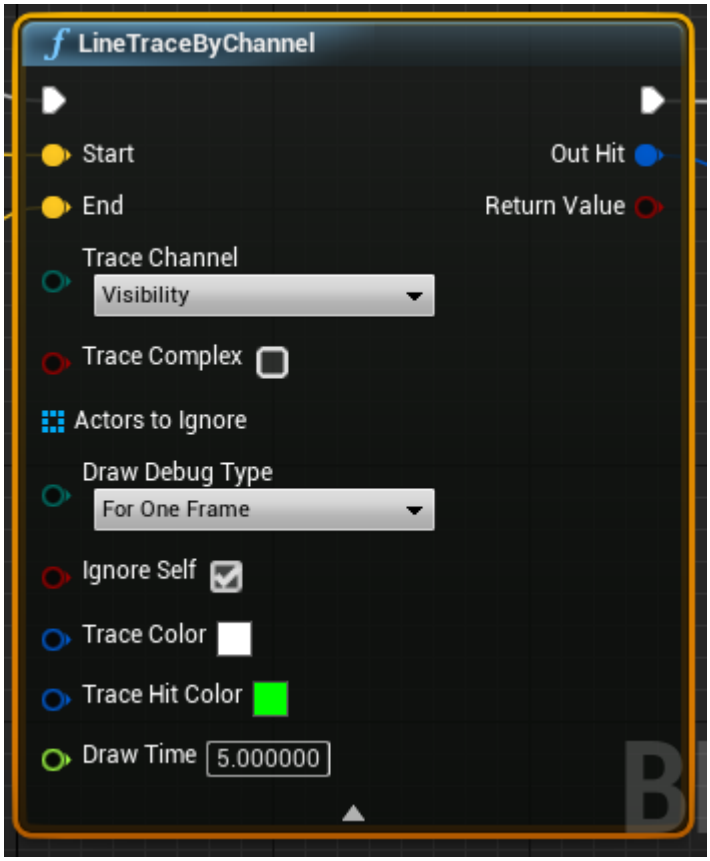


Finally, connect the output HIT ACTOR pin from the Break Hit Result node to the input OBJECT pin of the Cast to BP_BlueCube node. Compile and save and then close the window.

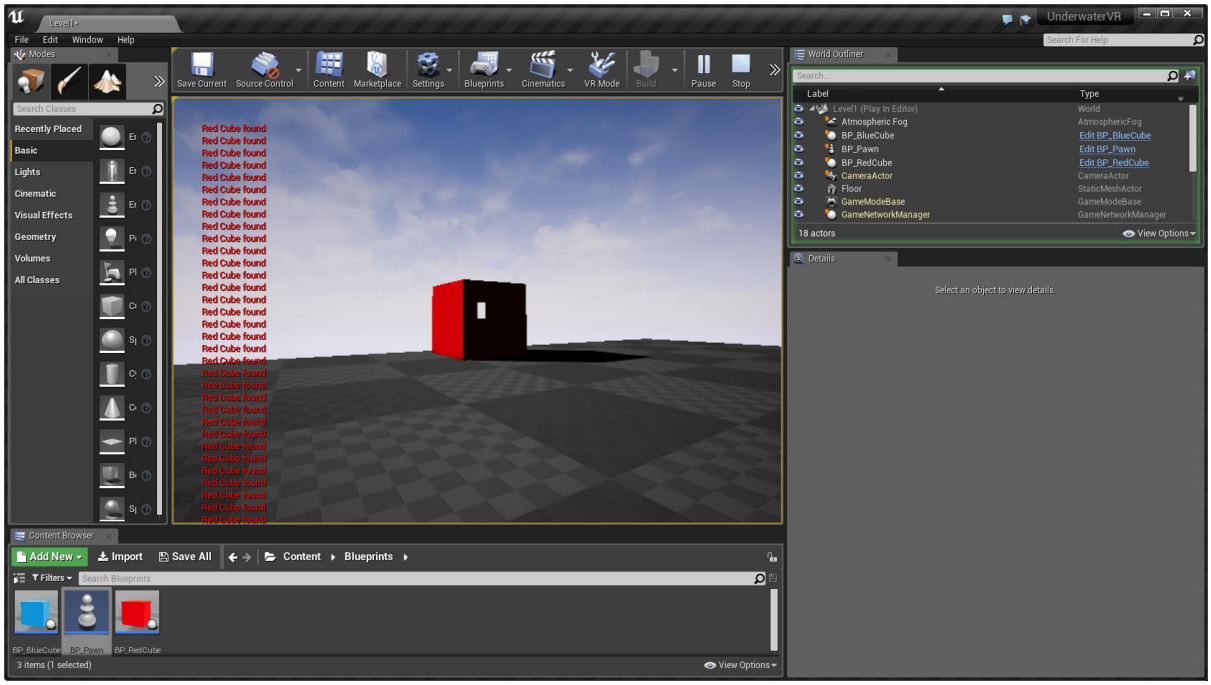


To test, build and deploy the project to your mobile device. You should be able to position the two cubes in the centre of view and see the found text appear.

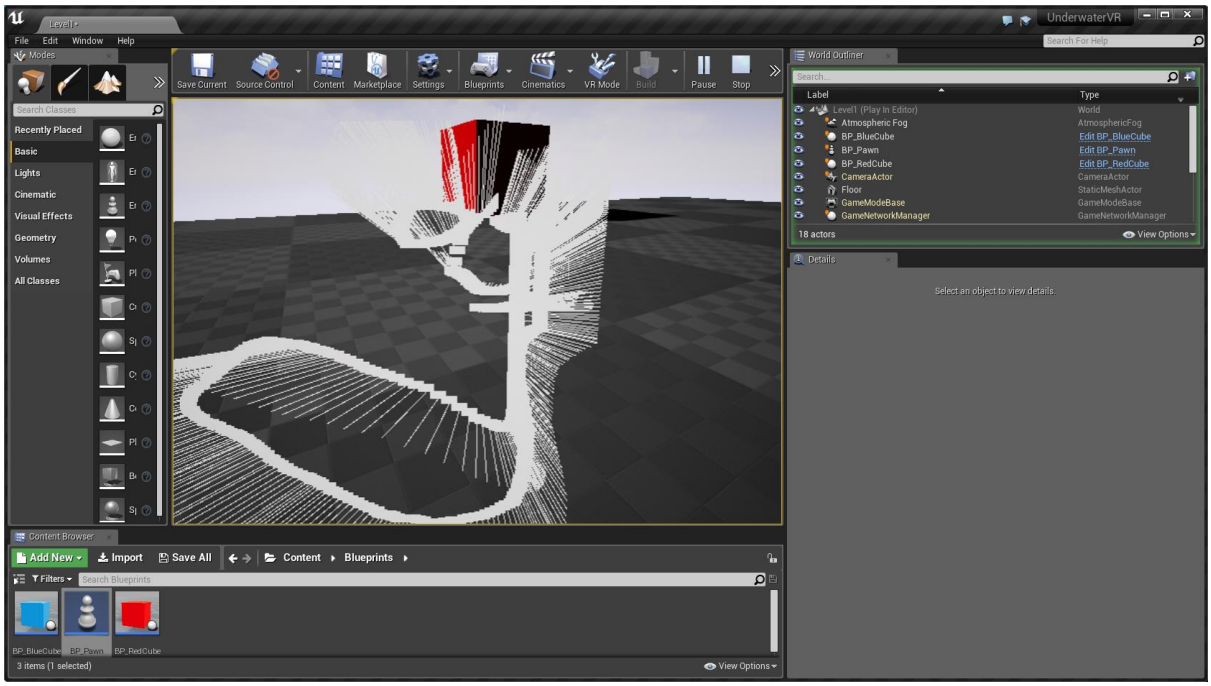
NOTE: If experiencing problems, it is sometimes helpful to activate the DRAW DEBUG TYPE (pin inside the LINE TRACE BY CHANNEL node) and select FOR ONE FRAME (refer Event Graph of the BP_Pawn blueprint). Don't forget to COMPILE and SAVE.



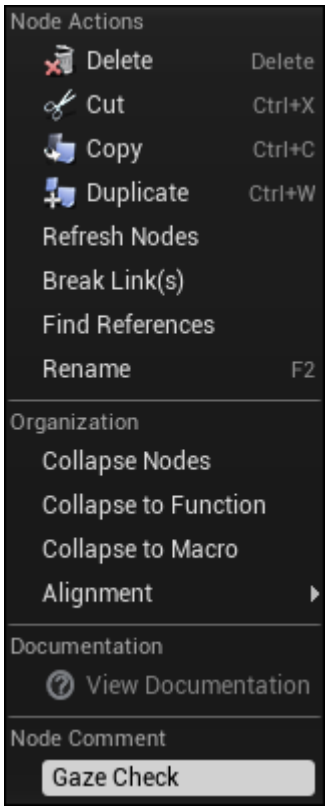
When playing, this will display a small white rectangle, which when placed over one of the cubes will activate the text message.



If changing the DRAW DEBUG TYPE to PERSISTENT, you will get an idea of where you have looked with the camera and the ray trace itself being drawn and remaining on the screen.



If desired, you may wish to create a comment for any of the nodes. The example screen shot below shows a comment “Gaze Check”. To do this, marque-select all of the nodes and then RMB-click to select Node Comment. Enter the comment directly here.



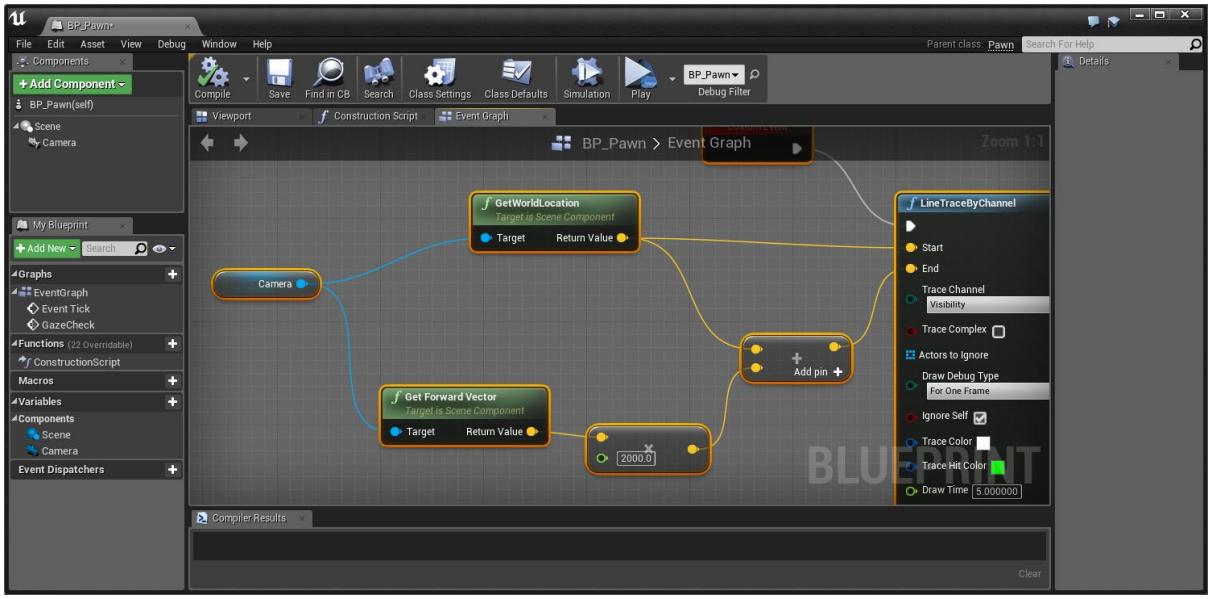
ADDITIONAL NOTES:

The following link is a useful video tutorial aimed at providing background information on look-based interaction (Mitch's VR Lab):
https://www.youtube.com/watch?v=X-7Hu0HzbDU

This video looks at two ways of look-based interaction in Unreal Engine (although there are more):

- **Line Trace by Channel** (used above)
  - type of collision response
- **Line Trace by Object**
  - another type of collision response

It also looks at the basis of using the location node to determine the camera position and the forward vector node (in this example, using 2000 units extended from the camera position) to detect anything that is within this 'line of sight' measurement.



Finally, the video also looks at using an interface (e.g. Interactable) containing three methods:

- OnFocus() --- function which deals with the initial contact of an object that comes into view
- OnBlur() --- function which deals when the object in focus moves out of view (opposite of focus)
- OnActivate() --- function which deals with what the focused object is to do or perform

Blueprints that implement interfaces can use their own customised implementations of the methods used in the interface. For example, an OnActivate() method may contain a particle effect for one blueprint and for another blueprint, it may simply consist of some orientation calculation.

This will be explored later.

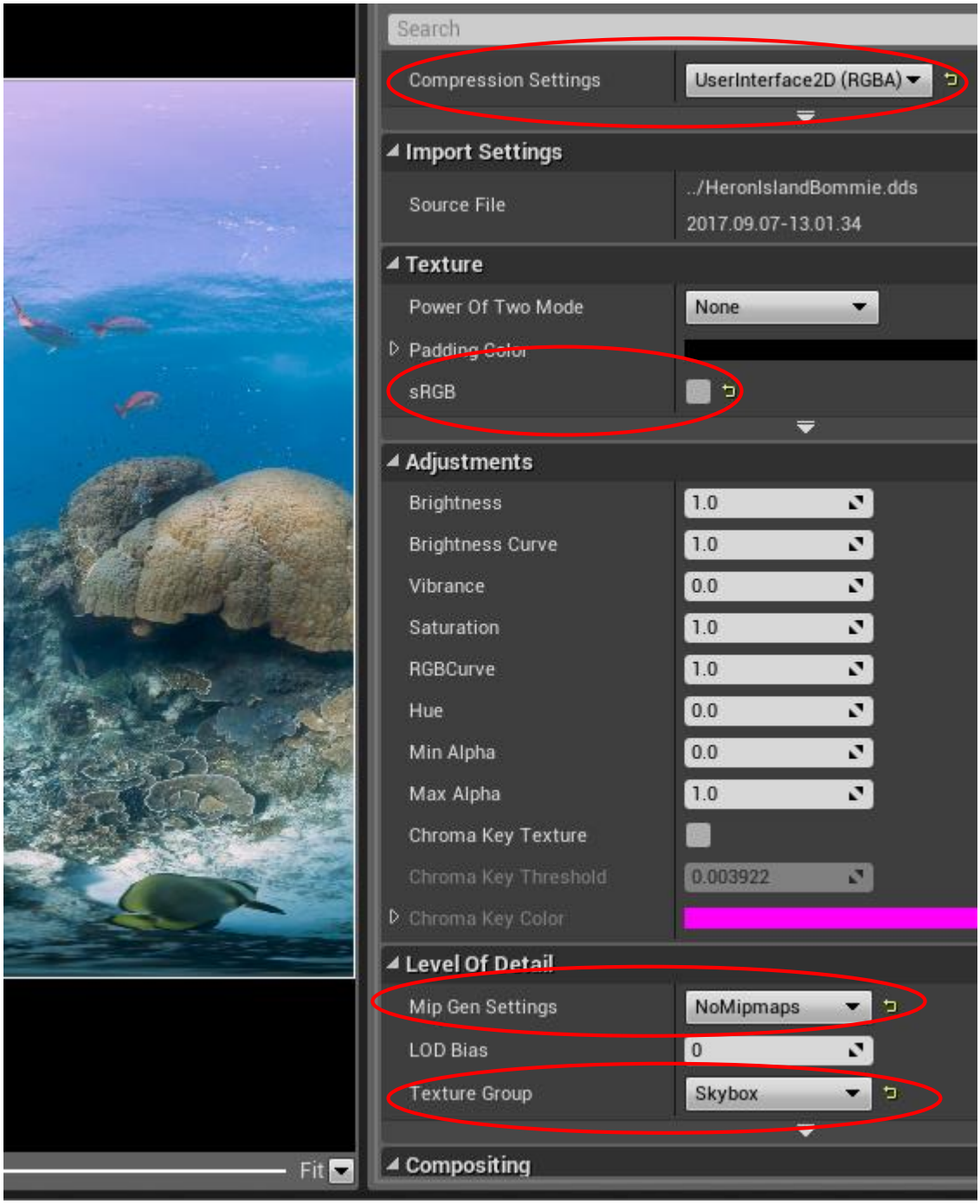
4. Setting up a cubemap in Unreal Engine

Locate your .DDS formatted images into the Textures folder of your Unreal Engine project.

Next, open the Textures folder in your Unreal Engine project and RMB-click to select the import option to set up your .DDS files.

Once imported, double-click to edit the settings and then click SAVE:

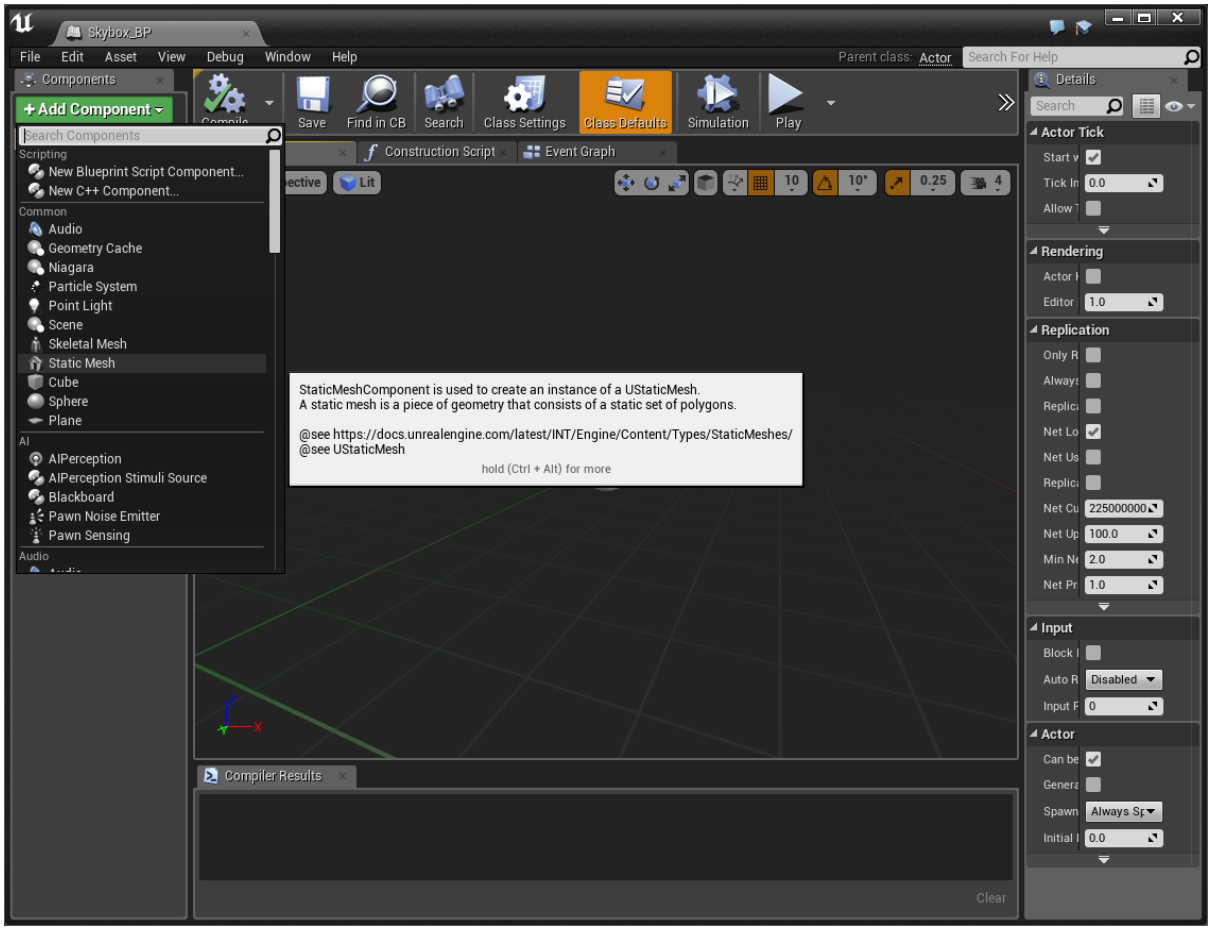
- Compression: Change Compression Settings to User Interface 2D (RGBA)
- Texture: Uncheck sRGB
- Level of Detail: MIP Gen Settings – change to No MIP MAPS
- Texture Group: Change to SKYBOX



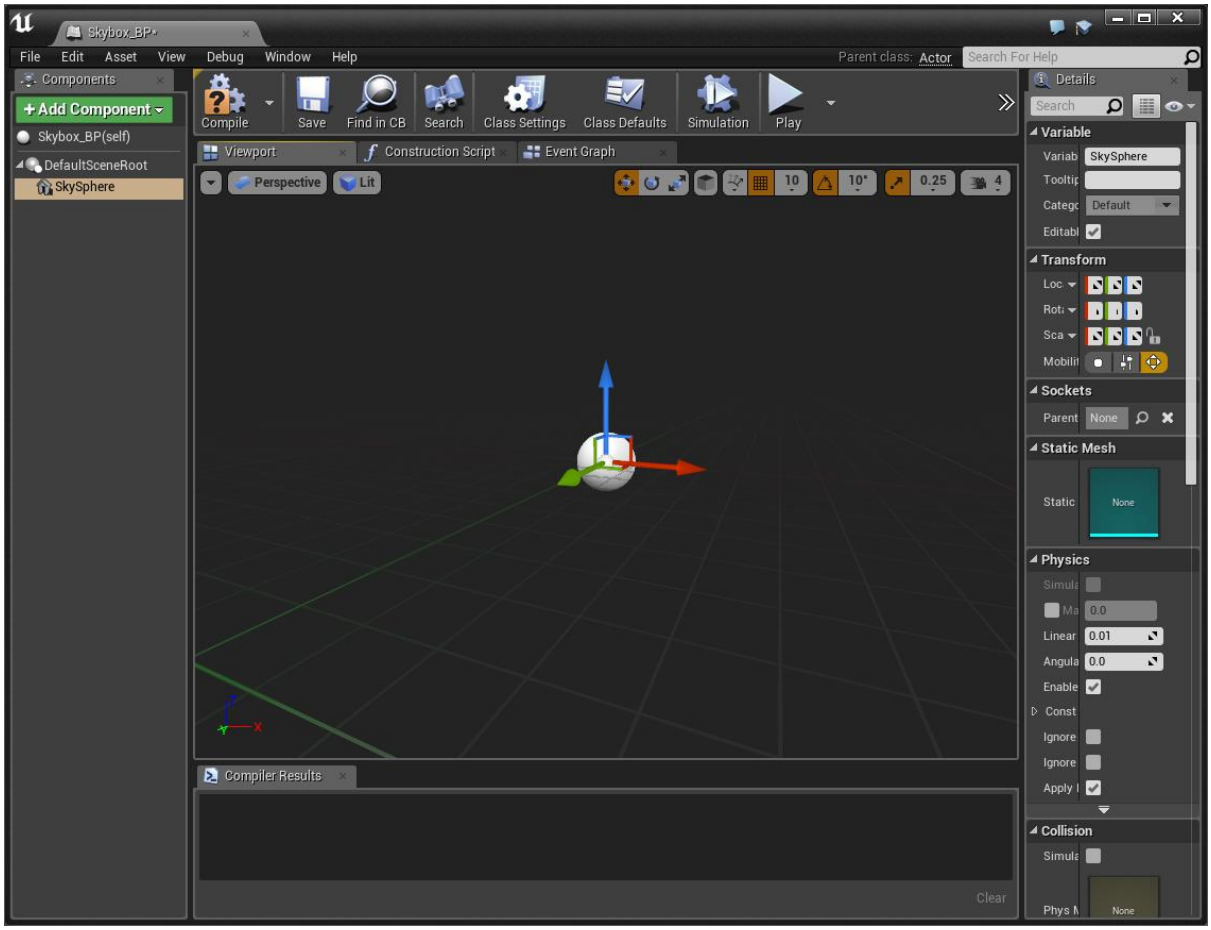
Name your imported .DDS file.

Inside the Blueprints folder, create a new blueprint class (choose Actor) and name it “BP_Skybox1”.

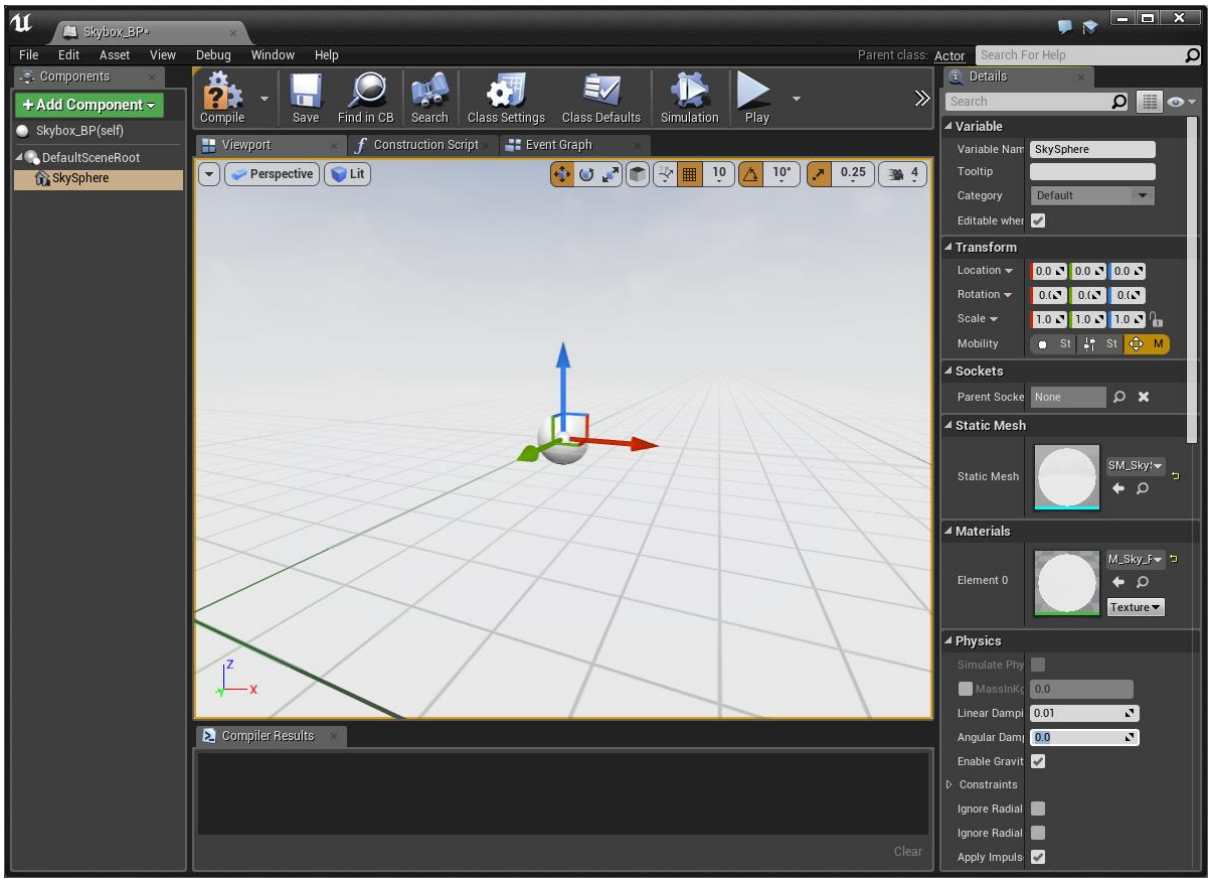
Double-click the BP_Skybox1 to open the editor, and then click the + ADD COMPONENT green button and search/select STATICMESH.



Name it “SkySphere”



In the STATIC MESH section, search and select “SM_SkySphere” … you may need to go to the VIEW OPTIONS and check SHOW ENGINE CONTENT.



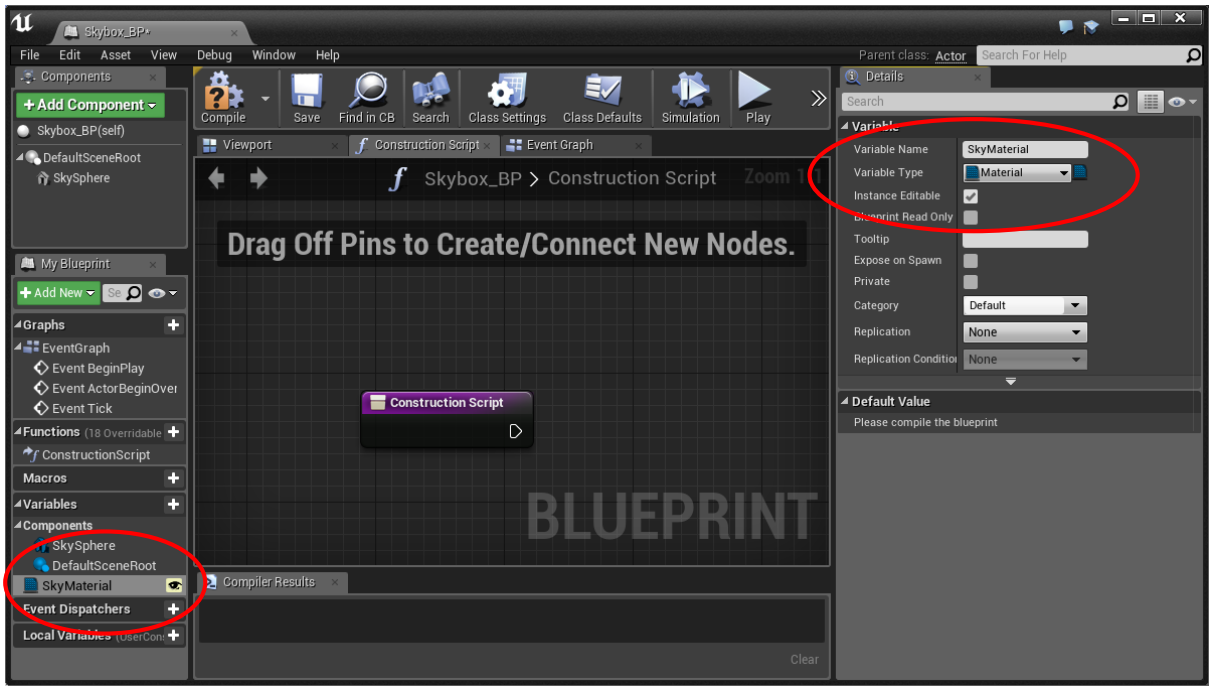
Change COLLISION PRESETS to NO COLLISION.

Next, select the CONSTRUCTION SCRIPT TAB.

Select WINDOW -> MY BLUEPRINT to show elements that belong to the Construction Script node.

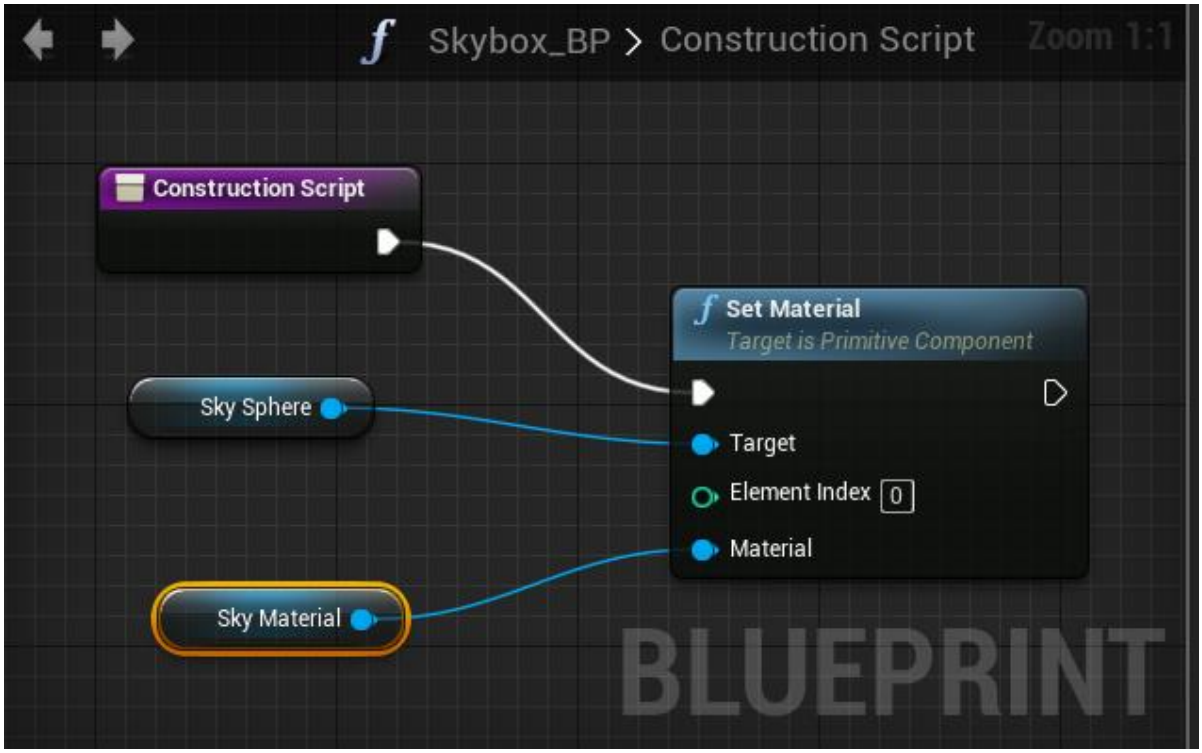
Add a variable and name it “SkyMaterial”.

Change variable type to MATERIAL (under OBJECT TYPES) and check the INSTANCE EDITABLE (to make it public) --- over to the right, under the DETAILS tab.



Select the white arrow (pointing to the right) and drag out the pin – search for “Set Material (Sky Material)” and select. Next, drag out the Material pin from the left input of the SET MATERIAL node and search/select GET SKY MATERIAL.

Select COMPILE and then SAVE and close.



Open the Materials folder, RMB-click inside and select New MATERIAL. Name it “M_Skybox1”.

Double-click to open the edit window.

On the left, change MATERIAL -> SHADING MODEL to UNLIT and check the TWO SIDED checkbox.

RMB-click inside the node editor and choose CONSTANT -> CONSTANT 3 VECTOR.

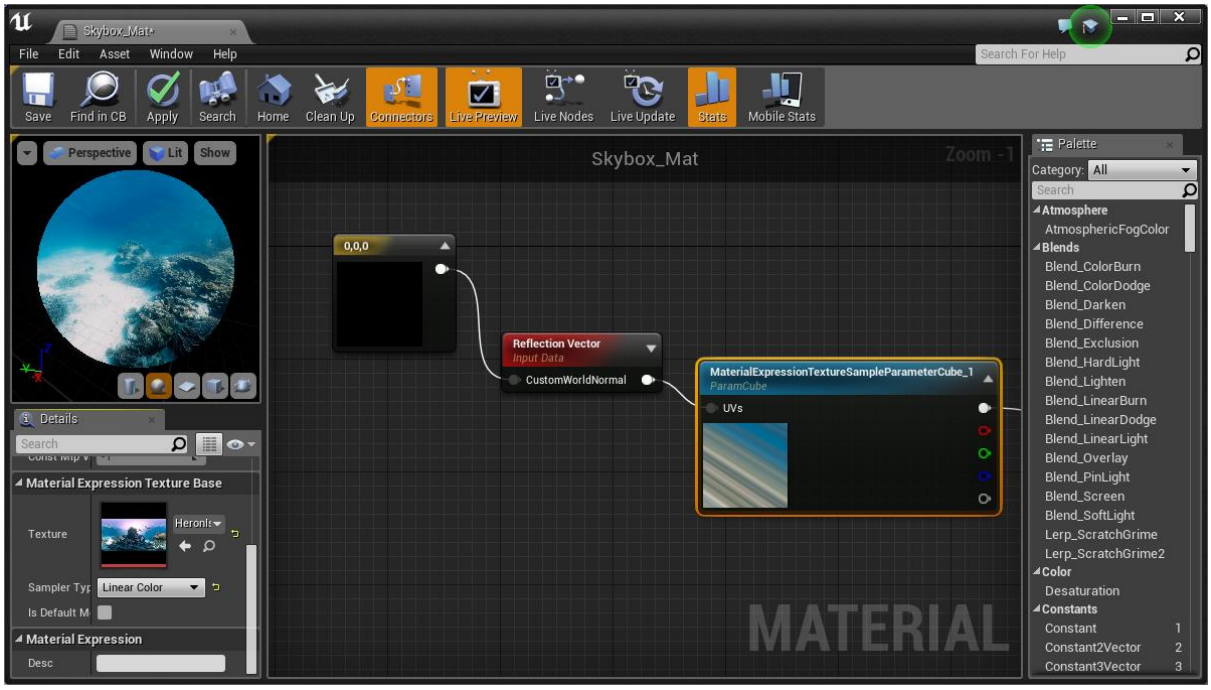
Drag out its only output pin and search/select the REFLECTION VECTOR WS node.

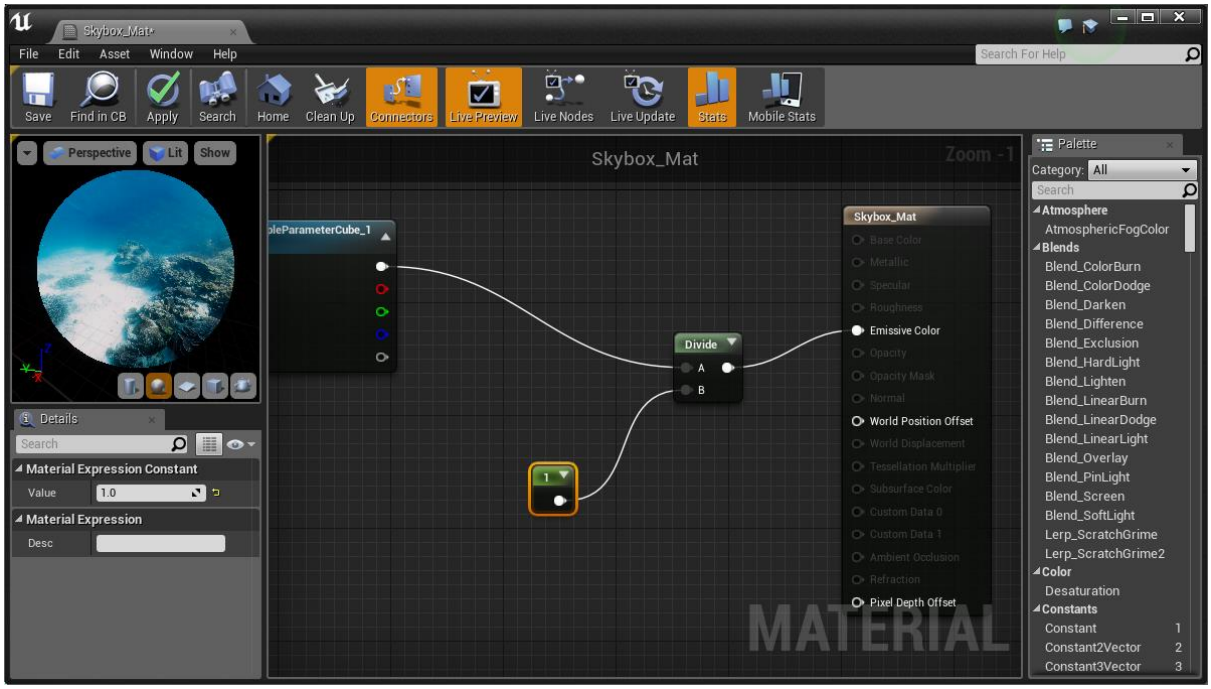
Drag out its only output pin and search to select the TEXTURE SAMPLE PARAMETER CUBE.

It comes loaded with a default texture, which will need changing. Over to the left, under MATERIAL EXPRESSION TEXTURE BASE, click to search for the .DDS texture that you imported into Unreal.

Select the last node created – it will have 5 output pins with different colours. Select the top white pin and drag off to search and select DIVIDE (a Math operation). Take the pin output from B and search/select a single CONSTANT. Change the constant value to 1 (at left – Material Expression Constant). From the output pin of DIVIDE, plug into EMISSIVE COLOR of the M_Skybox node.

You should see the preview in the top left, which is also clickable with the LMB.

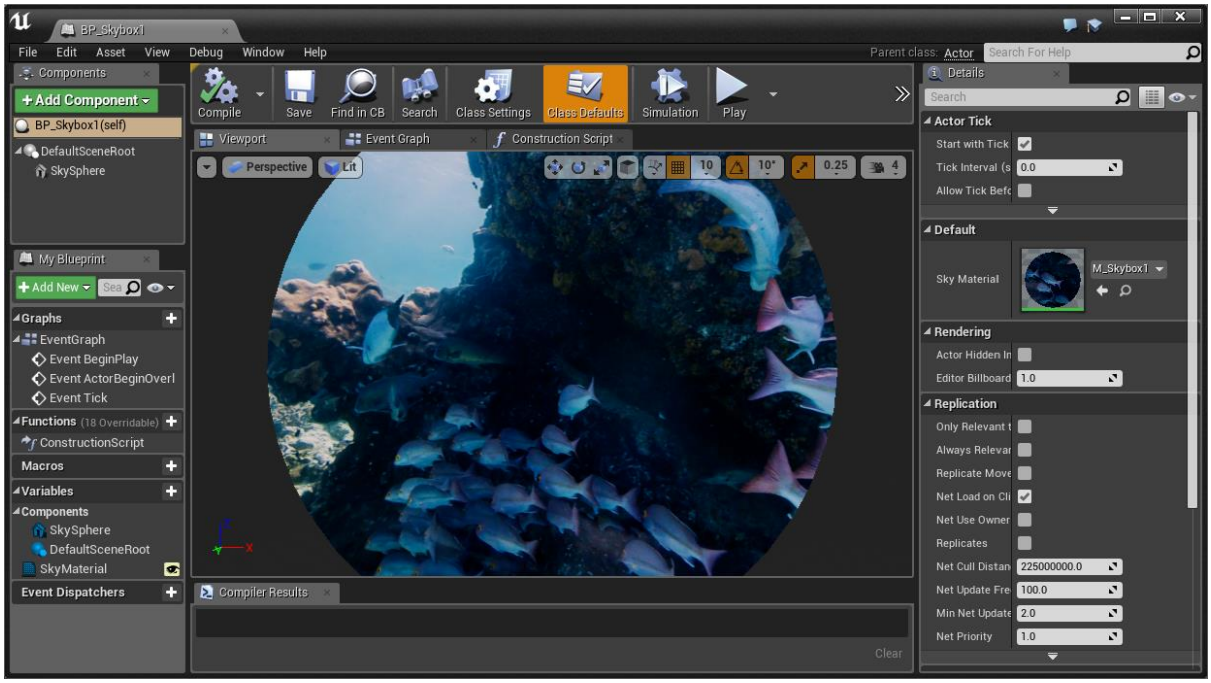




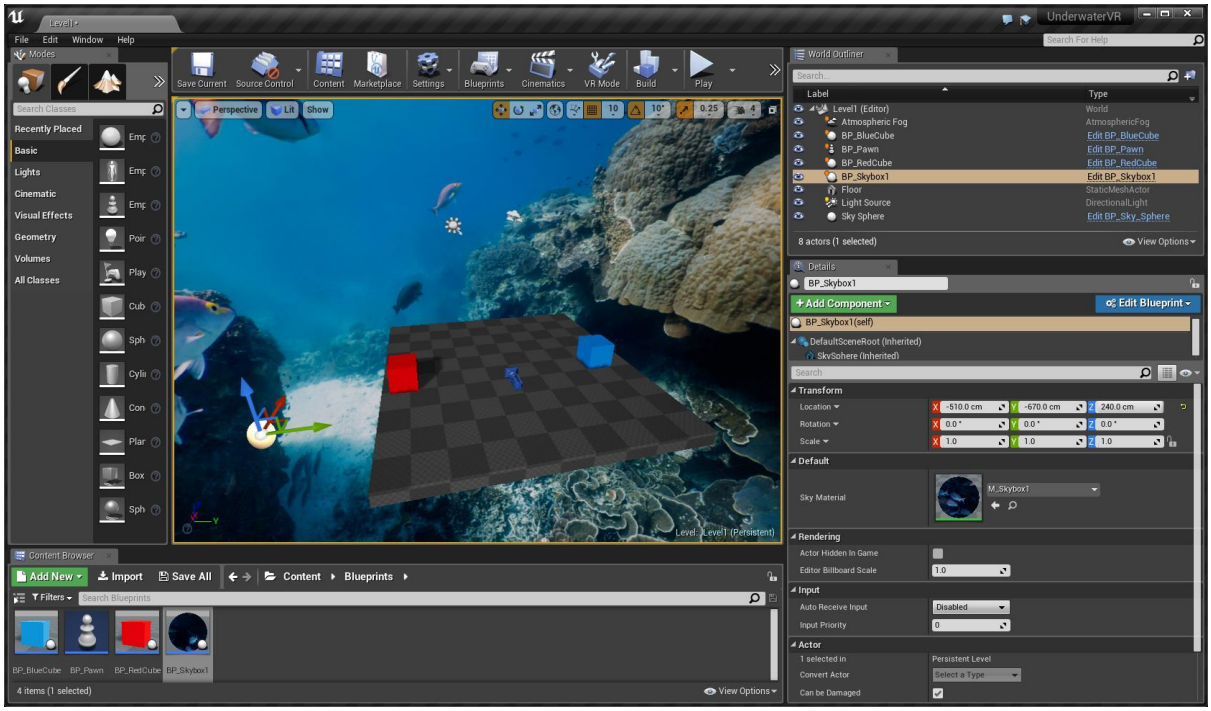
If the appearance of the preview doesn't look right, it may be necessary to check sRGB.

Click APPLY and SAVE and close the editor window.

Finally, double-click the blueprint "BP_Skybox1" and in the Sky Material, search and select the material "M_Skybox1". Compile and Save.

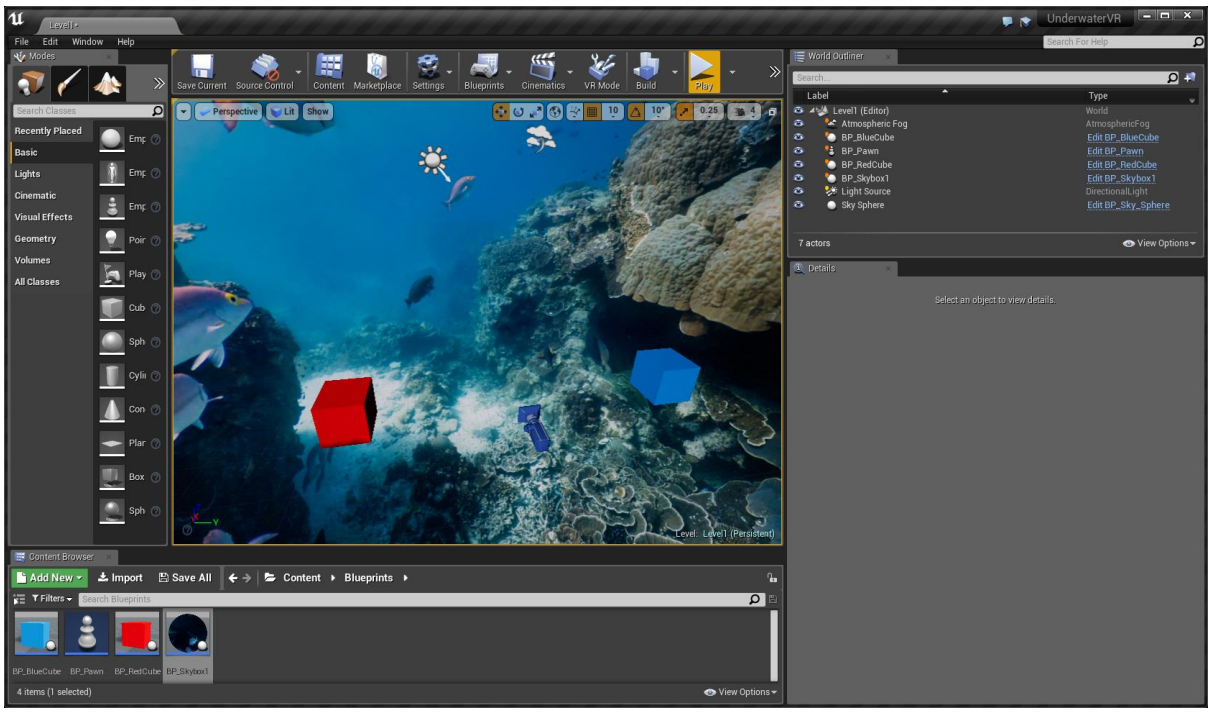


Drag the BP_Skybox1 blueprint into the scene to view.



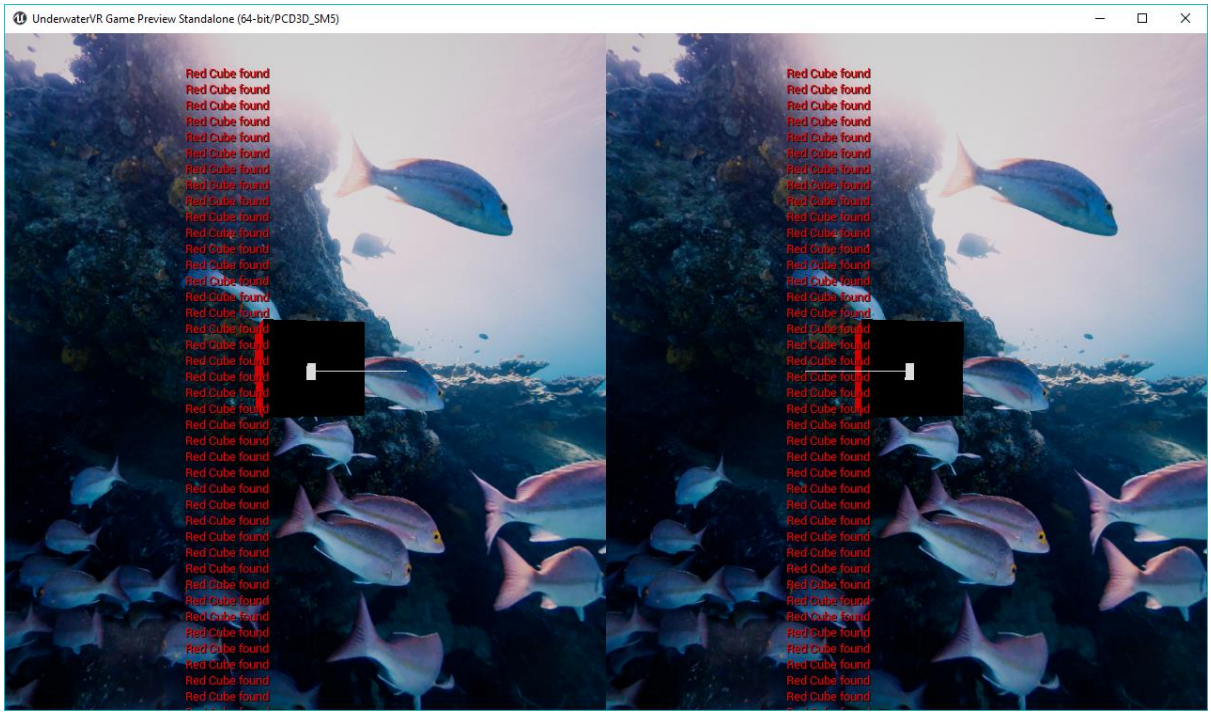
Delete the default plane and leave the two interactive cubes for the time being.

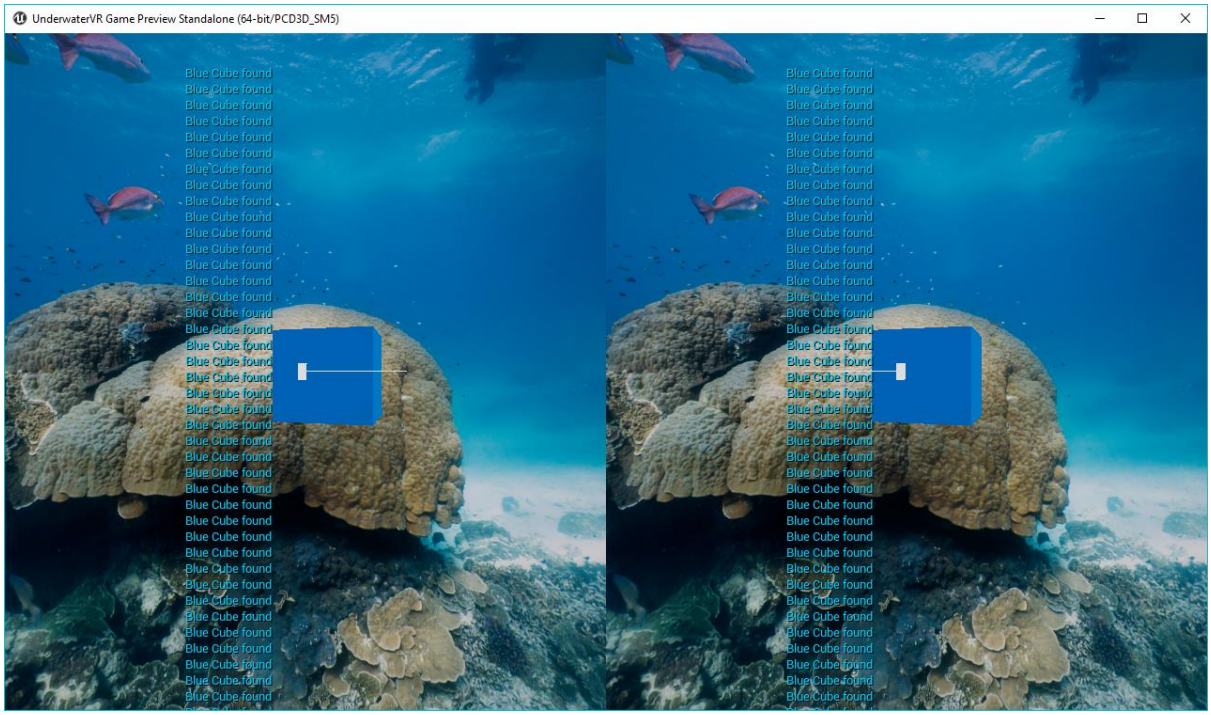
Build the lighting and geometry (BUILD menu in top menu bar). To fix the warnings about the performance hit (re: BP_Skybox1 receiving a pre-shadow which causes a performance problem unless bCastDynamicShadow is set to false), select the BP_Skybox1 in the Outliner and in the details section below it, select the Default Scene Root (Inherited) and in the TRANSFORM settings below that, set the MOBILITY from Movable to Static. Do the same for the SkySphere (Inherited). Finally rebuild the lighting only).



Save, build and deploy to your Android device for testing (LAUNCH in top menu bar).

NOTE: You can always test the app inside Unreal Engine (and view the two screens) using PLAY -> VR PREVIEW. To exit, simply press the ESCAPE key.





Technical note regarding Preshadow warning message:

Any movable components in the scene when combined with stationary lights have to use a special kind of shadow called a Preshadow. This handles the static environment casting dynamic shadows onto the dynamic object. The entire environment between the movable object and the stationary light will have to be rendered into the Preshadow, but only when the movable object actually moves enough to require an update.

If the movable object is very large (e.g. the BP_Skybox1 with its material), and it moves, this can cause the entire scene to have to be re-rendered into the Preshadow depth map. This is what the warning is about. This will involve both a CPU and GPU cost, however it will only happen for one frame (results are cached until the object moves far enough to invalidate the cache) so it's usually pretty hard to measure. If the movable objects move every frame, it can show up in 'stat shadowrendering' (CPU) or 'profilegpu' (GPU).

Problems?

Is the 360-image clear and crisp when viewing in Google Cardboard/Daydream? Is it shimmering or slightly blurred or both? How does it look with only one eye open?

https://answers.unrealengine.com/questions/472544/bad-quality-of-cubemap-with-mipmaps.html

Is the text (string output) clear and legible? Or is it only clear when only one eye is open? Will users with eye issues get focusable VR?

http://www.pnas.org/content/114/9/2183.full (technical look at the use of focus-tunable lenses in VR HMUs (head-mounted units).

Do note that the Google Cardboard lenses don't offer the same quality as the more expensive Oculus and HTC Vive units!

NOTE: To create an interactive VR app containing several 360-degree images, you will need several images set up with separate material objects. This will be addressed later in the tutorial.

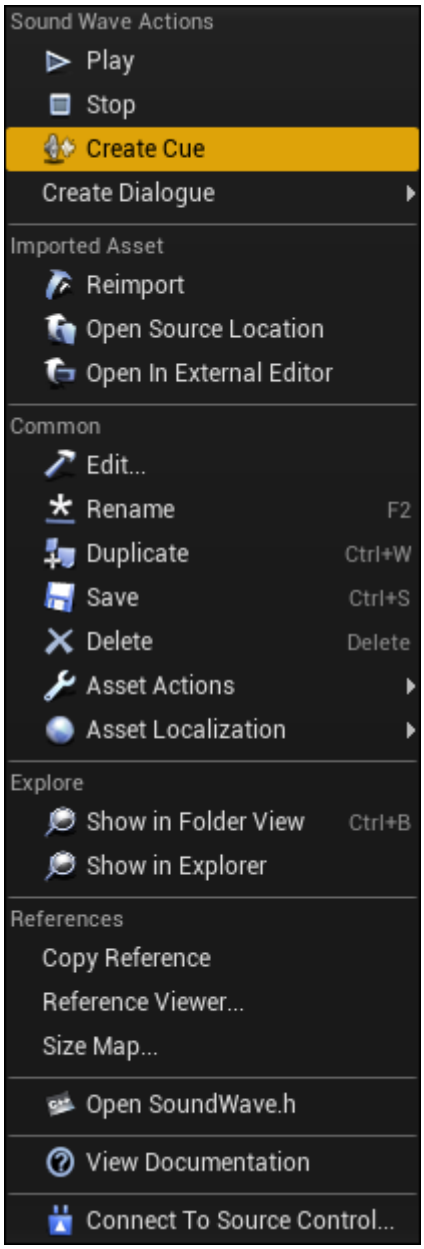
5. Adding background music

There are many options in dealing with audio content within Unreal Engine. Audio can of course consist of background music and sound effects; all of which can be set up to play interactively or to automatically play when a level is loaded.

This section will introduce a simple way to utilise background music into the scene.

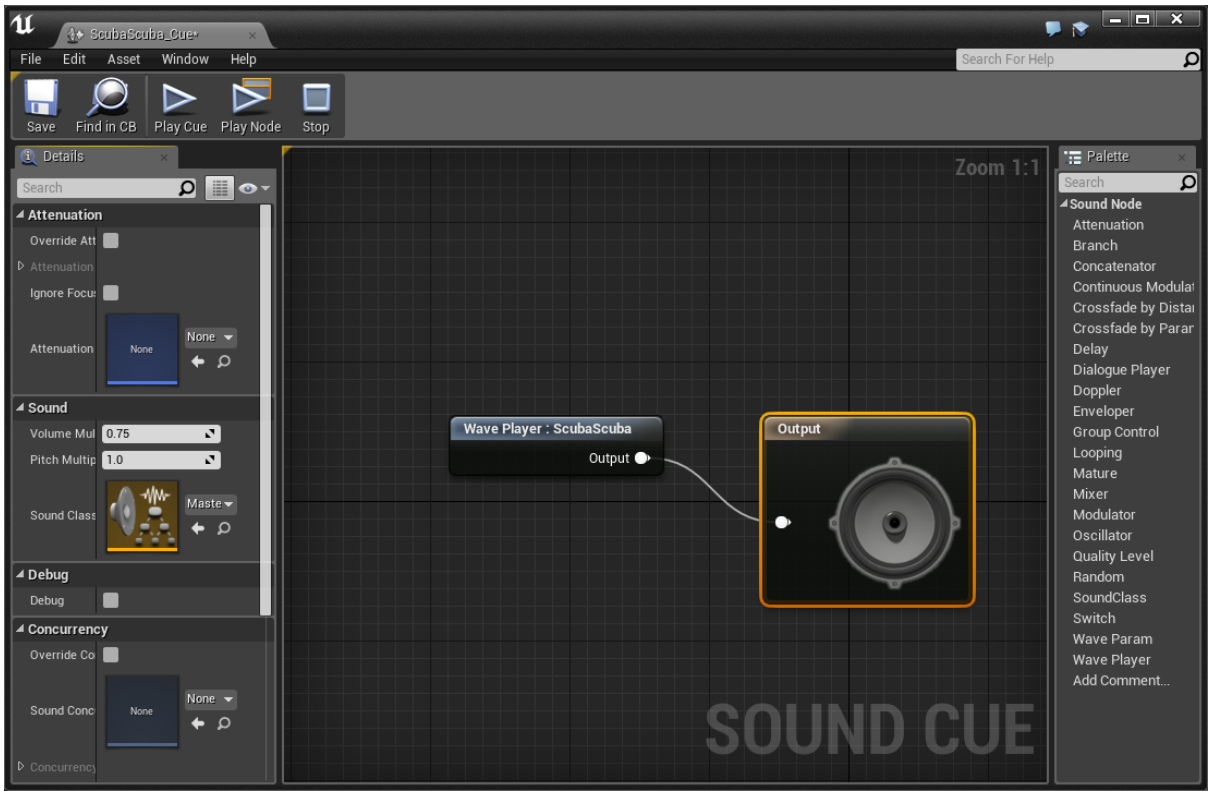
First, create a new folder in the CONTENT area and name it “Audio”. Next, import a 16-bit 48KHz WAV file of your music track (NOTE: MP3 format won’t work).

RMB-click the imported audio and select CREATE CUE.

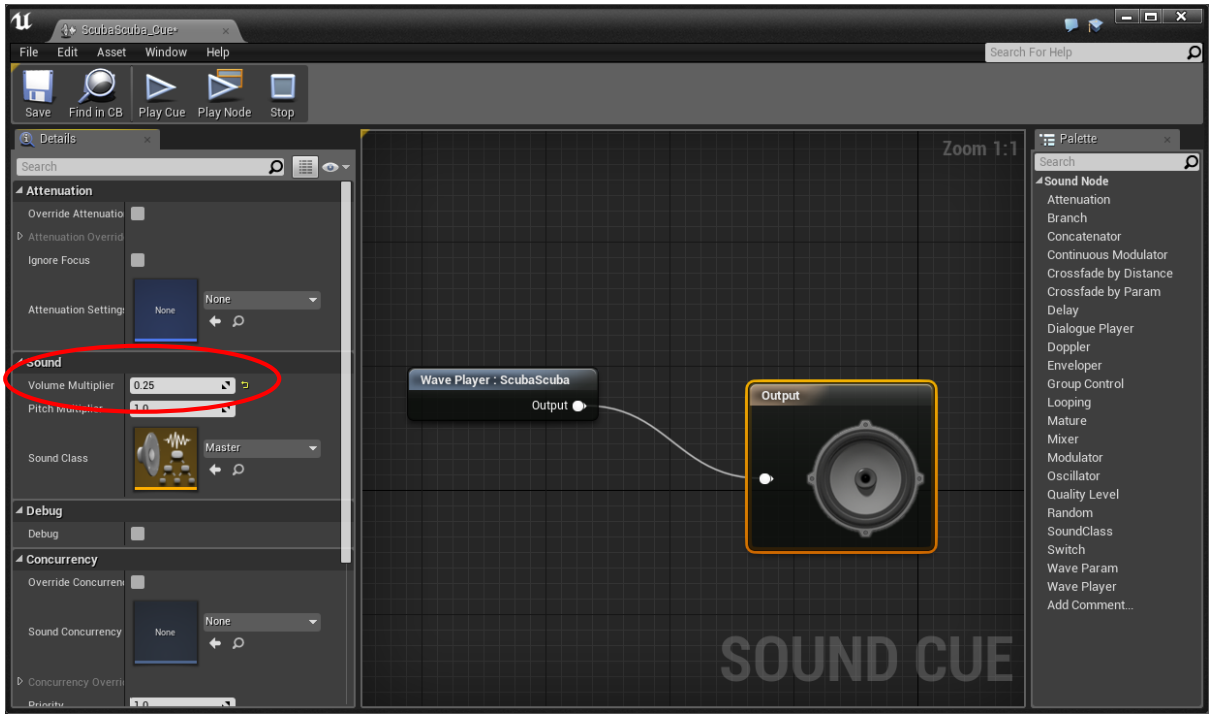


The cue object will automatically be named after the name of the imported .wav file.

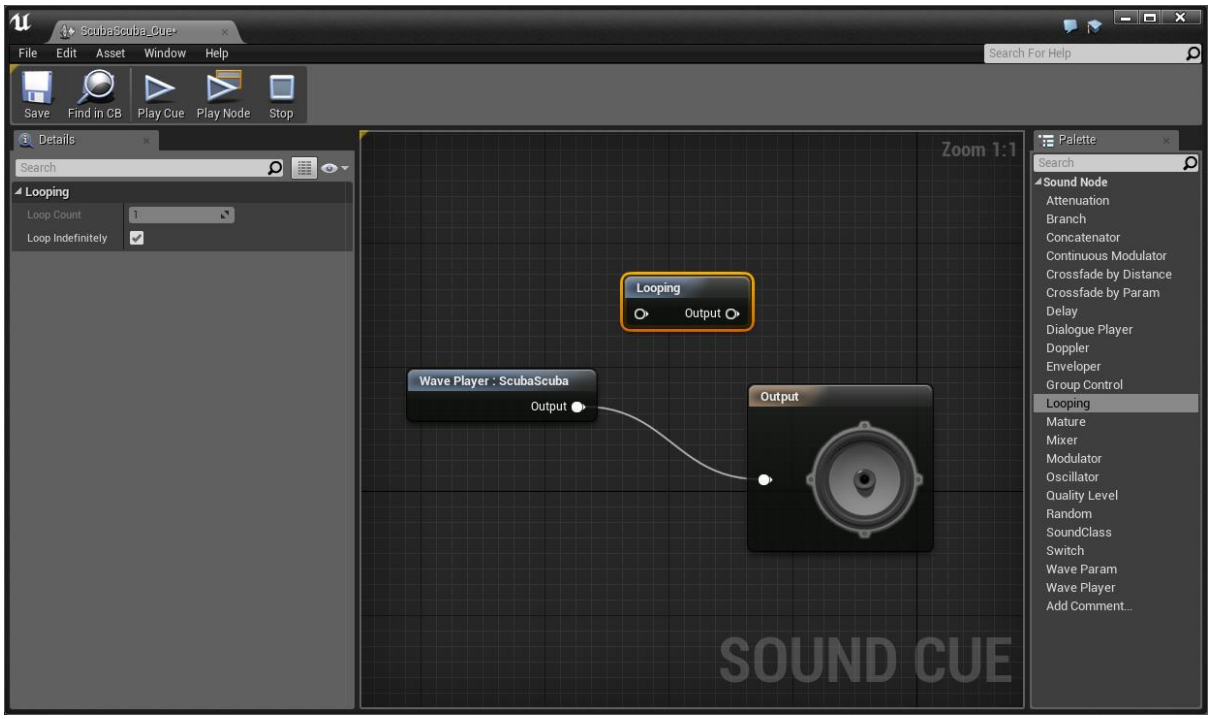
RMB-click the cue object and select EDIT. This will open the edit window with two nodes displayed.



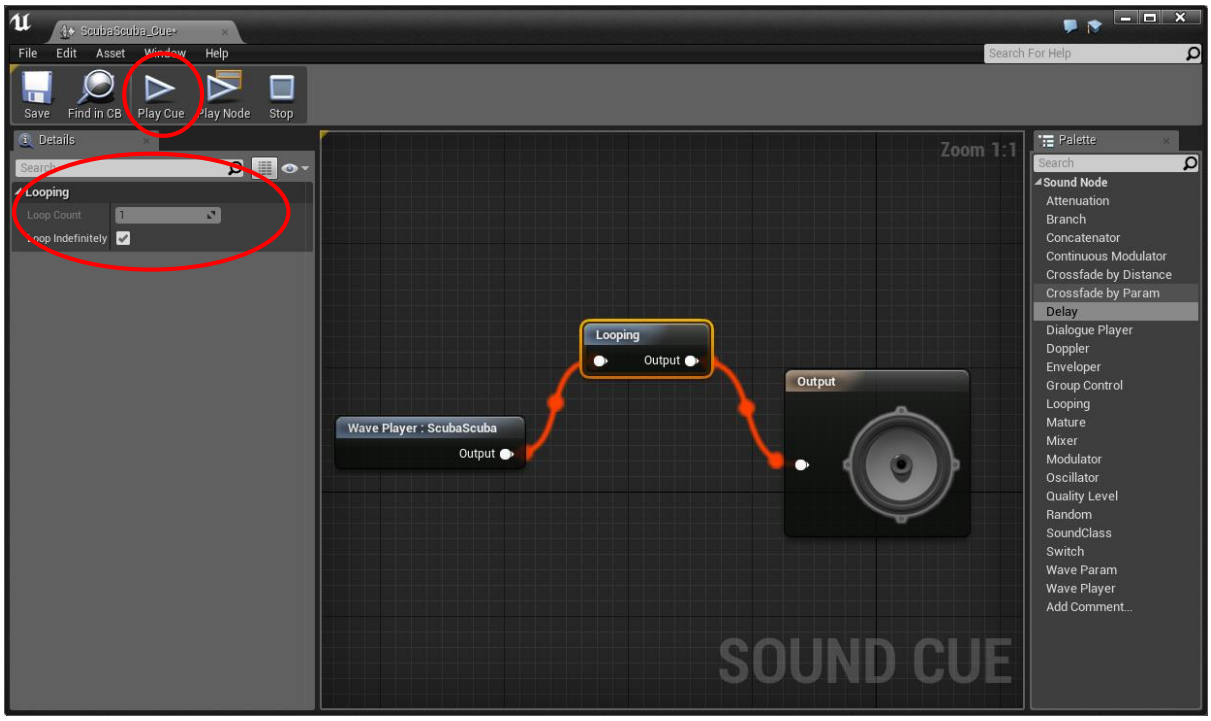
Select the OUTPUT node and on the left side (details settings), you can apply settings (under SOUND) to make changes to the volume and/or pitch. Change the volume multiplier to 0.75 (this will make the music level three quarters that of full volume.



To loop the sound, select the LOOPING option in the palette on the right and drag/drop the node in the main editing window.

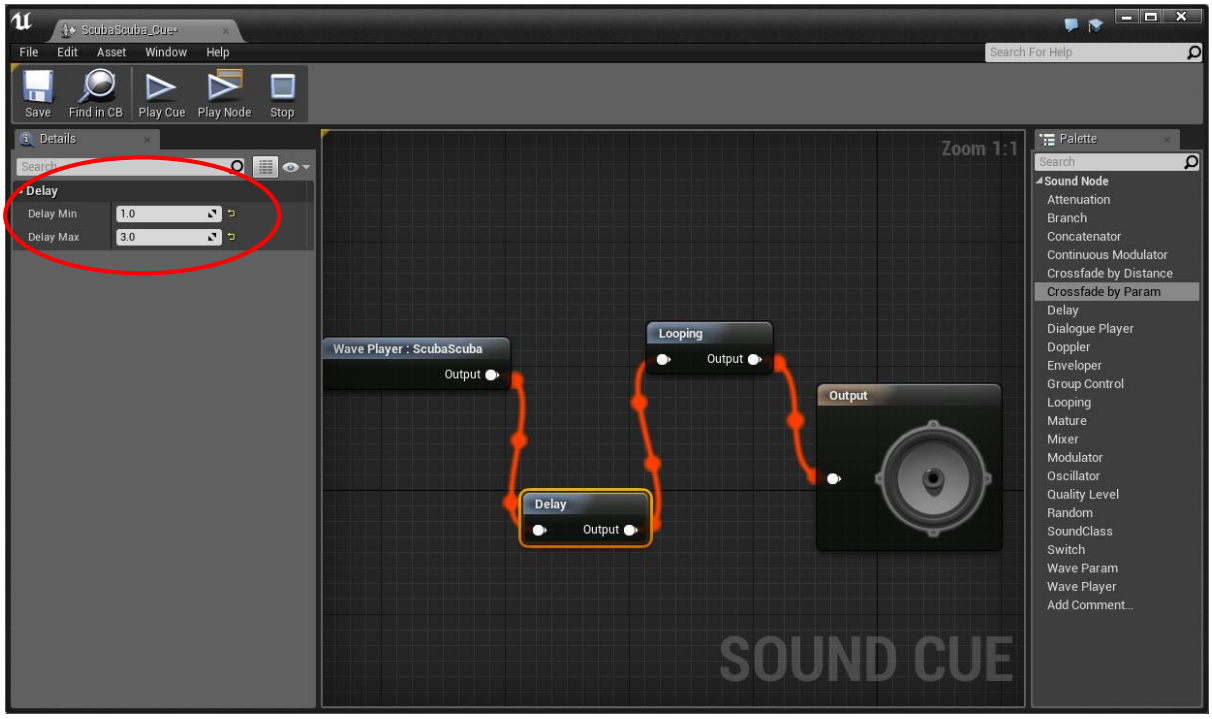


Next, connect the OUTPUT pin of the WAVE PLAYER node to the INPUT pin of the LOOPING node; then take the OUTPUT pin of the LOOPING node to the INPUT pin of the OUTPUT node. Select the LOOPING node and on the left side, in the settings, check the LOOP INDEFINITELY checkbox (if you desire a limited number of loops, uncheck this and enter the value for this. Click the PLAY CUE at the top to see how the flow is represented. You should see the orange connections with an animated flow moving from left to right (refer below example screen shot).



It is possible to add other sounds (including Unreal Engine's supplied audio content) using another WAVE PLAYER node and then by using a MIXER node, mixing them together with various levels prior to connecting them to the final OUTPUT.

To provide a slight delay to the playing of the sound, select the DELAY node from the palette and drag/drop into the editor window. Make the appropriate connections between the input and output pins of the WAVE PLAYER and LOOPING nodes. Select the DELAY node and make changes to the settings for the DELAY MIN (time in seconds) and DELAY MAX (time in seconds). Refer to the screen shot example below.



Save the audio cue and finally drag/drop it into the scene. Test the sound by playing the game.

NOTE: Adding the audio this way into the scene means that it will play when this particular scene is loaded. If other scenes are used, then the audio will not play (as it belongs to another scene). If it is desirous to play audio over multiple scenes, then the audio needs to be added to the game engine (for global playing).

For more information on using audio and sound in Unreal Engine, go to:
https://docs.unrealengine.com/latest/INT/Engine/Audio/

For more detail on how to set up a mixer for several audio inputs, go to:
https://docs.unrealengine.com/latest/INT/Engine/Audio/SoundCues/Editor/index.html

6. Adding interaction via interfaces

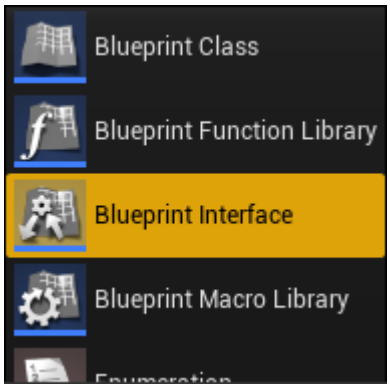
The scene now has some basic interaction with cubes (albeit only printed text), a cubemap with a 360-degree image and background music. Going forward, we will need some means of interaction with a user-interface so that the application can allow changes in the material component (M_Skybox1), which is a part of the BP_Skybox1 blueprint object. To that end, some gaze interaction with a 'hotspot' UI is required – especially if using the Google Cardboard.

How can the changing of the 360-degree image in our material come about? One possibility is to create a second scene (level) with another BP_Skybox, M_Skybox, BP_Pawn, and lighting etc. Another more 'economic' possibility is to programmatically change the material object (M_Skybox1) for the BP_Skybox1 blueprint object based on an interaction.

One of the most effective ways of facilitating interactions between several blueprints is to create an interface. An interface consists only of methods but with no implementation. The interface is then implemented by a blueprint using it, which in turn implements any or none of the methods from the interface.

Blueprint interfaces

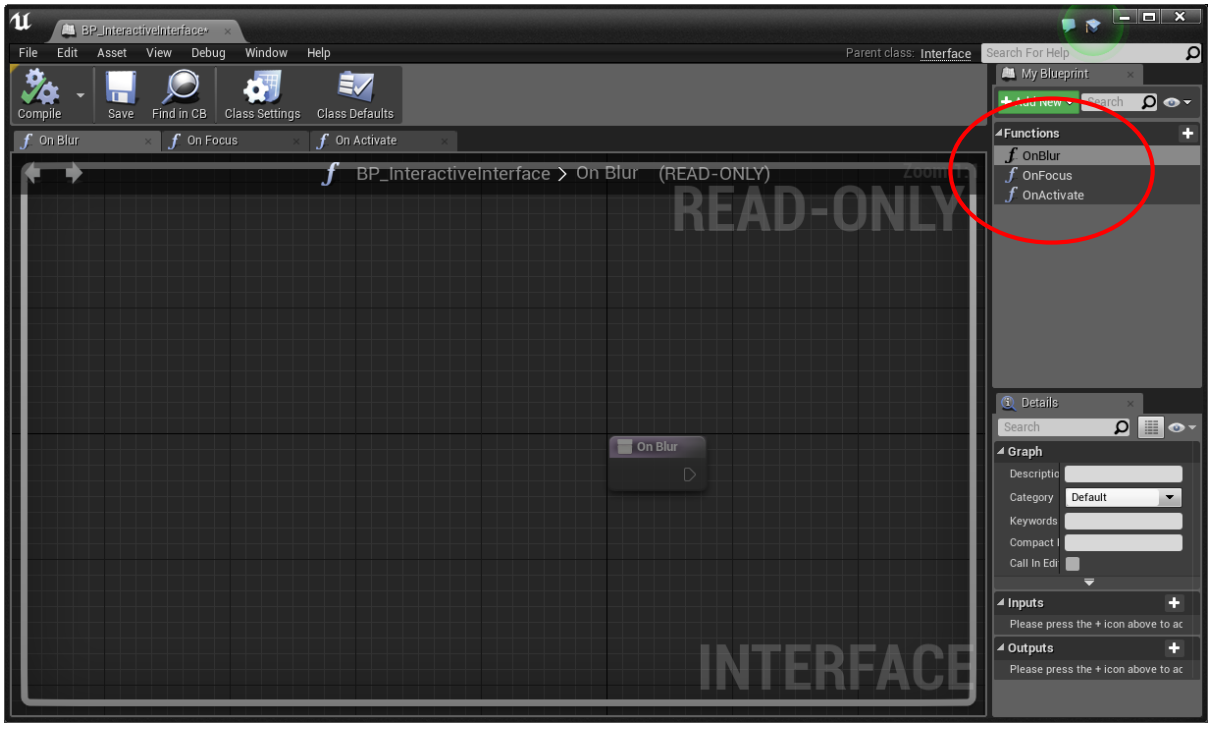
RMB-click in the Blueprints folder and select BLUEPRINTS -> BLUEPRINT INTERFACE (at text-based listing on bottom).



Name the object “BP_InteractiveInterface” and then double-click to open the editor window.

Next, add three functions (use the + button to the Functions panel on the top right)

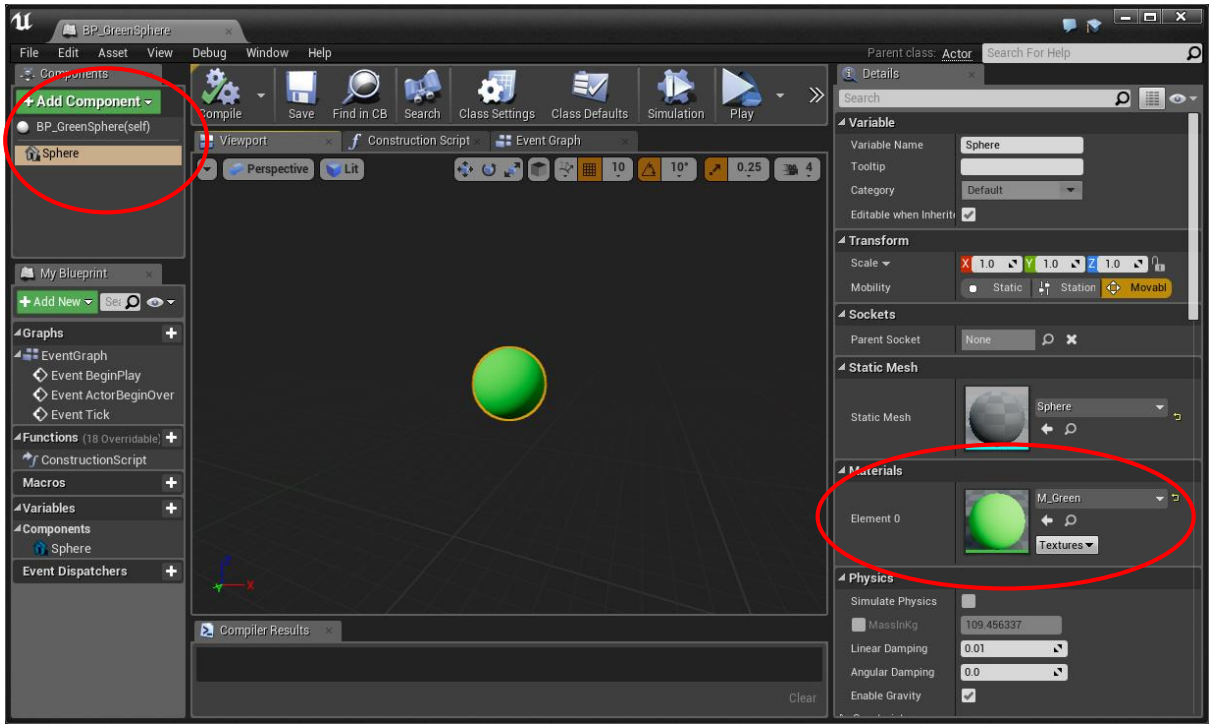
- OnBlur()
- OnFocus()
- OnActivate()



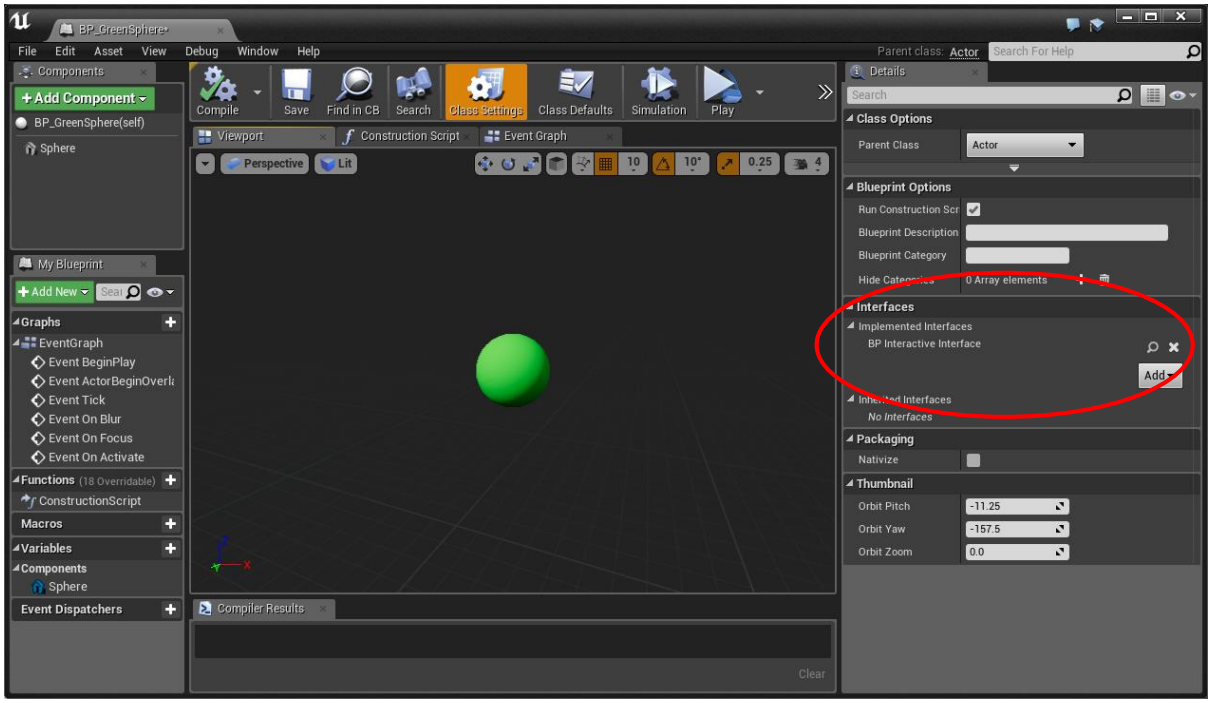
Finally, COMPILE and SAVE.

Go to the MATERIALS folder and make a duplicate of M_Red and name it “M_Green”. Double-click M_Green to edit and change the base colour to green.

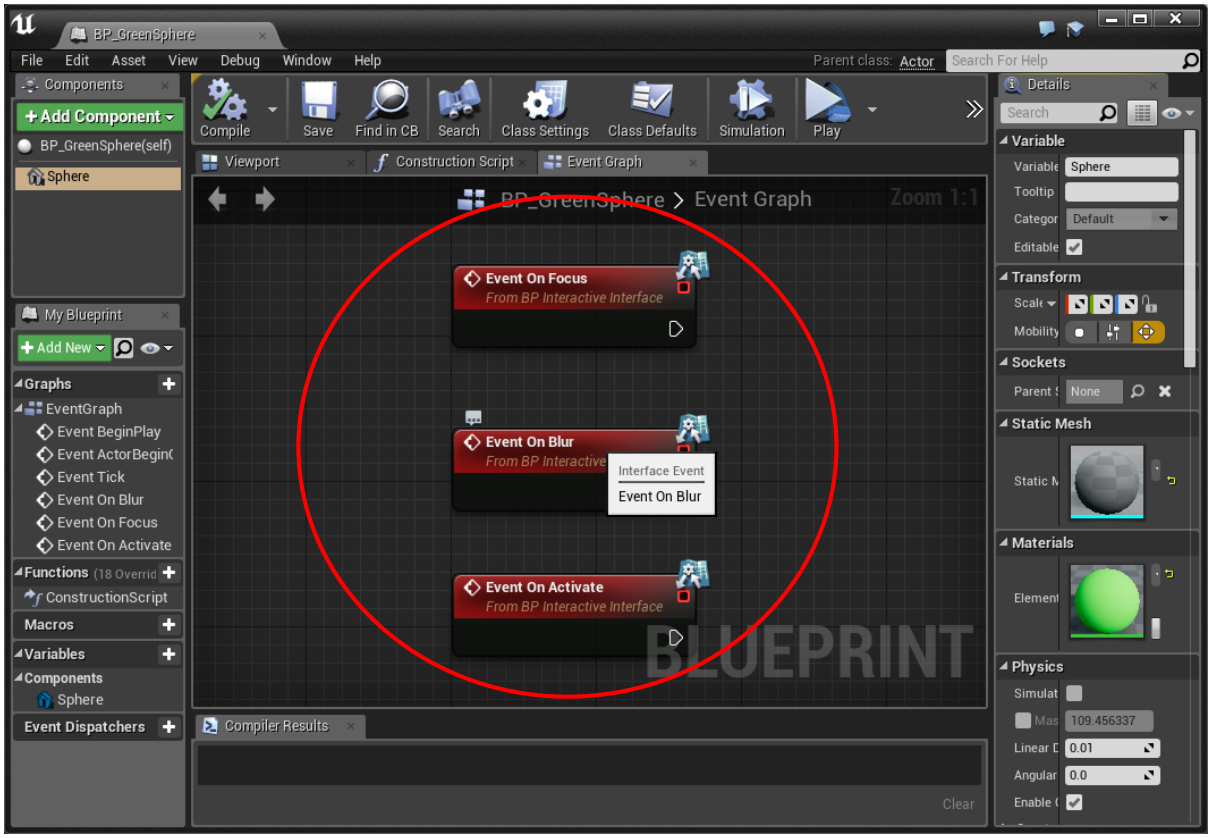
Go to the BLUEPRINTS folder and RMB-click to select the BLUEPRINT CLASS and then select ACTOR for the type. Name it “BP_GreenSphere”. Next, double-click to open the editor. Add a sphere primitive into the viewport and make it the root component. In the MATERIALS settings on the right, choose the M_Green material.



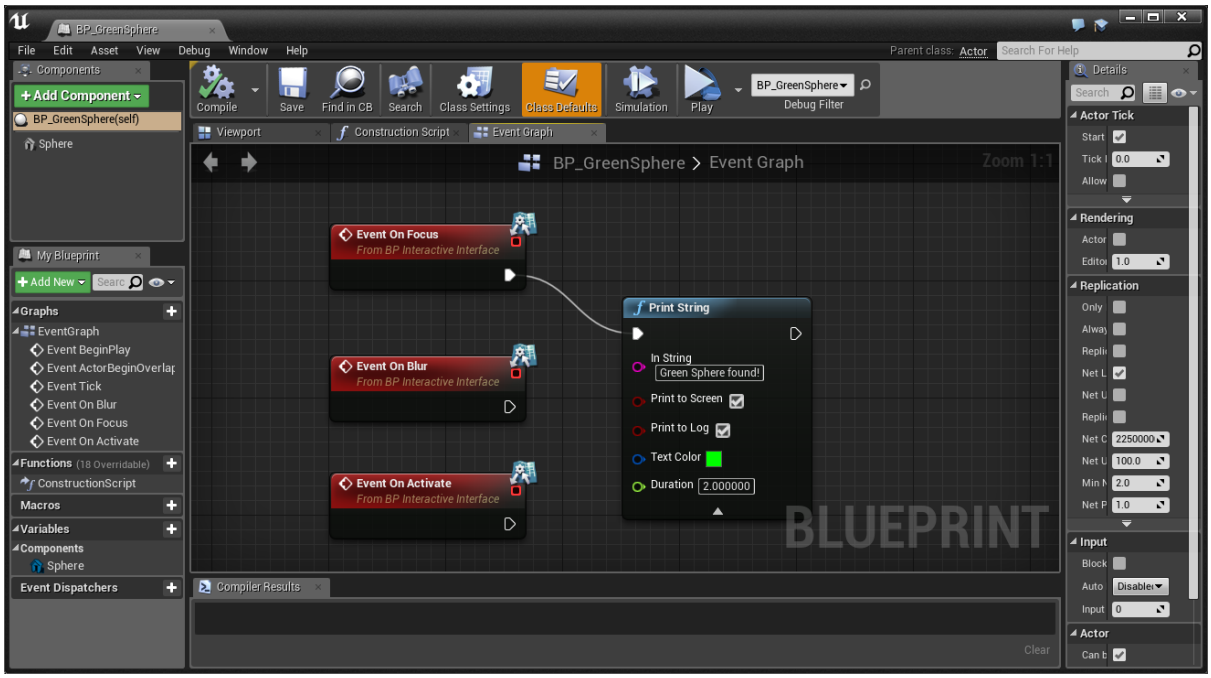
Click the CLASS SETTINGS at the top, which will in turn reveal the class and interface settings and options. Go to the INTERFACES and under IMPLEMENTED INTERFACES, select BP Interactive Interface (the one you created before).



Next, RMB-click inside the EVENT GRAPH window and search/select EVENT ON FOCUS. Do the same for EVENT ON BLUR and EVENT ON ACTIVATE. You should then have the three event trigger nodes set up as per the sample screen shot below.



Drag out the white output arrow for the EVENT ON FOCUS node and search/select PRINT STRING. Enter an appropriate string text message (IN STRING) – for example: “Green Sphere found!”



COMPILE and SAVE.

Next, double-click the BP_RedCube blueprint in the blueprint folder to open the editor. Select the class settings and add the BP Interactive Interface.

Following this, add the three event trigger nodes – EVENT ON FOCUS, EVENT ON BLUR and EVENT ON ACTIVATE.

Drag out the white arrow output and search/select PRINT STRING. Enter an appropriate string value.

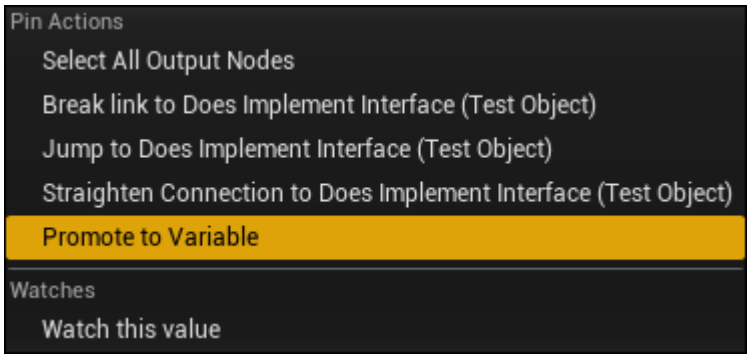
COMPILE and SAVE.

Repeat this again for the BP_BlueCube.

You should now have three blueprint interactable objects, each of which implements the BP_InteractiveInterface interface blueprint --- remember, interfaces that are implemented share the same name functions, but can have different implementation code. This means that the cubes and the sphere will do different things when we interact with them.

Double-click the BP_Pawn blueprint to open the editor. Select the two PRINT STRING nodes at the end and delete. Next, select the CAST TO BP_REDCUBE and CAST TO BP_BLUECUBE nodes and delete as well.

RMB-click on the HIT ACTOR pin (BREAK HIT RESULT node) and select PROMOTE TO VARIABLE

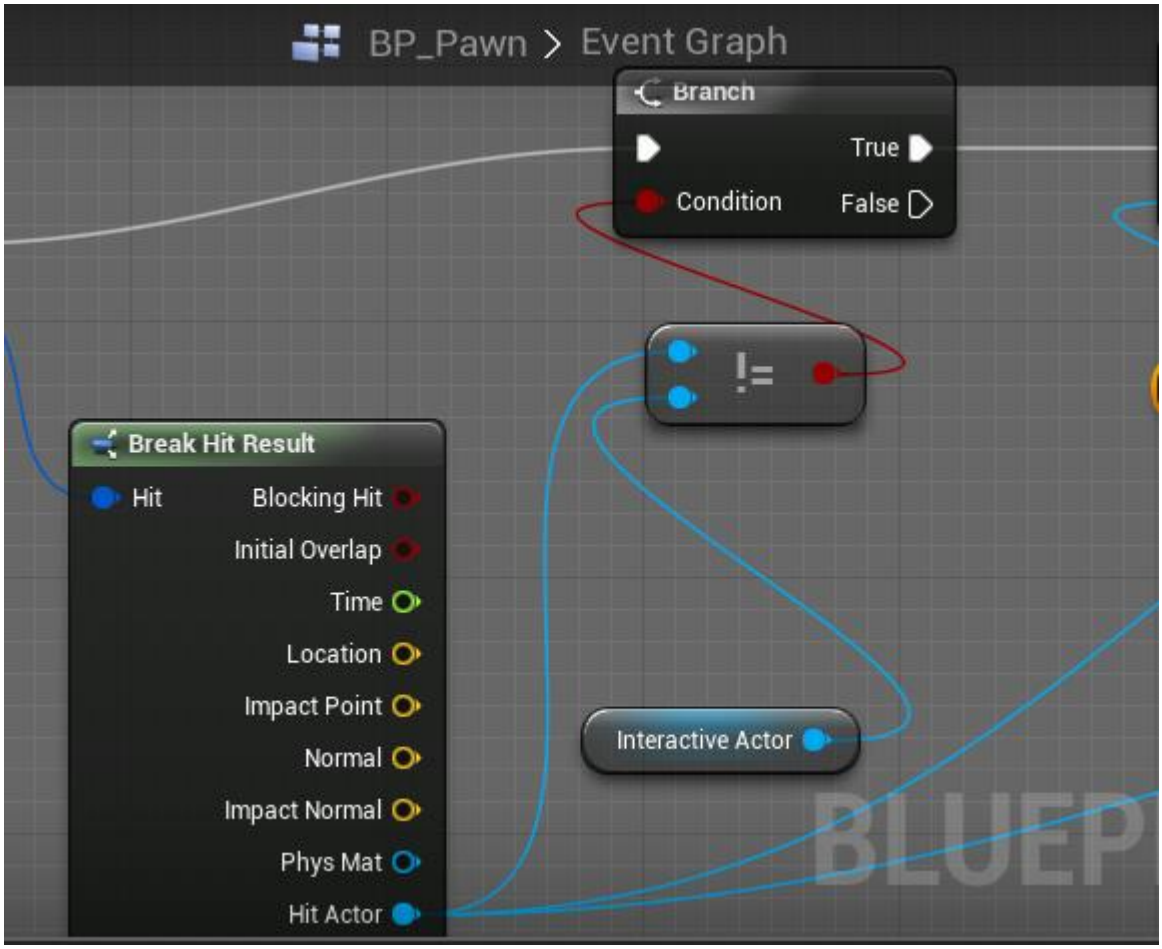


This will create a new object variable on the left side (under VARIABLES) – name it "InteractiveActor".

Drag out the variable name "InteractiveActor" directly from the VARIABLES listing out and drop anywhere on the right of the BREAK HIT RESULT node.

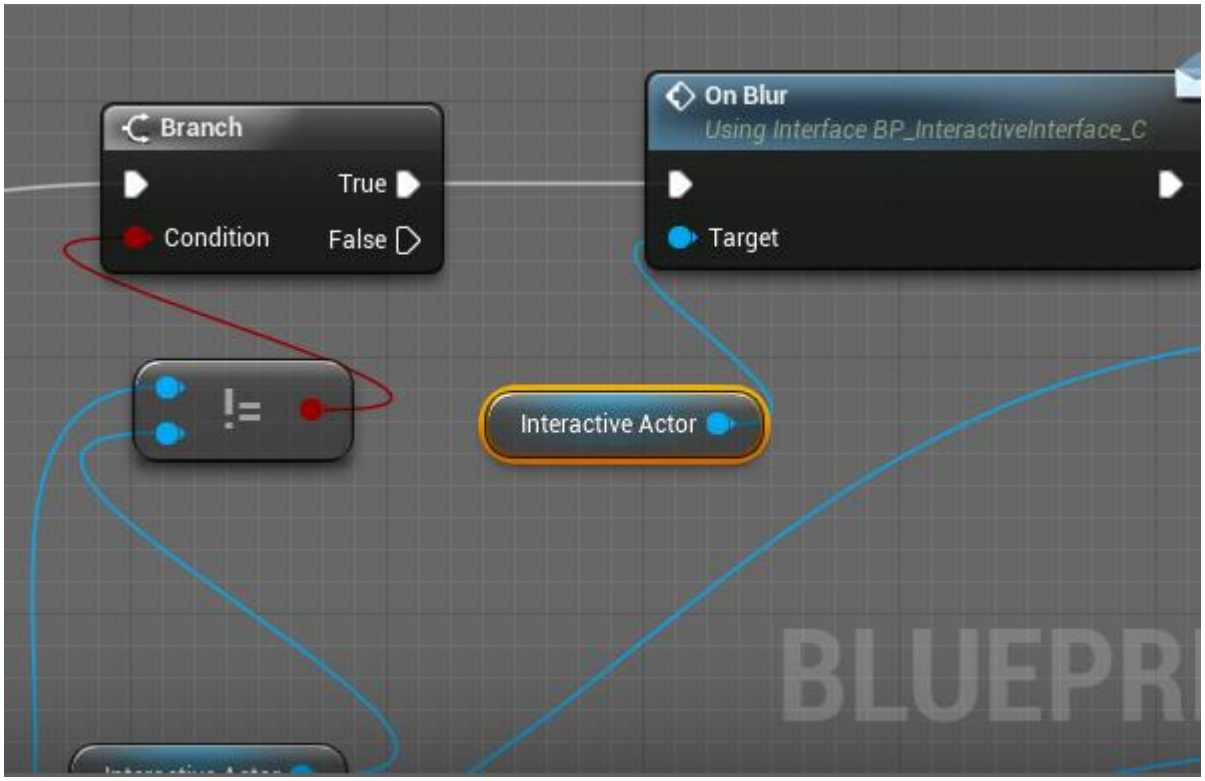
From the BREAK HIT RESULT node, select the HIT ACTOR and drag out the pin. Search/Select NOT EQUAL (OBJECT) or simply enter "!=". Next, connect the output pin of INTERACTIVE ACTOR node to the bottom left input pin of the != node.

Drag out the white output pin at the top left of the LINE TRACE BY CHANNEL node and search/select BRANCH. Connect the red output of the != node to the bottom left CONDITION input pin of the BRANCH node. Refer screen shot example below.



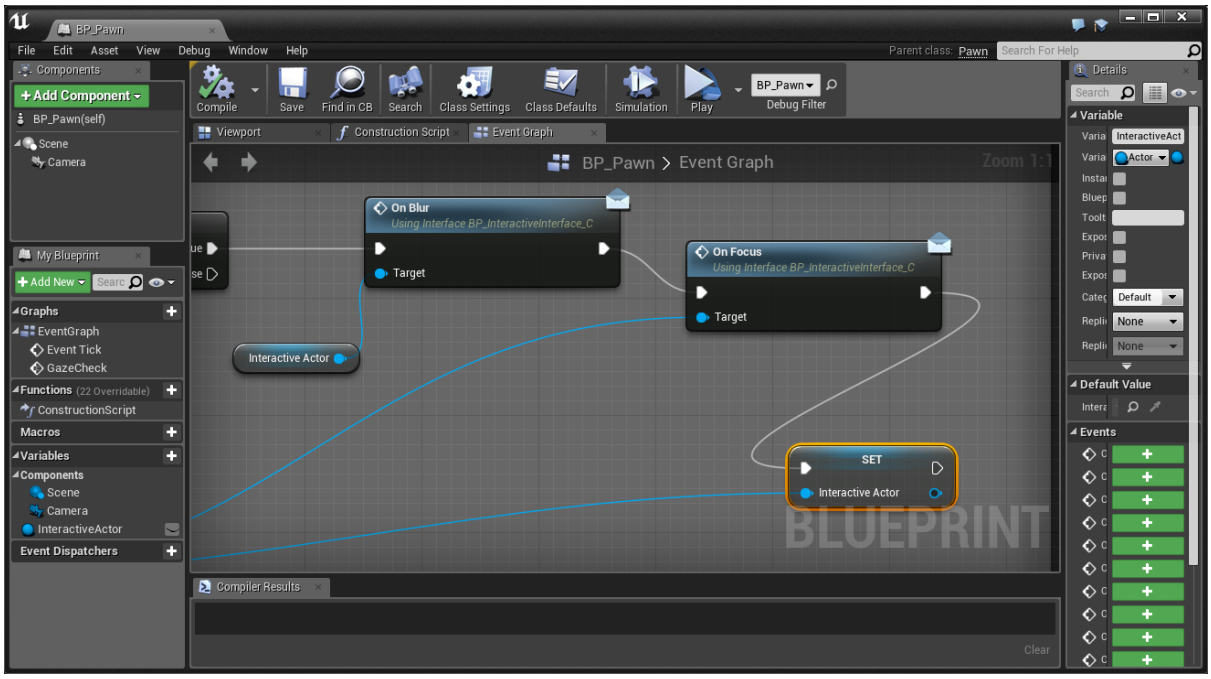
Drag out the TRUE pin on the BRANCH node and search/select ON BLUR.

Drag out the INTERACTIVEACTOR variable again to a position left of the ON BLUR node. Next, connect the INTERACTIVE ACTOR node output to the TARGET input pin of ON BLUR. Refer screen shot example below.

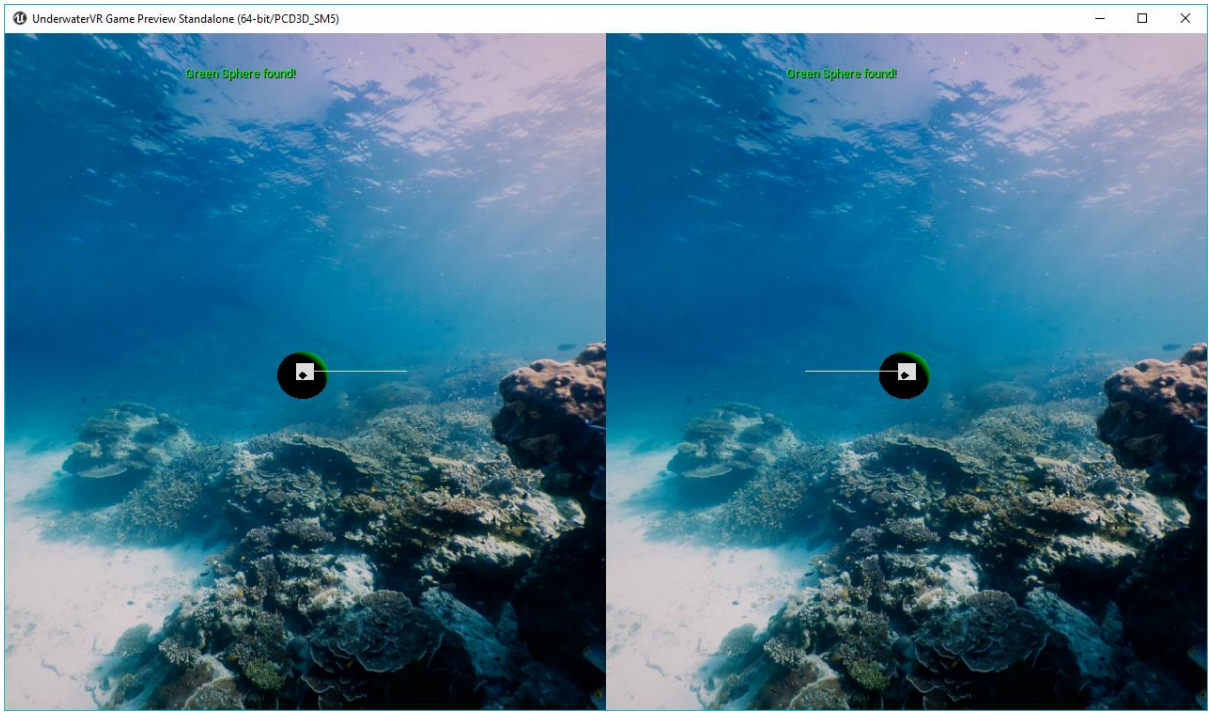


Drag out the HIT ACTOR pin from the BREAK HIT RESULT node and search/select ON FOCUS. The HIT ACTOR pin should connect with the TARGET pin of ON FOCUS. Next, connect the white output arrow of ON BLUR node to the white input arrow of ON FOCUS node.

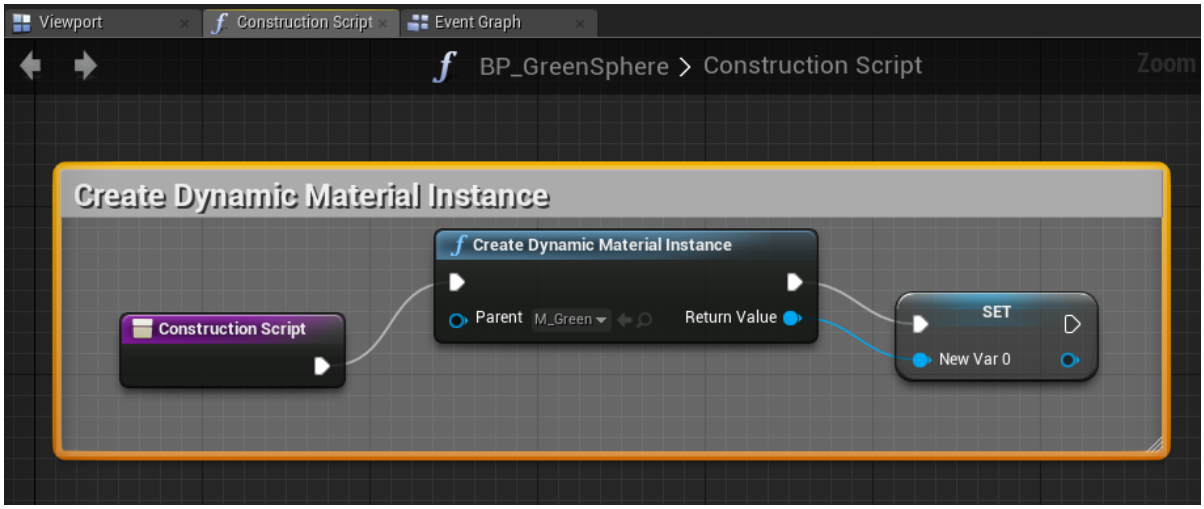
Drag out the HIT ACTOR pin again from the BREAK HIT RESULT node and search/select SET INTERACTIVE ACTOR. There should be a connection made between the HIT ACTOR pin and the INTERACTIVE ACTOR input pin of the SET node. Finally, connect the white output arrow of the ON FOCUS node to the white input arrow of the SET node.



Build and test the app inside Unreal Engine and check that the appropriate string messages appear whenever the camera is aimed at the red cube, blue cube or the green sphere.



Double-click the BP_GreenSphere blueprint object to open the editor. Go into the CONSTRUCTION SCRIPT window. Drag out the white output arrow of the CONSTRUCTION SCRIPT node and search/select CREATE DYNAMIC MATERIAL INSTANCE. RMB-click the RETURN VALUE output pin and select PROMOTE TO VARIABLE. It will appear on the left as the name “NewVar_0” and a SET node will appear with connections to both pins – refer screen shot example below.



Go to the EVENT GRAPH and delete the PRINT STRING node. Drag the Sphere component (In the VARIABLES listing on the left side) out and position it near the EVENT ON FOCUS node.

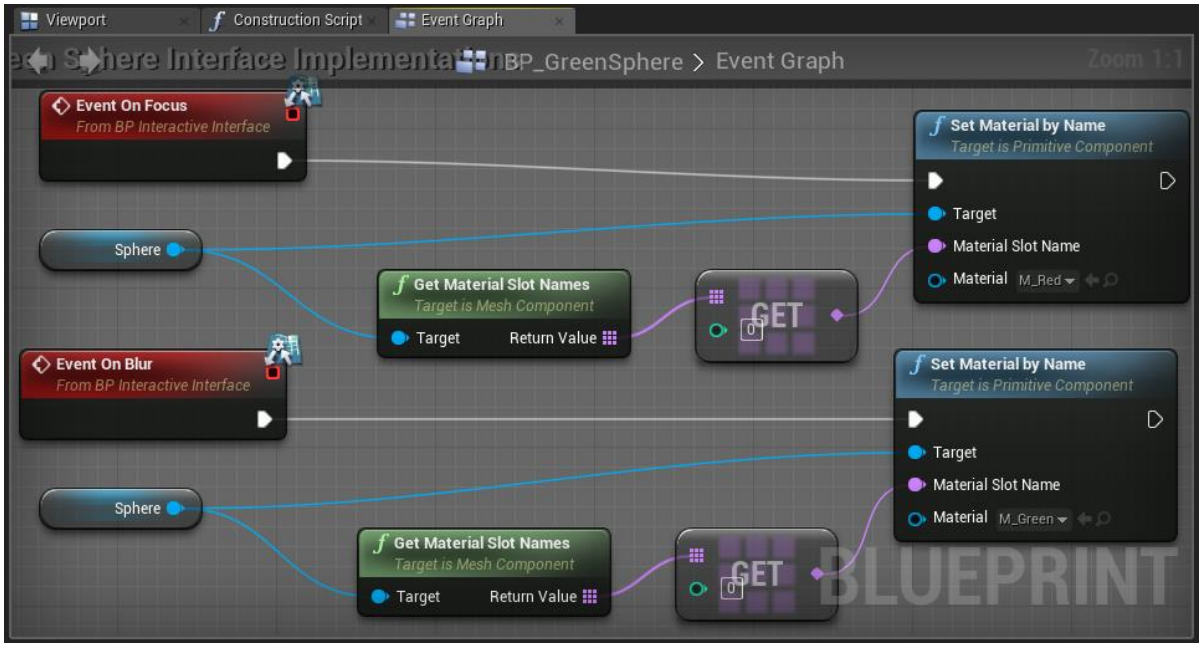
Drag the white output arrow of the EVENT ON FOCUS node, and search/select SET MATERIAL BY NAME. Connect the white output arrow of the EVENT ON FOCUS node to the white input arrow of the SET MATERIAL BY NAME node. Ensure that the Sphere output pin is connected to the TARGET input pin of the SET MATERIAL BY NAME node.

Finally, drag out the Sphere output pin and search/select GET MATERIAL SLOT NAMES. Drag out its return value to GET (A REF). Connect the output pin of the GET node to the MATERIAL SLOT NAME input of the SET MATERIAL BY NAME node.

In the SET MATERIAL BY NAME node, select the MATERIAL option of M_Red (this will be the on Focus material).

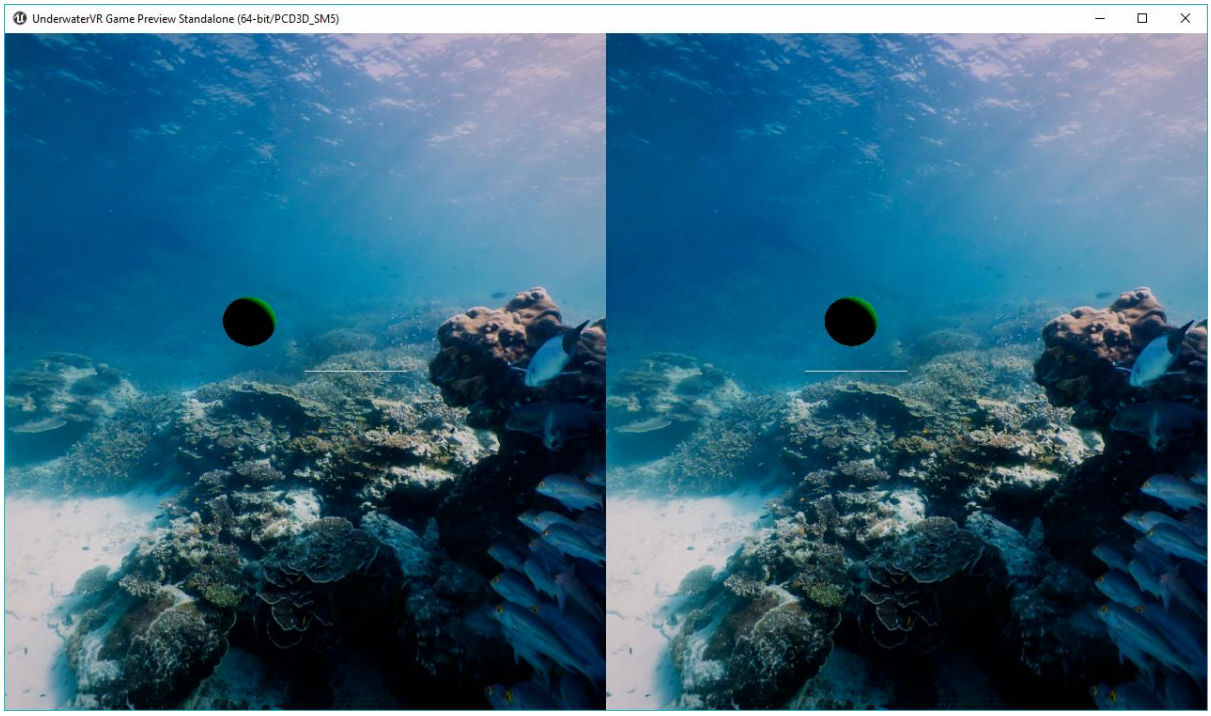
Repeat this for the EVENT ON BLUR node – refer screen shot example below.

In the SET MATERIAL BY NAME node, select the MATERIAL option of M_Green (the default material for the green sphere).

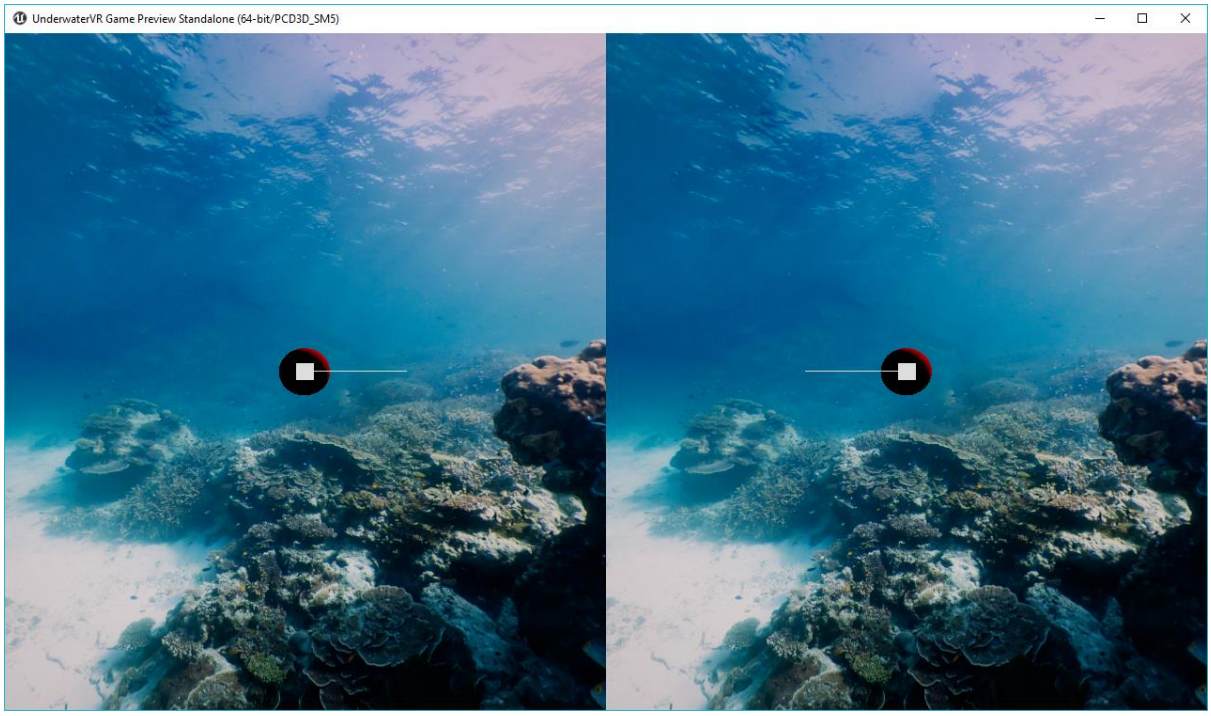


COMPILE and SAVE.

Save and play the app for testing inside Unreal Engine. You should see the material change in the sphere object (green when it is not focused and red when it is focused).

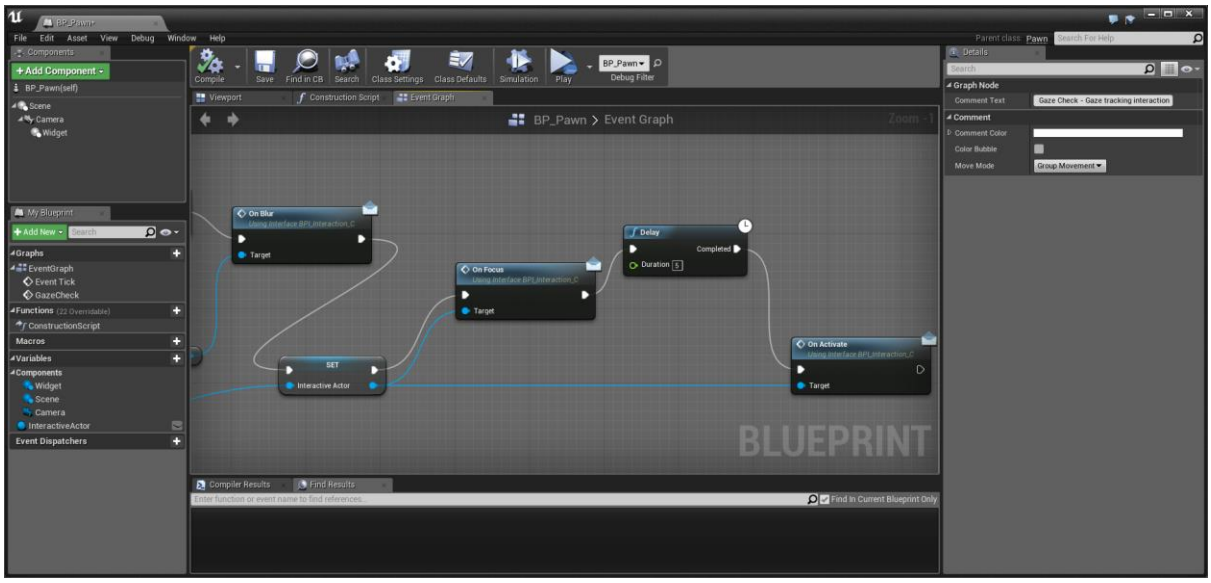


Green material present on sphere (On Blur)



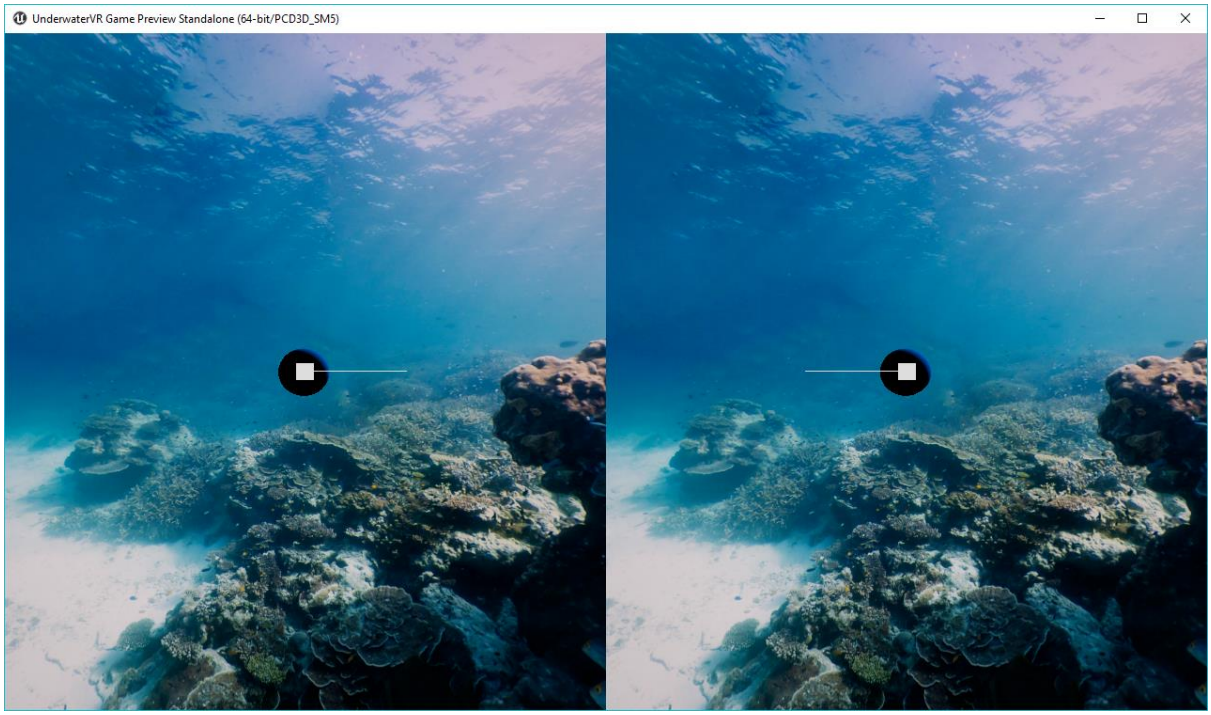
Red material present on sphere (On Focus)

Double-click the BP_Pawn blueprint and from the SET node, drag out and search/select the DELAY node. Enter a duration value of 5 (number of seconds for delay). Drag out the white arrow output of the DELAY and search/select ON ACTIVATE. Connect the output pin from the SET node to the input TARGET pin of the ON ACTIVATE node.



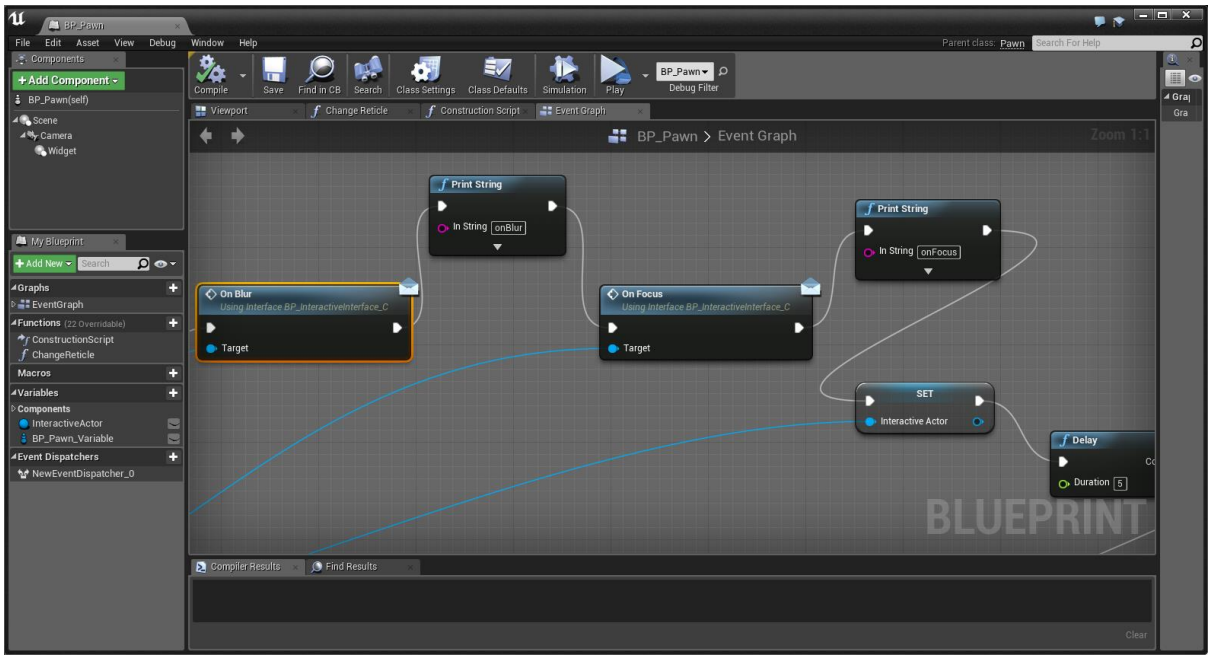
COMPILE and SAVE.

Next, double-click the BP_GreenSphere blueprint and in the EVENT GRAPH, select the white output arrow of the EVENT ON ACTIVATE node and drag out. Search/Select the node SET MATERIAL BY NAME. Complete this section as you did for the other two events; the difference is in the material – set this value to M_Blue. COMPILE and SAVE again. Test the app internally and then focus on the green sphere. It should change to the blue material (M_Blue) after 5 seconds of being in focus.

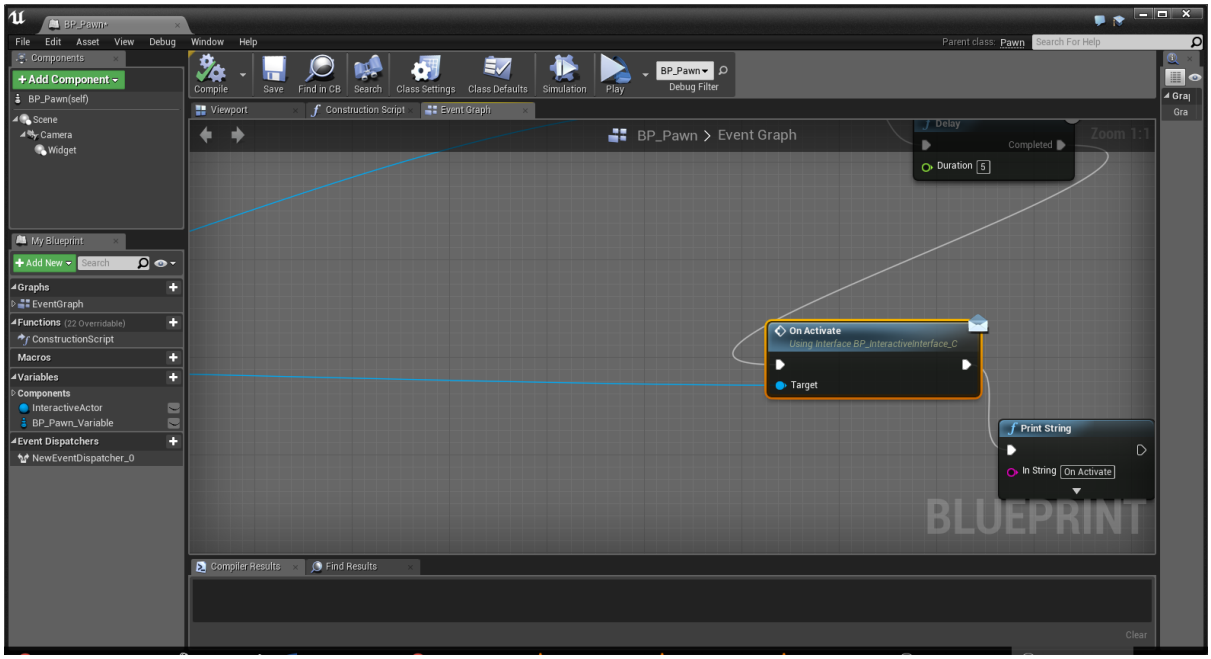


Optimising code

If you were to include a PRINT STRING node between the ON BLUR and ON FOCUS node --- and --- another PRINT STRING node between the ON FOCUS and SET nodes (refer example screen shot below), we could see what results when the gaze interaction is made on any of the cubes or the sphere.



In addition, we shall check what happens when OnActivate() is called by adding a PRINT STRING node when this is executed.



You would see that each time the gaze interaction is made (i.e. that focus is made on one of the interactable objects), the message “On Focus” and then “On Blur” appears; and when the gaze is removed (i.e. that blur is then activated), the same message appears, in the same order! We also note that the message “On Activate” appears 5 seconds after “On Focus” is displayed (noting that “On Focus” only appears for two seconds) – either with the gaze on or off.

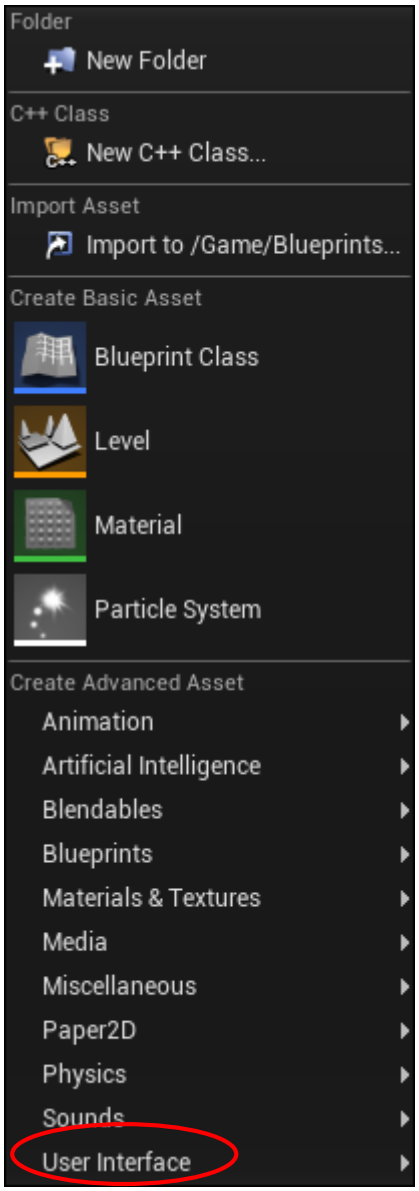
This is obviously not an optimum way to go as far as programming goes. More on this later.

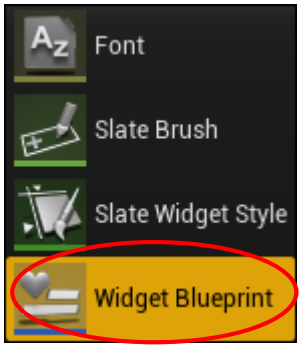
7. Creating a Gaze Reticle using a UI Widget

A VR gaze reticle is a graphical pointer or cursor used to trigger actions when looking at objects or UI components. Typical examples can be a simple 2-D dot, crosshair, circle, square or other geometric shape. For this exercise, you will need to either download or create your own 2-D cursor (it needs to be a PNG file, ensuring that the background is transparent). This tutorial will use a blue circle PNG graphic imported into Unreal Engine placed in a folder named “UI” and named “blue_circle”.

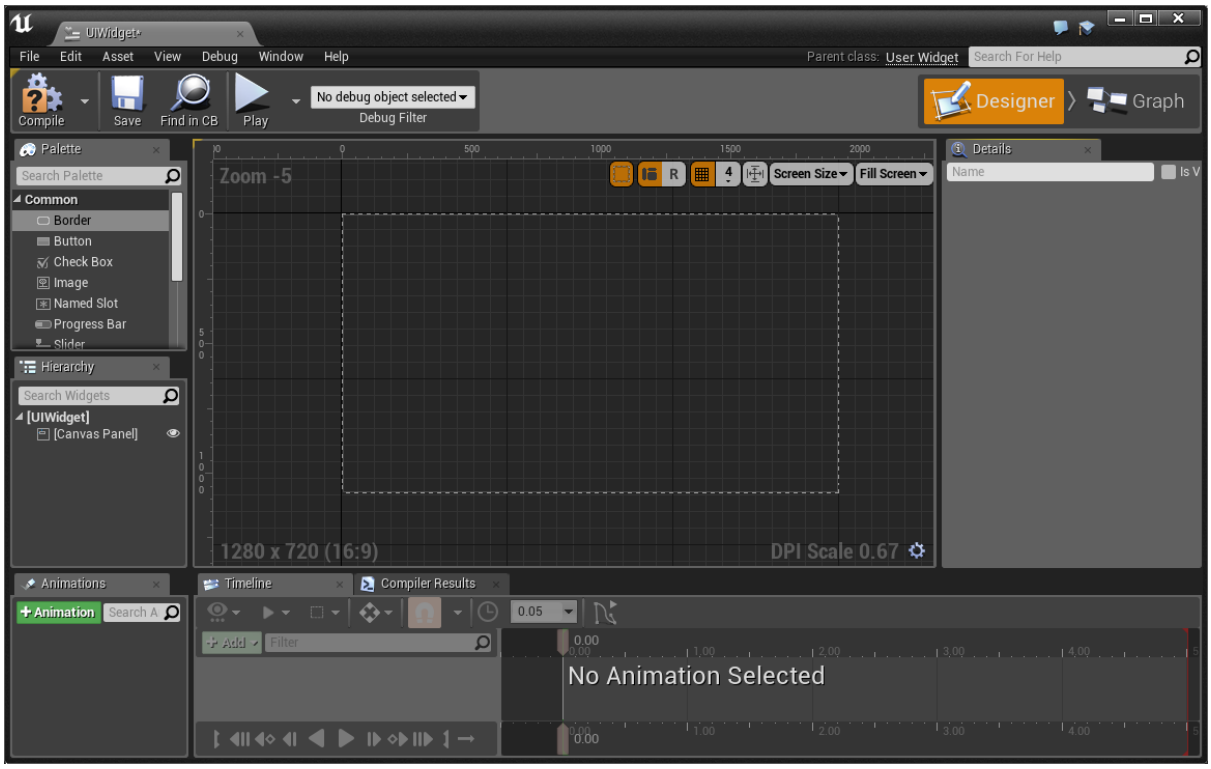
RMB-click in the Blueprints folder and choose USER INTERFACE -> WIDGET BLUEPRINT (refer screen shot example below). Name it “WBP_Circle”.

NOTE: Unreal Engine refers to the visual UI authoring tool as UMG UI Designer where UMG stands for “Unreal Motion Graphics”.

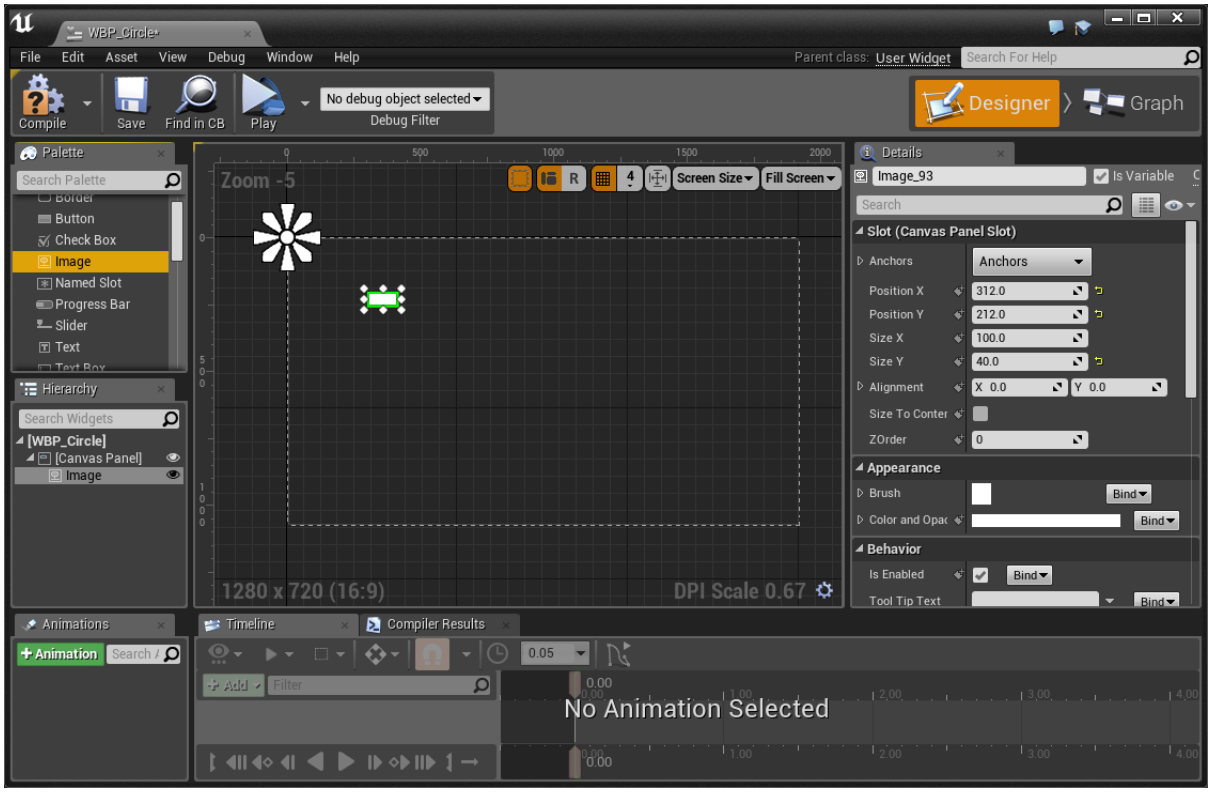




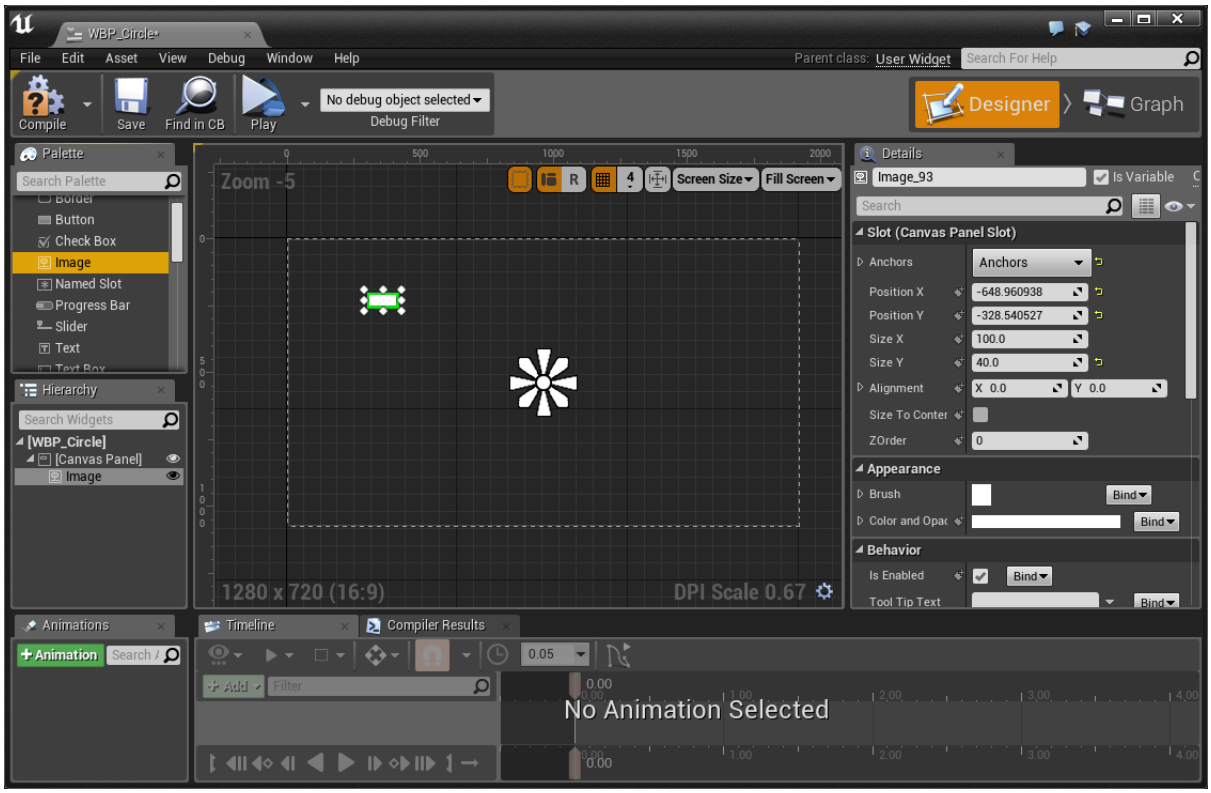
Double-click WBP_Circle (Widget Blueprint) to open the editing window.



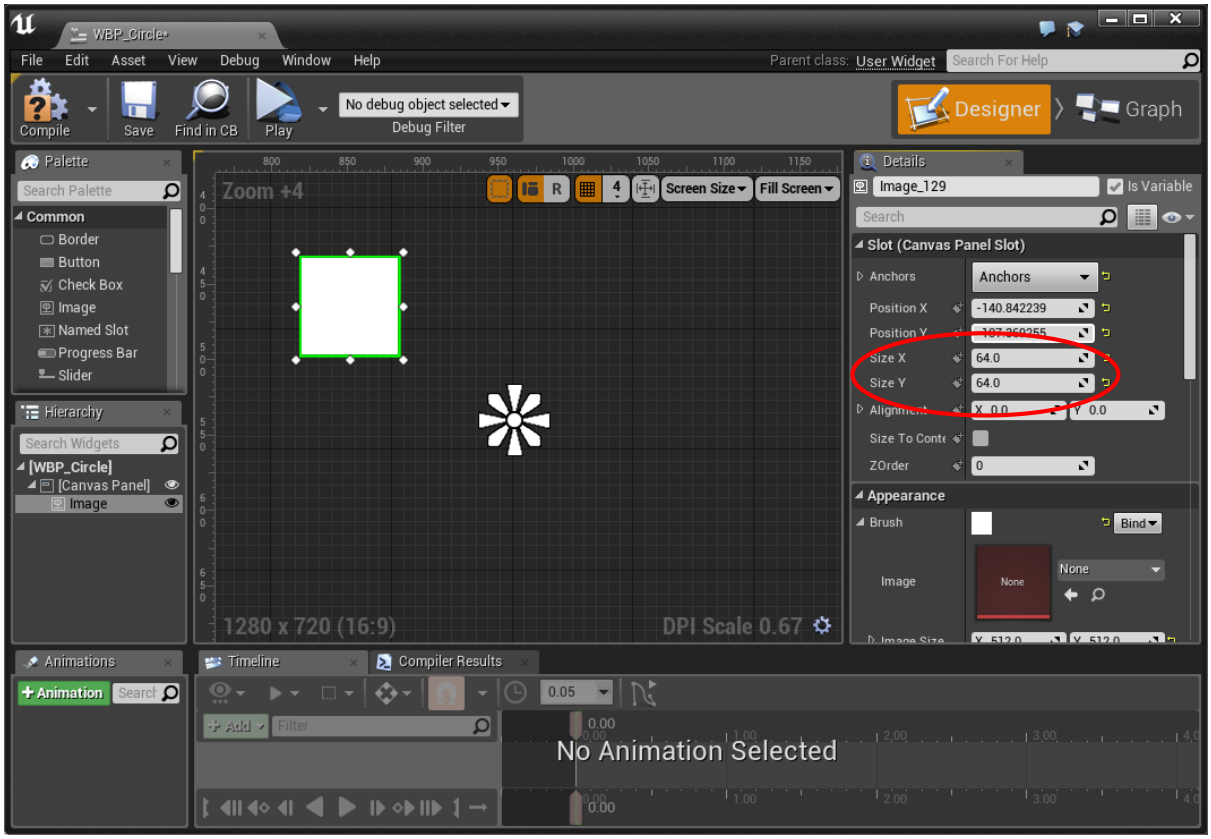
In the Palette window, select and drag out the IMAGE onto the main builder window. You should see a small (green bordered) rectangle appear inside a rectangle defined by a broken line. Also, you should see a large asterisk icon – this is the anchor.



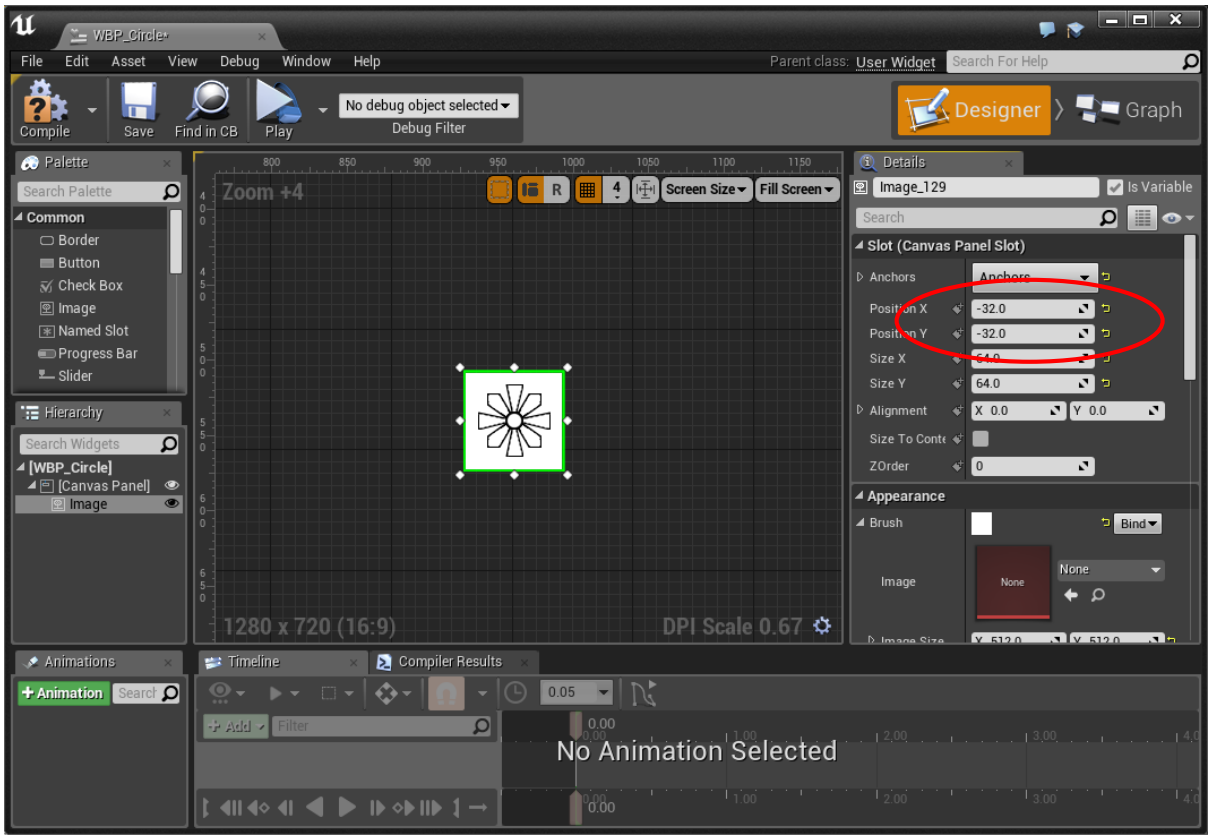
Select the Anchors drop-down menu and choose the centred anchor icon



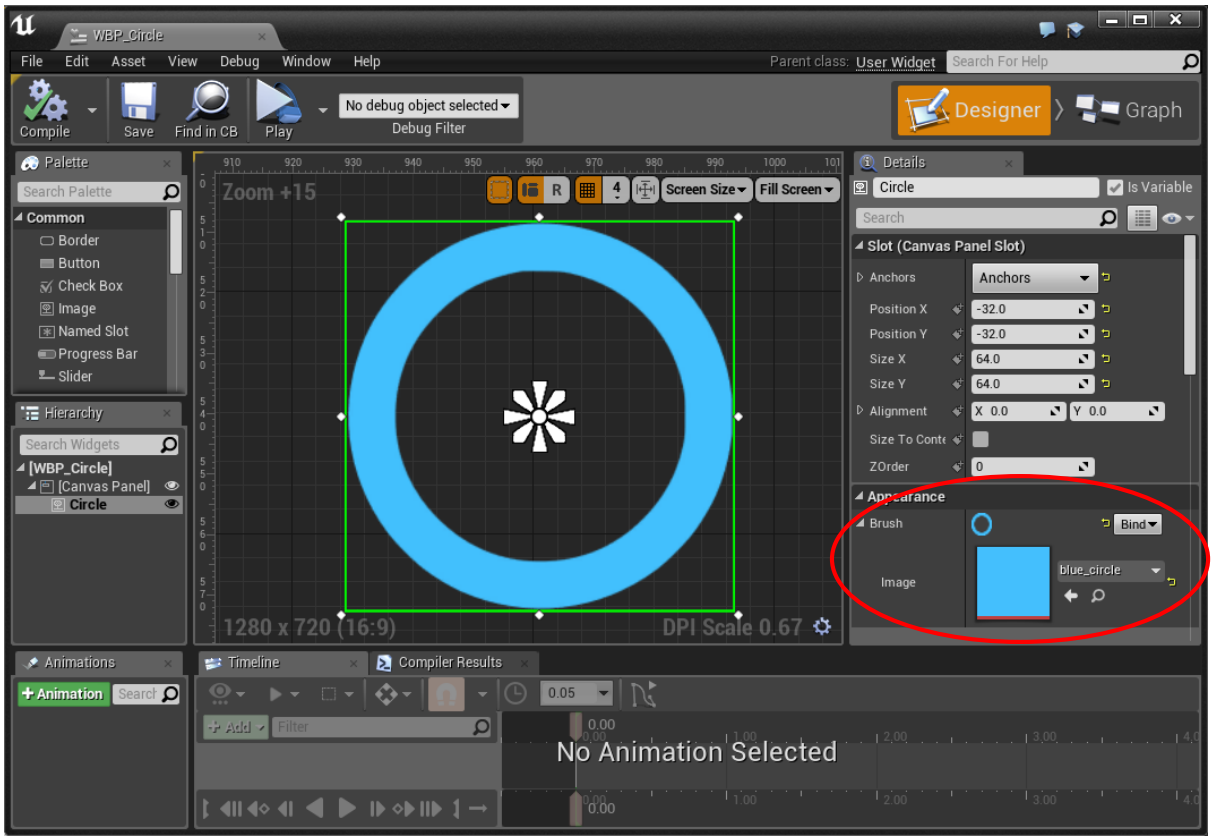
With the small rectangle selected, change the Size X to 64 and the Size Y to 64.



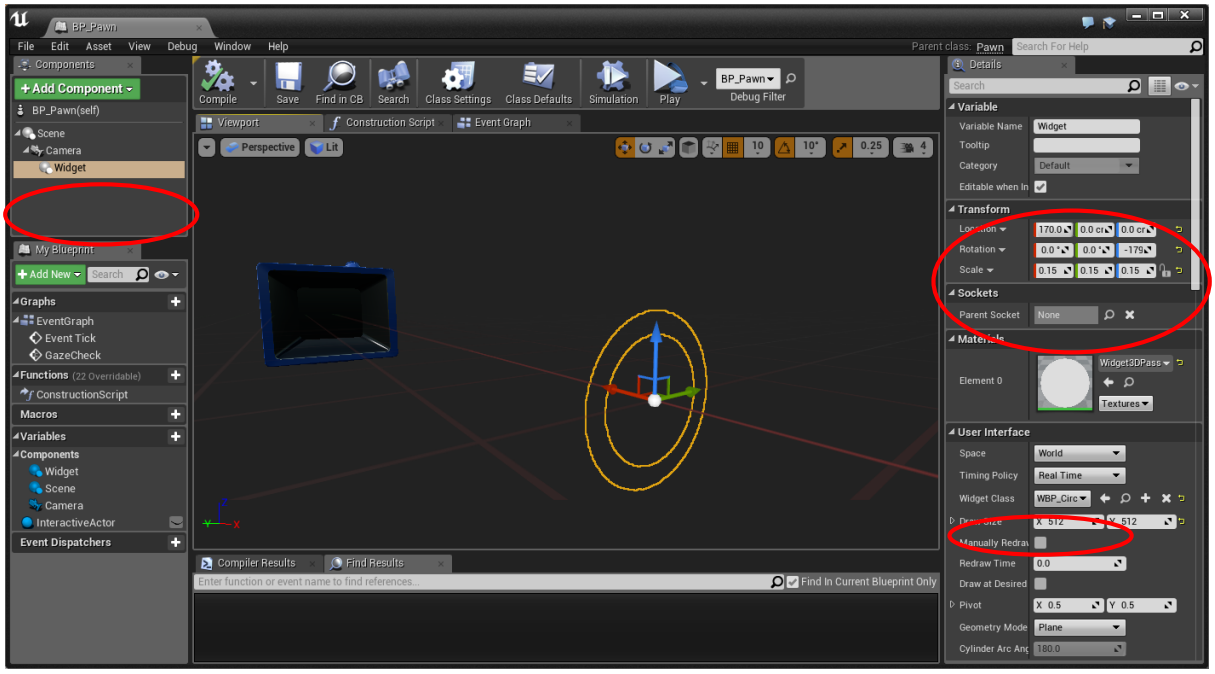
Change the Position X to -32 and the Position Y to -32. The square should now be centred with the anchor – refer example screen shot below.



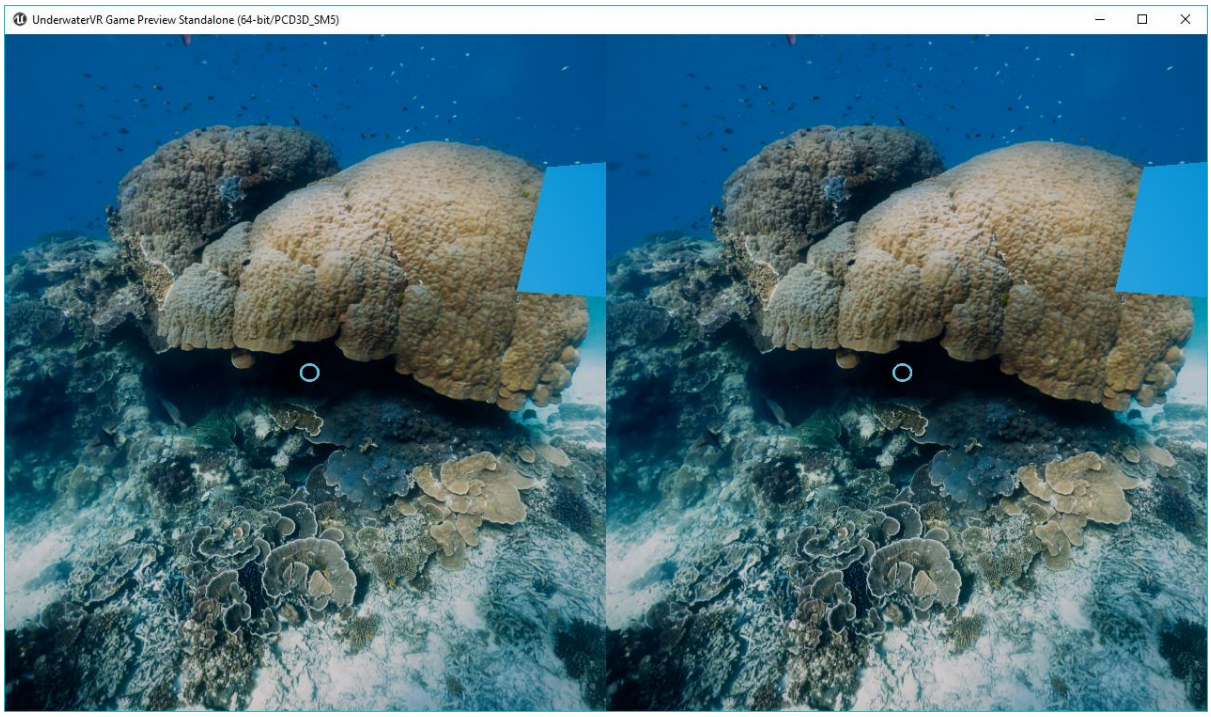
Next, select the “blue_circle” texture from APPEARANCE -> BRUSH -> IMAGE (refer screen shot below). Name the image “Circle” so that it appears as such in the hierarchy. COMPILE and SAVE.



Next, open the editor for the BP_Pawn blueprint. Click the green + ADD COMPONENT button and search/select WIDGET. In the USER INTERFACE settings on the right side, select the WBP_Circle as the WIDGET CLASS. Next move the widget – using the TRANSFORM -> LOCATION settings, move it x-axis only (x, 170). In the TRANSFORM -> ROTATION, rotate on z-axis only (z, 180). Finally, scale the widget down to: (x, y, z) (0.15, 0.15, 0.15). COMPILE and SAVE.



Build and test the app within Unreal Engine – you should see your blue-circle gaze reticle move when you move the camera.



For documentation on the Widget Blueprint Editor, see:
https://docs.unrealengine.com/latest/INT/Engine/UMG/UserGuide/WidgetBlueprints/index.html

NOTE: Before you can test this on the Android mobile device, you will need to deal with DPI ("Dots Per Inch") scaling – a means by which, depending on the resolution/dpi of the target screen, the DPI scaler will scale accordingly. Unreal Engine's UMG supports automatic scaling for our resolution-independent UI.

NOTE: When moving over an interactive object, an animation of the gaze reticle is possible as a way of providing visual feedback to the user. The following link is to the animation documentation of UI widgets: https://docs.unrealengine.com/latest/INT/Engine/UMG/UserGuide/Animation/

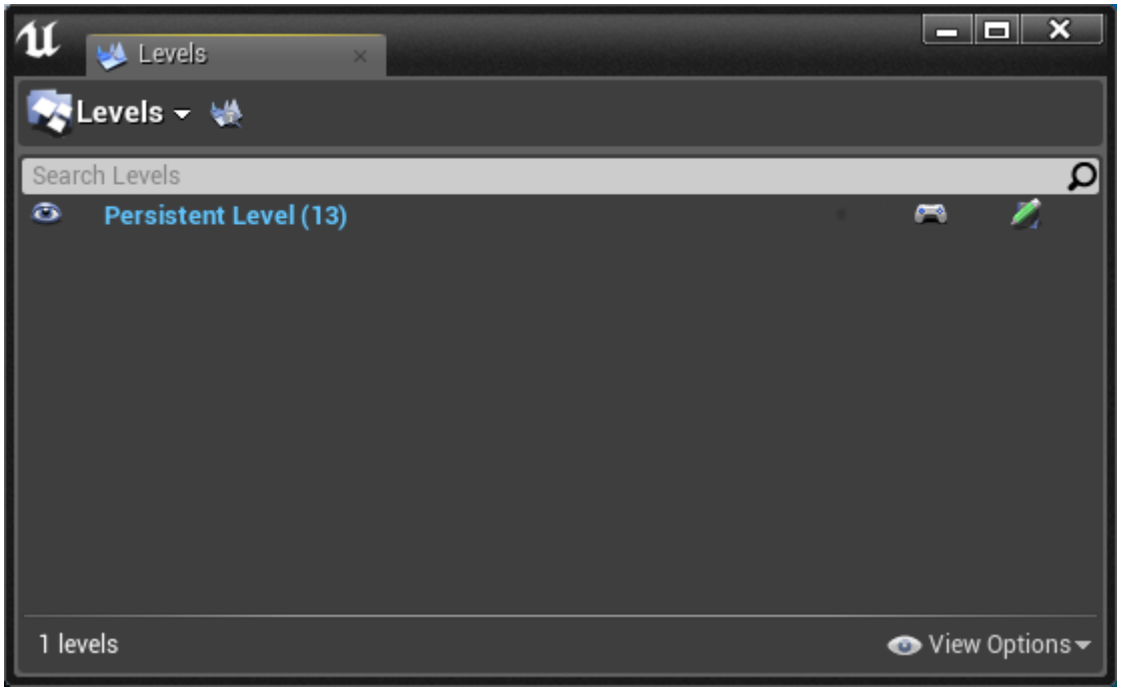
Another useful and insightful tutorial video on UI Animation:
https://www.youtube.com/watch?v=Urrig6FP4nY

8. Working outside of the Persistent Level

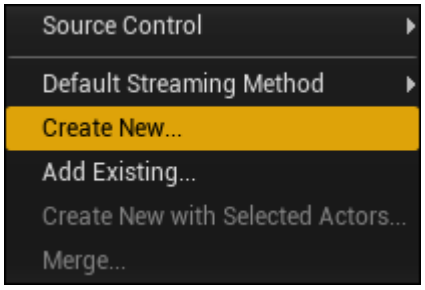
Everything so far has been created in what is known as the Persistent Level – something which can be thought of as the Scene root or master level. Additional levels and sub-levels can be set up to facilitate level streaming - the dynamic loading and unloading of sub-level game elements and scenes as needed.

This section will deal with the setting up of a sub-level in order to facilitate communication between different blueprints. To do this, we will need to set up a sub-level under the Persistent Level.

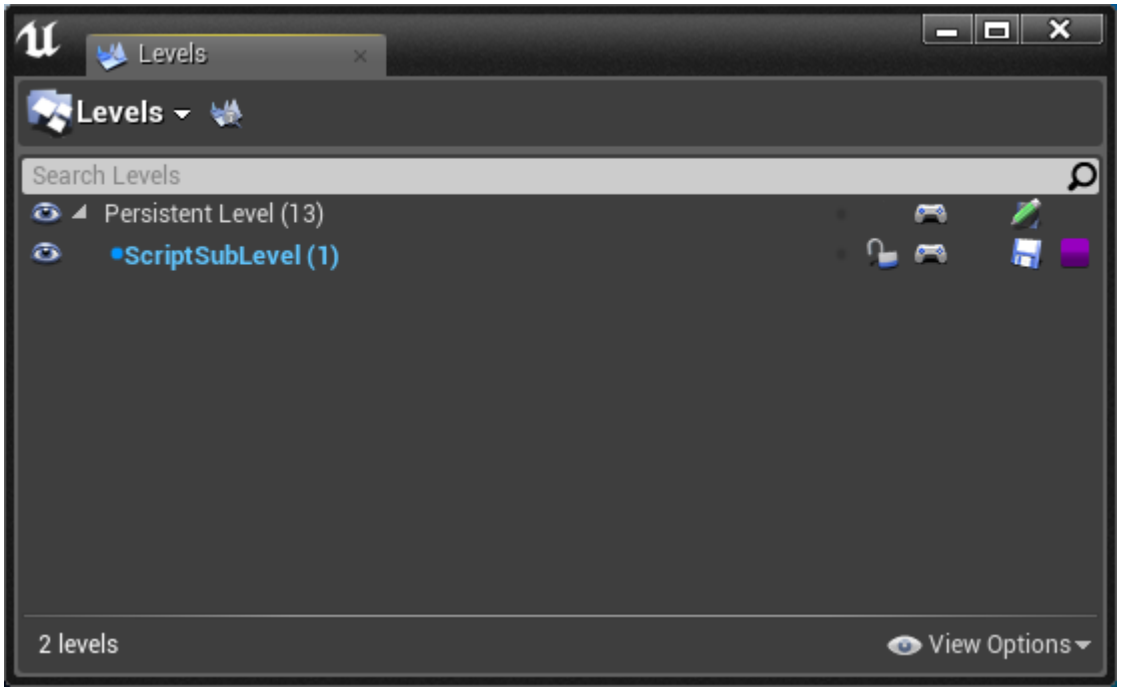
Go to WINDOW -> LEVELS



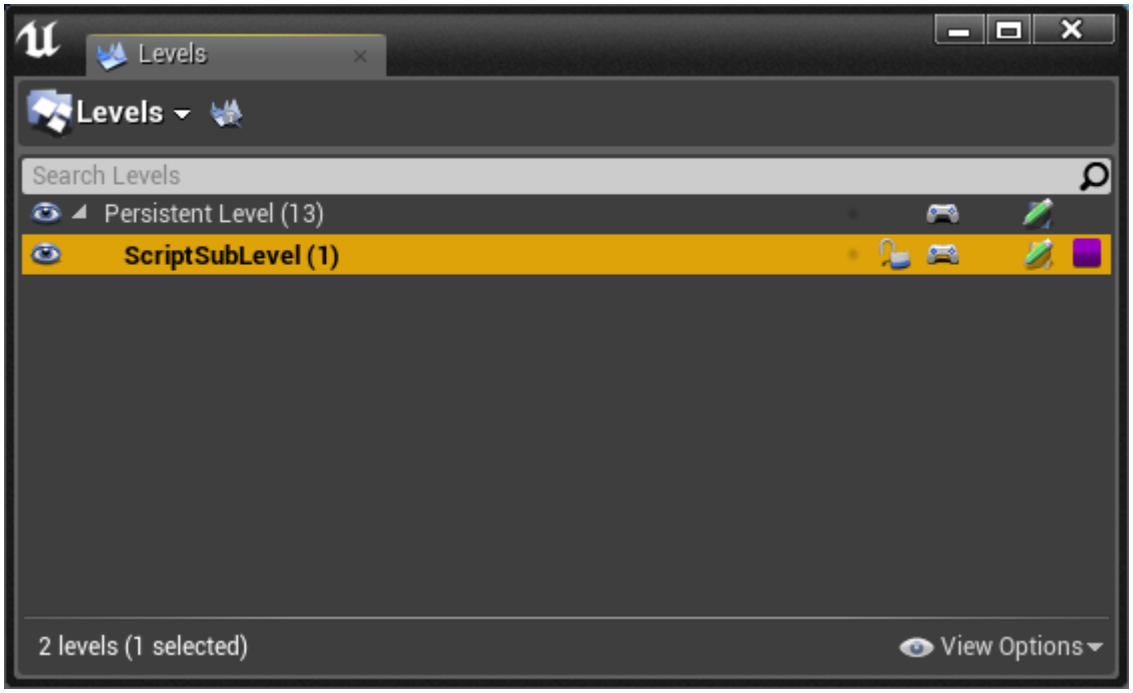
Select LEVELS and then choose CREATE NEW



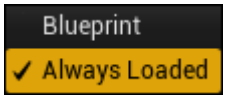
Name the new level “ScriptSubLevel” and click the SAVE button. This level will contain the blueprints that will need to communicate with one another.



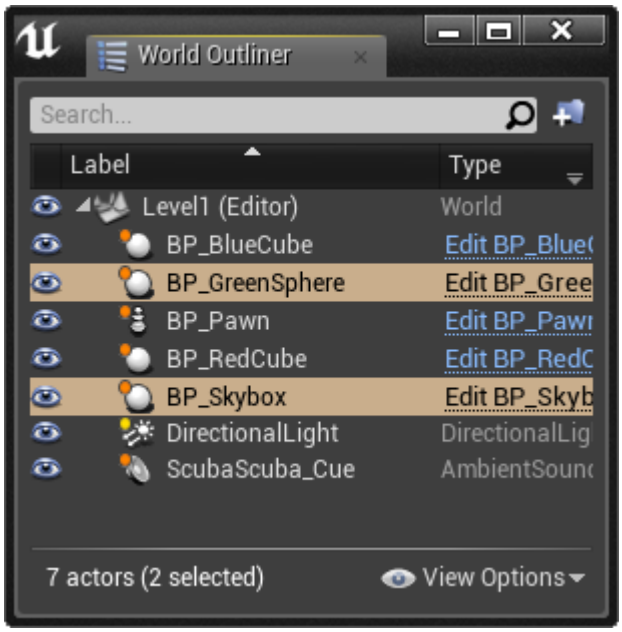
With the ScriptSubLevel selected,



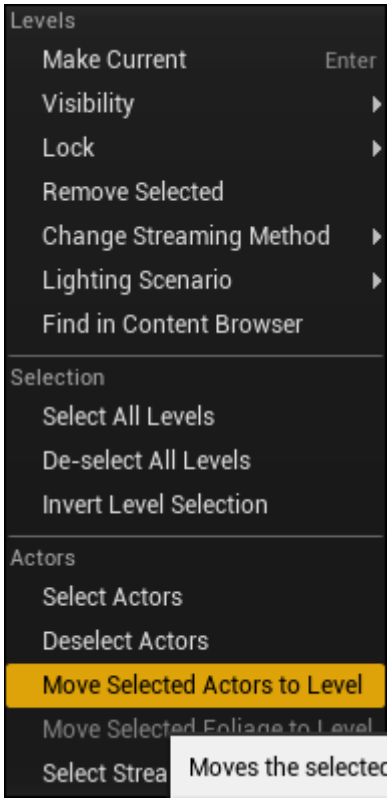
RMB-click and choose CHANGE STEAMING METHOD -> ALWAYS LOADED

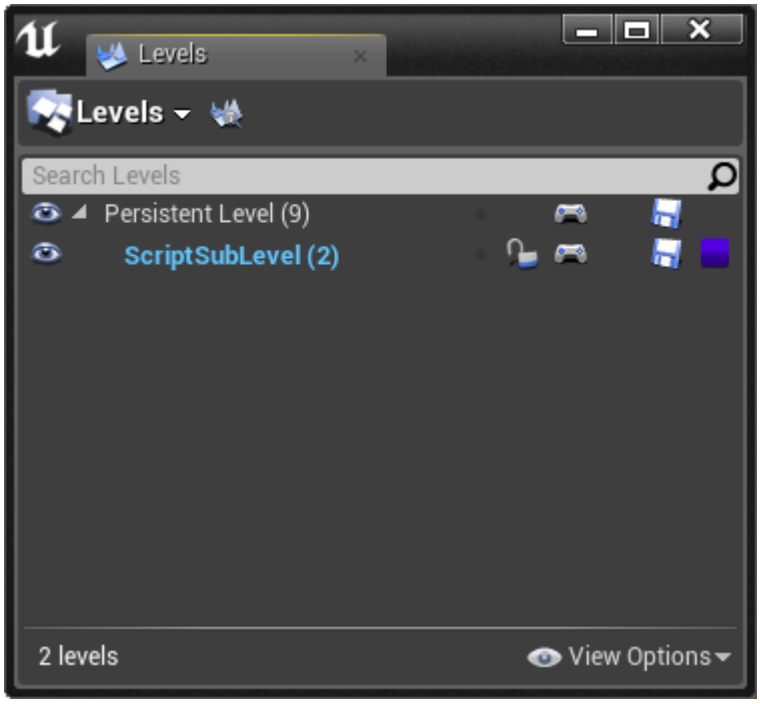


Select the BP_GreenSphere and BP_Skybox1 blueprints that you wish to set up inter-communication contact with.

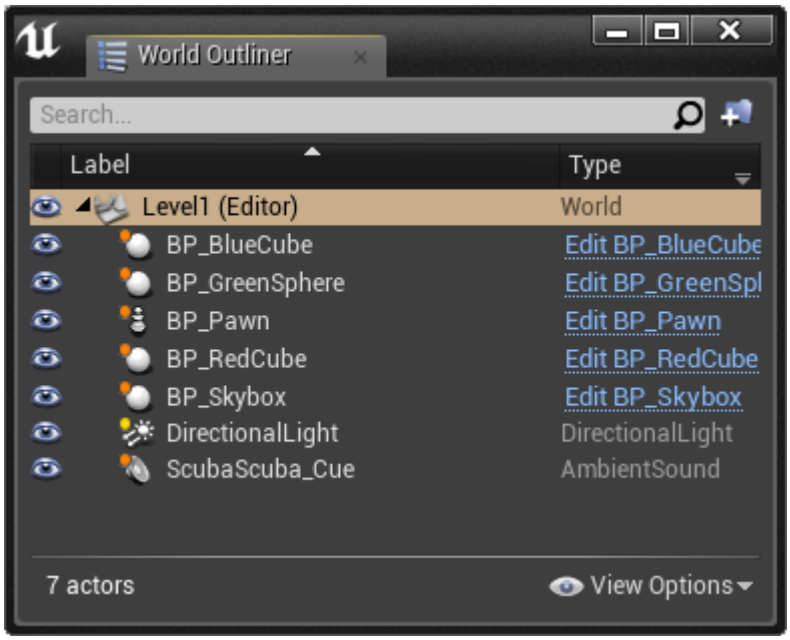


Next, RMB-click and choose MOVE SELECTED ACTORS TO LEVEL. This will move the two blueprint objects to the sub-level.





If you view the World Outliner, you should still see Level 1 with all components – only the BP_GreenSphere and the BP_Skybox1 are new part of the newly created sub-level.



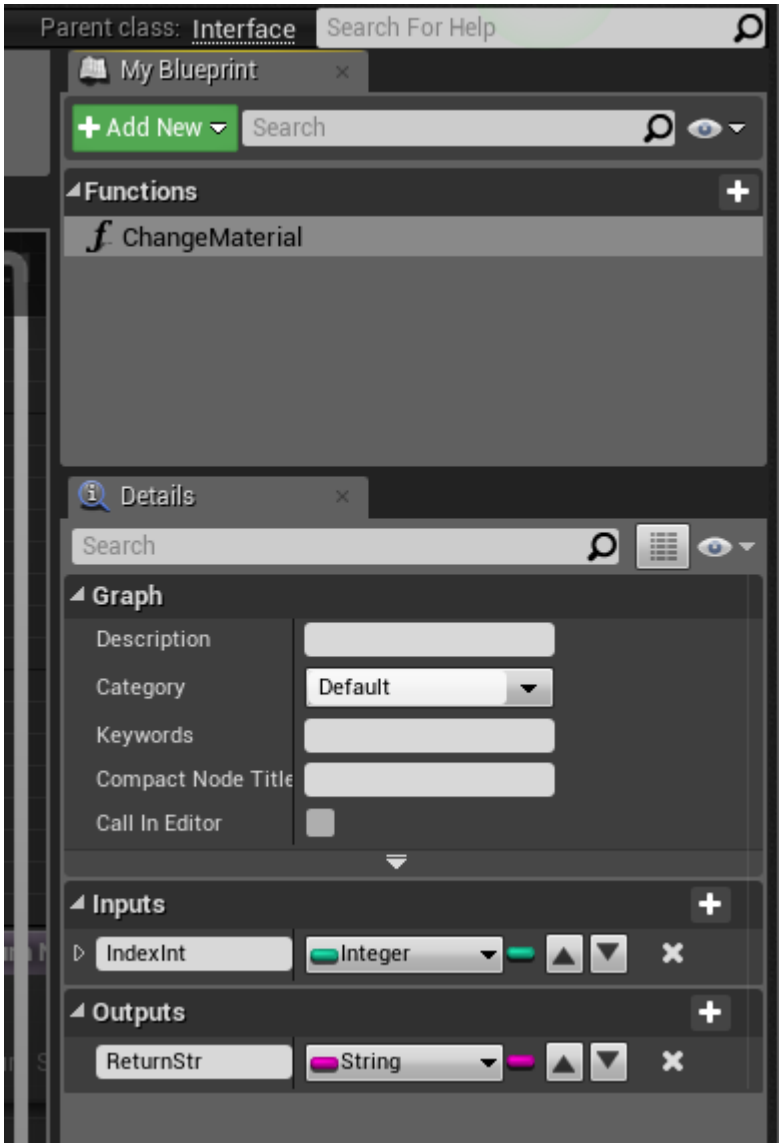
9. Creating a second interface and additional cubemaps

Create a second interface and name it “BPI_Material”. For this interface, create a function and name it “ChangeMaterial”.

For this function, create an input and name it “IndexInt” with data type of integer.

Also, create an output and name it “ReturnStr” with data type of string.

COMPILE and SAVE.



Open the BP_Skybox1 editor window and click the CLASS SETTINGS – implement the new interface “BPI_Material”.

Next, add two variables – a string array named “StringArray” and a Material array named “SkyMaterial” and ensure that the instances of both variables are editable.

For the StringArray, add 4 array elements. In this example there are:

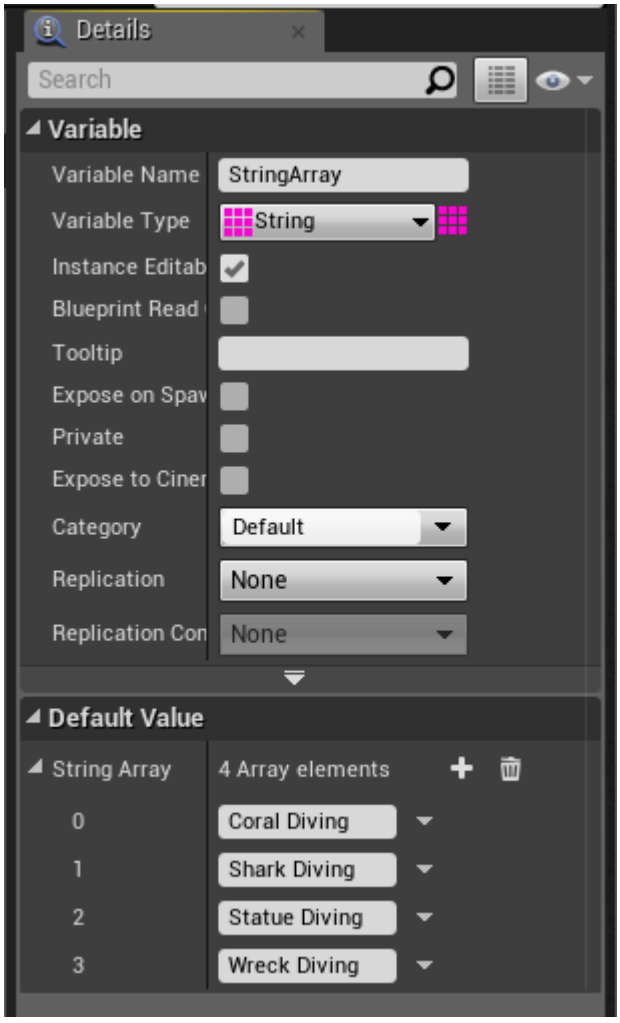
Index [0] --- Coral Diving

Index [1] --- Shark Diving

Index [2] --- Statue Diving

Index [3] --- Wreck Diving

NOTE: Arrays can be selected using the 3x3 grid icon to the right of VARIABLE TYPE.

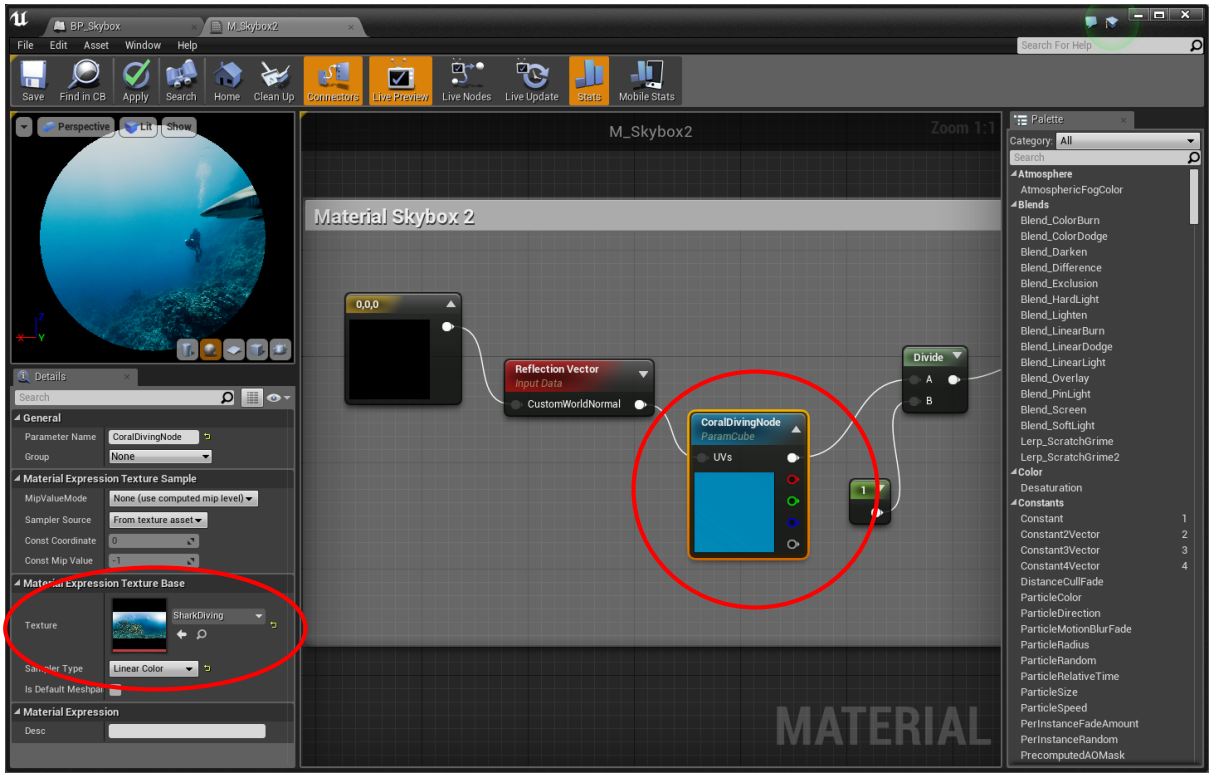


For the Material array named “SkyMaterial” … this will require 3 additional materials that you will need to create. For now, COMPILE and SAVE.

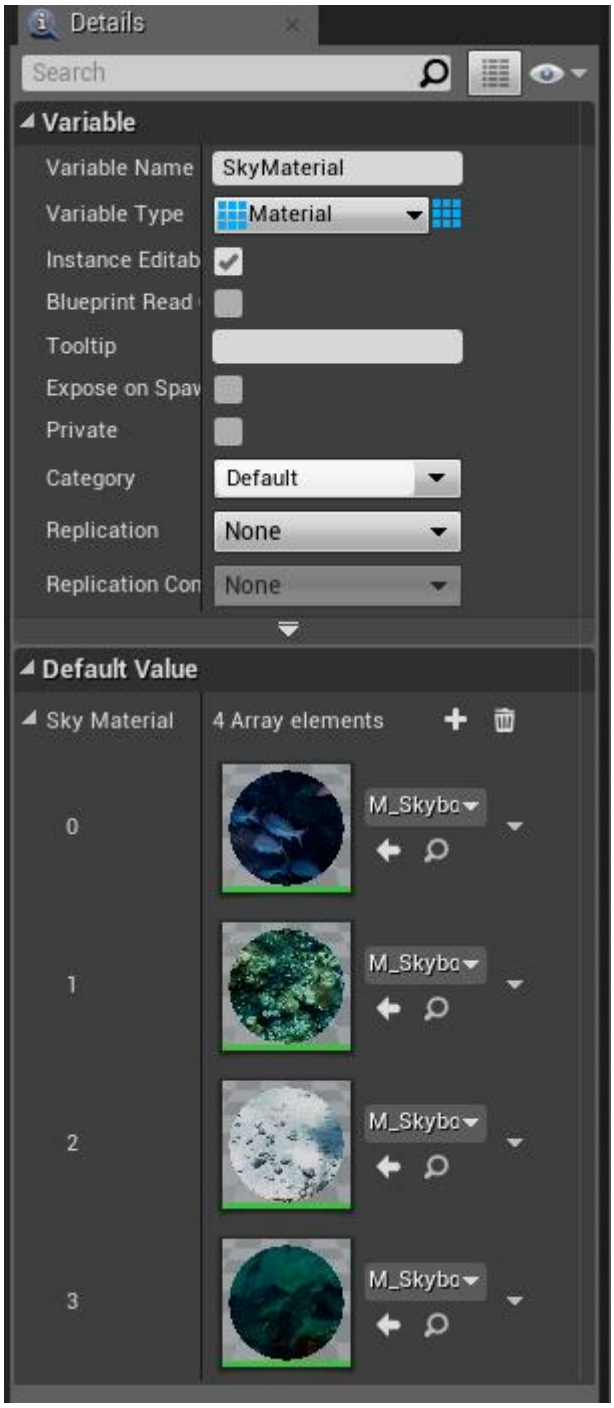
Go to the MATERIALS folder and RMB-click the M_Skybox1 material and select DUPLICATE. Name it “M_Skybox2”. Do it again another two times, naming the duplicates “M_Skybox3” and “M_Skybox4”.

For each of the duplicates, select the node that contains the settings for the texture. In the MATERIAL EXPRESSION TEXTURE BASE section, change the texture --- this assumes that you have already created a set of three additional textures (from three other 360-degree images).

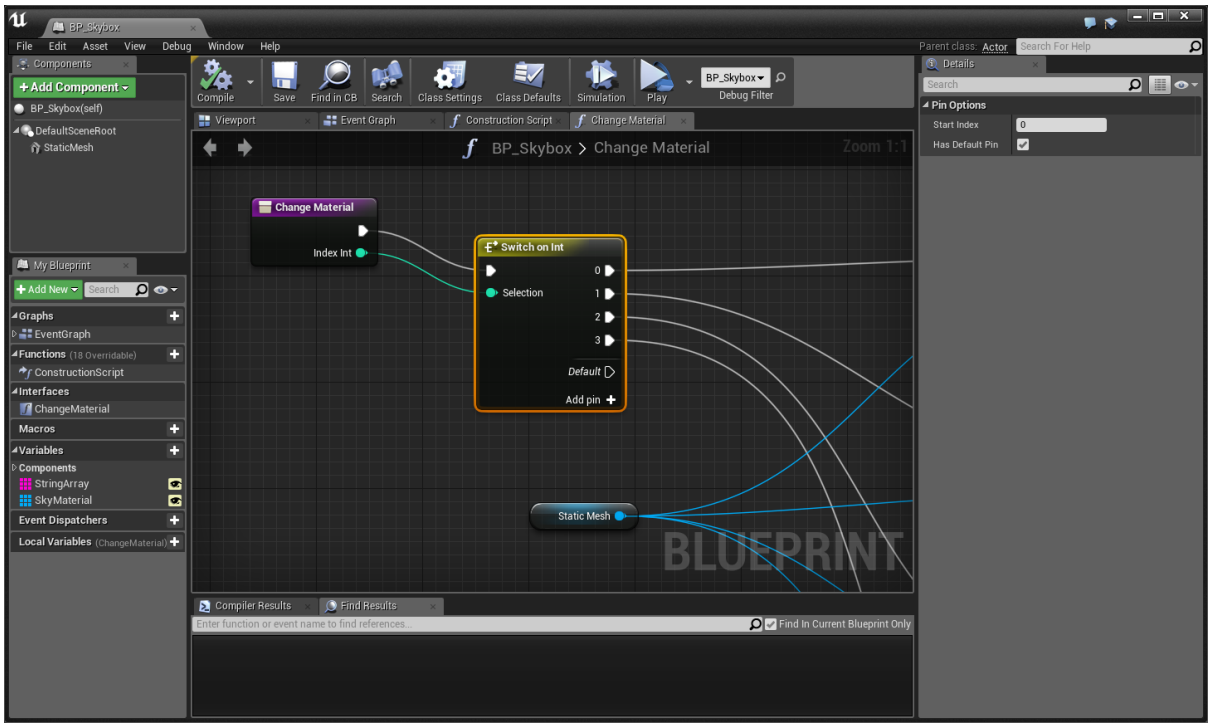
For each duplicate, click the APPLY and SAVE buttons.



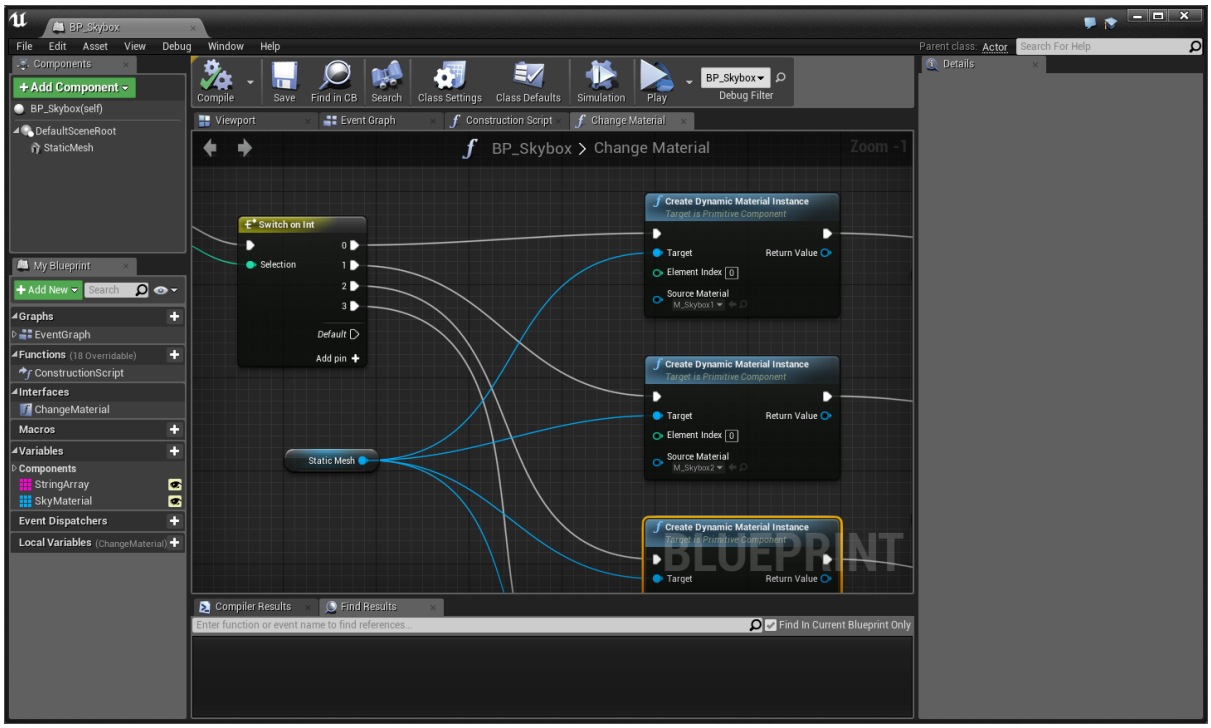
Go back to the BP_Skybox1 editor window and add the additional three materials into the Material array (SkyMaterial).



Open a new graph window for the ChangeMaterial() function. Drag out the white output pin of the CHANGE MATERIAL node and search/select SWITCH ON INT. Ensure that the green Index Int output is connected to the Selection input pin of the SWITCH ON INT node. Select and drag the STATICMESH into the graph.



Next, drag out the pin of the 0 output (SWITCH ON INT node) and search/select the CREATE DYNAMIC MATERIAL INSTANCE node. Connect the STATIC MESH node to the blue TARGET input pin of this DYNAMIC MATERIAL node. Keep the element index at 0 (default). Select “M_Skybox1” for the SOURCE MATERIAL input. You will need to do this for the other three integer outputs – 1, 2 and 3.

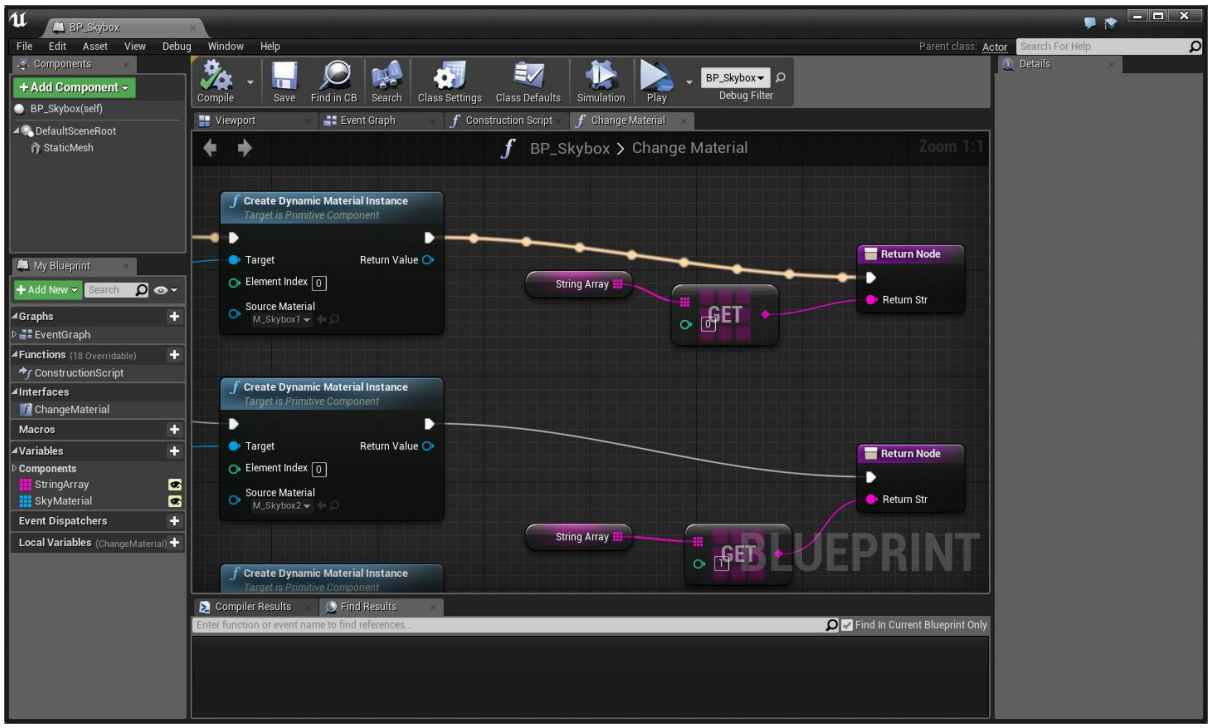


Each input SOURCE MATERIAL for the CREATE DYNAMIC MATERIAL INSTANCE will be a different material.

Next, drag out the white output pin of the DYNAMIC MATERIAL node to the white input pin of the RETURN NODE.

You will need to copy the RETURN node another three times, so that each output of the DYNAMIC MATERIAL node (depending on the integer input received) will be executed.

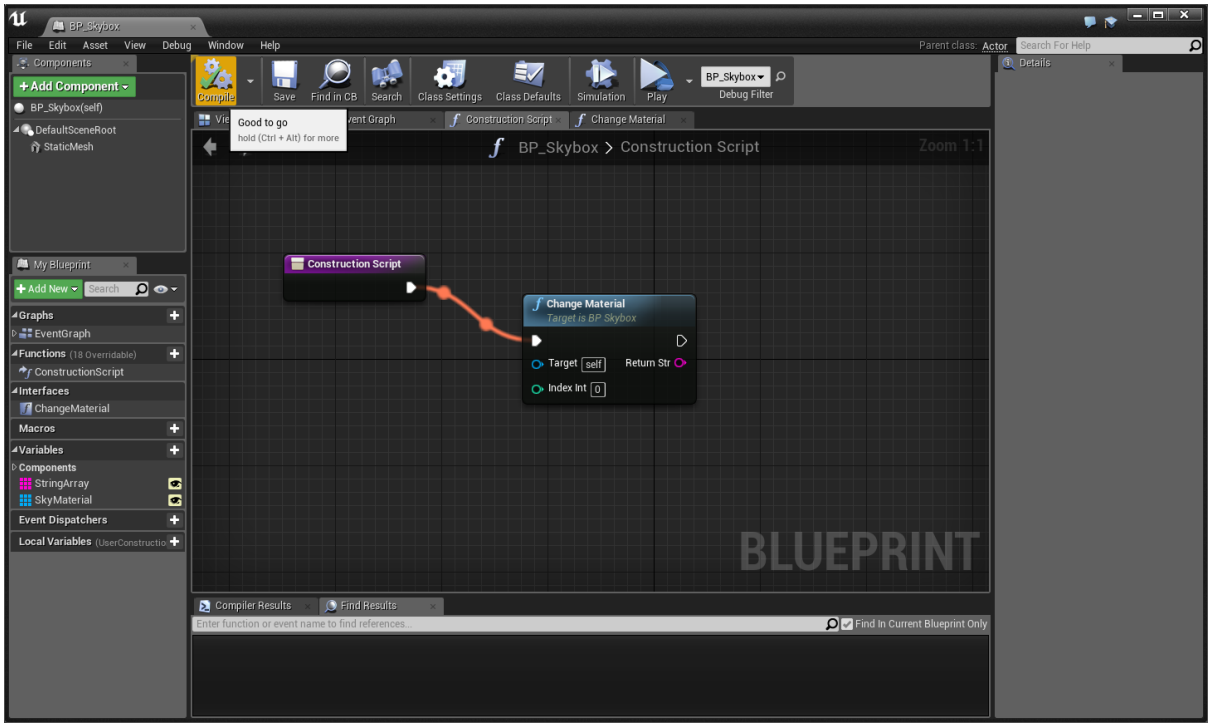
To output a string value, drag out the STRING ARRAY component (choose GET) and then drag the small output pin and search/select GET (a Ref). Ensure that the value of 0 (being index [0]) is selected. The output pin of the GET node will need to be connected to the input pin RETURN STR of the RETURN NODE. Refer to screen shot example below.



For the remaining three DYNAMIC MATERIAL nodes, do the same, but for each GET value, you will need to use the values of [1], [2] and [3] to represent the index values for the STRING ARRAY.

COMPILE and SAVE. This finalises the CHANGEMATERIAL () function.

Finally, open the CONSTRUCTION SCRIPT and drag out the white output pin of the CONSTRUCTION SCRIPT node. Search/Select the function call CHANGE MATERIAL. The target input is SELF and the INDEX INT value is 0 by default. The RETURN STR can be optional. COMPILE and SAVE.



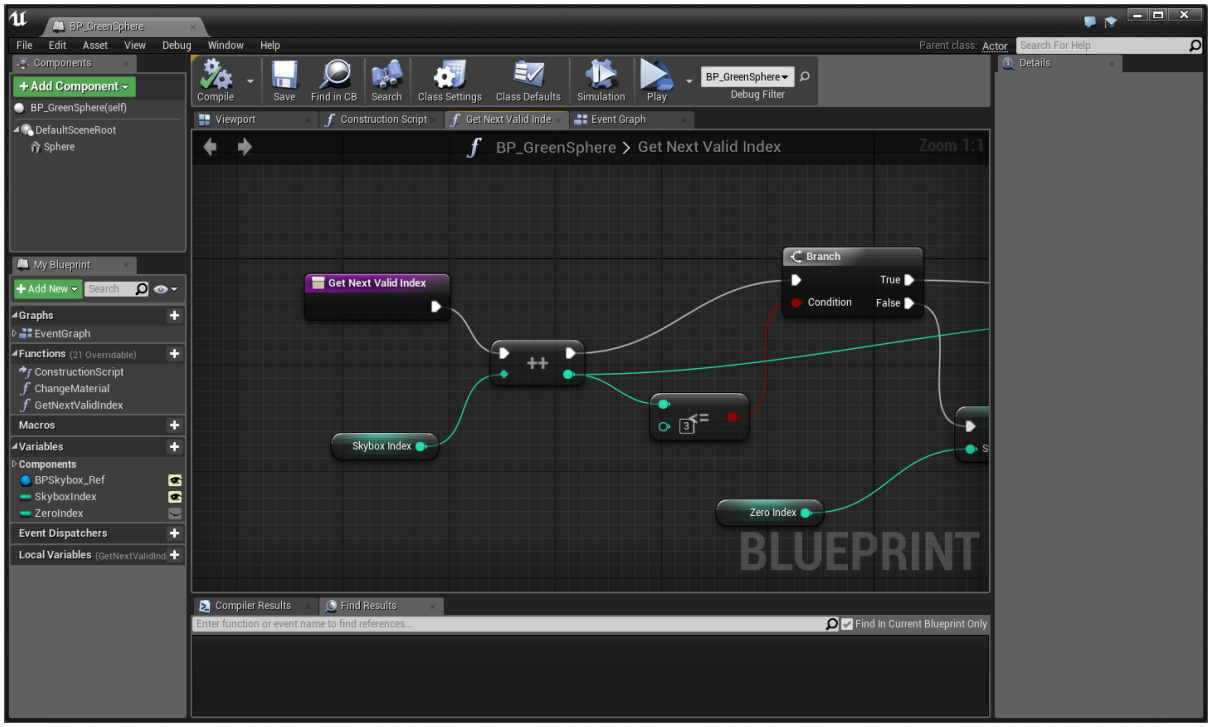
Open the BP_GreenSphere editor window and add a new function – name it “GETNEXTVALIDINDEX”.

Add a new variable – named “SkyboxIndex”. Make the data type an INTEGER and make it editable. Compile it and set up the DEFAULT VALUE of 0.

Add a second variable – named “ZeroIndex”. Make the data type an INTEGER and do not make it editable. Compile it and set up the DEFAULT VALUE of 0 as well.

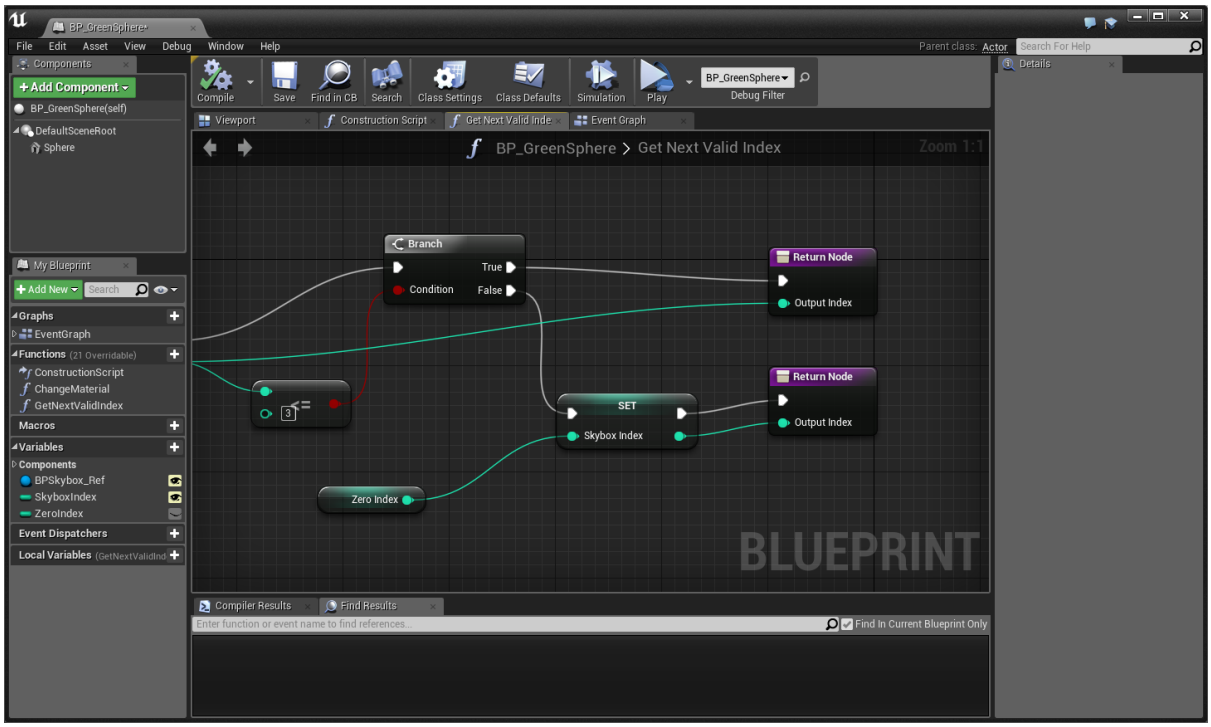
Open a new graph window for the new function GETNEXTVALIDINDEX.

Drag into the graph the SKYBOXINDEX (GET). Select the white output pin of the GET NEXT VALID INDEX node and search/select ++ (incrementation) node. Connect as shown in the screen shot example below. Next, drag out the white output pin of the ++ node and search/select for BRANCH node. Drag out the output green pin of the ++ node and search/select the <= node. Input the value of 3 in this node (3 is the last acceptable index value for the array). Connect the red pins of the <= and Branch nodes (refer screen shot below.



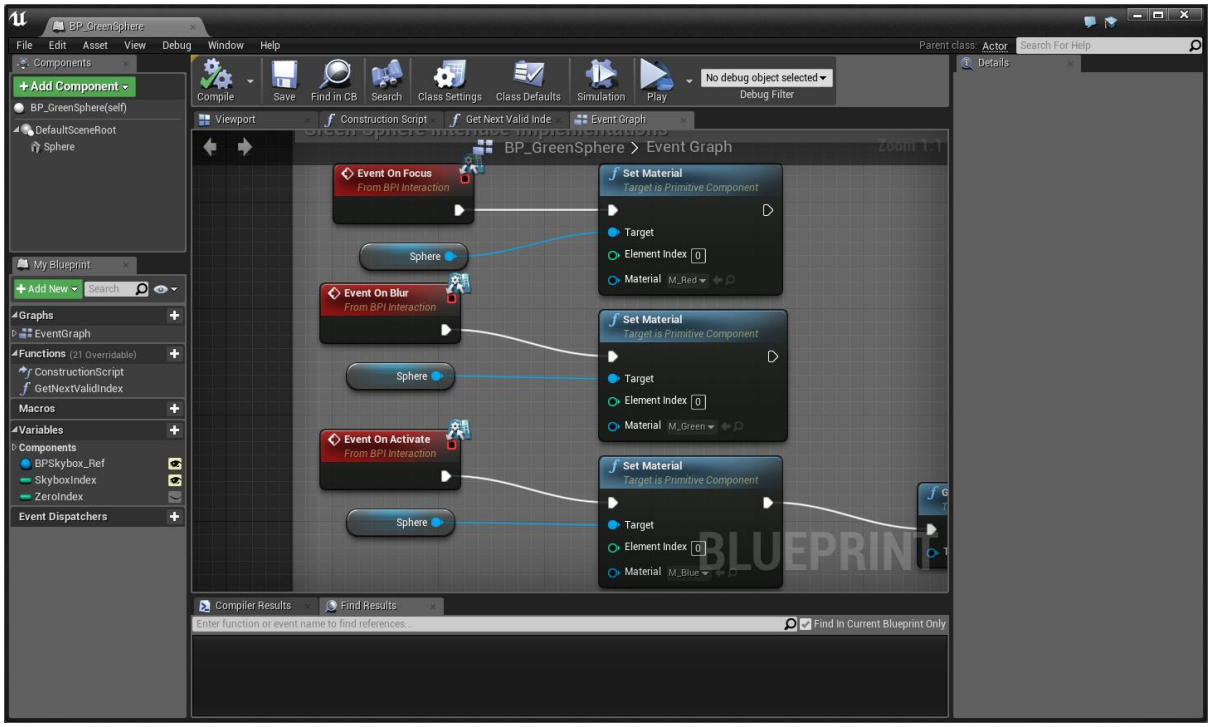
NOTE: In the Variables section (bottom left), you will see a yellow eye icon for a variable and an eye closed icon for another variable. The eye open means that the variable is public and can be editable external to the class. The eye closed means that the variable is private and only editable within the class.

Next, drag the ZERO INDEX (GET) into the graph. Search/Select the SET Skybox Index node. This SET node will need to be connected from the FALSE output pin (and the green pin connected from ZERO INDEX). Finally, you will need two RETURN nodes connected in the manner shown in the example screen shot below.

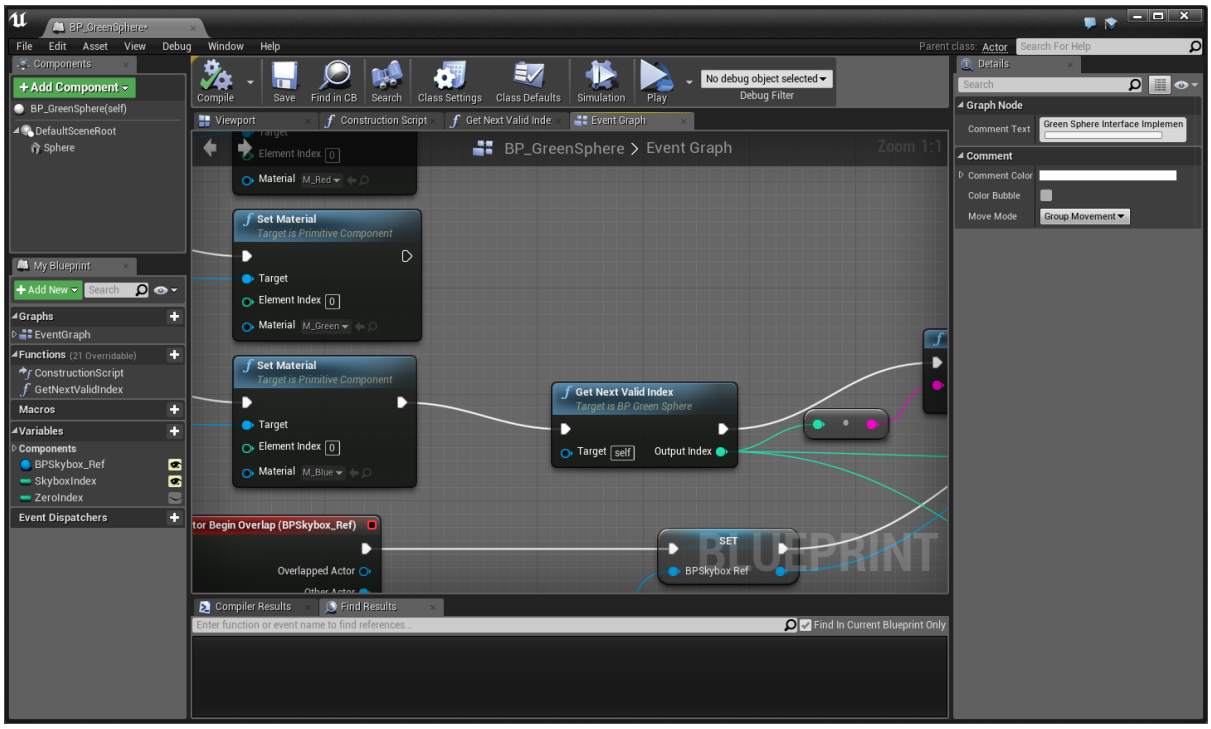


COMPILE and SAVE.

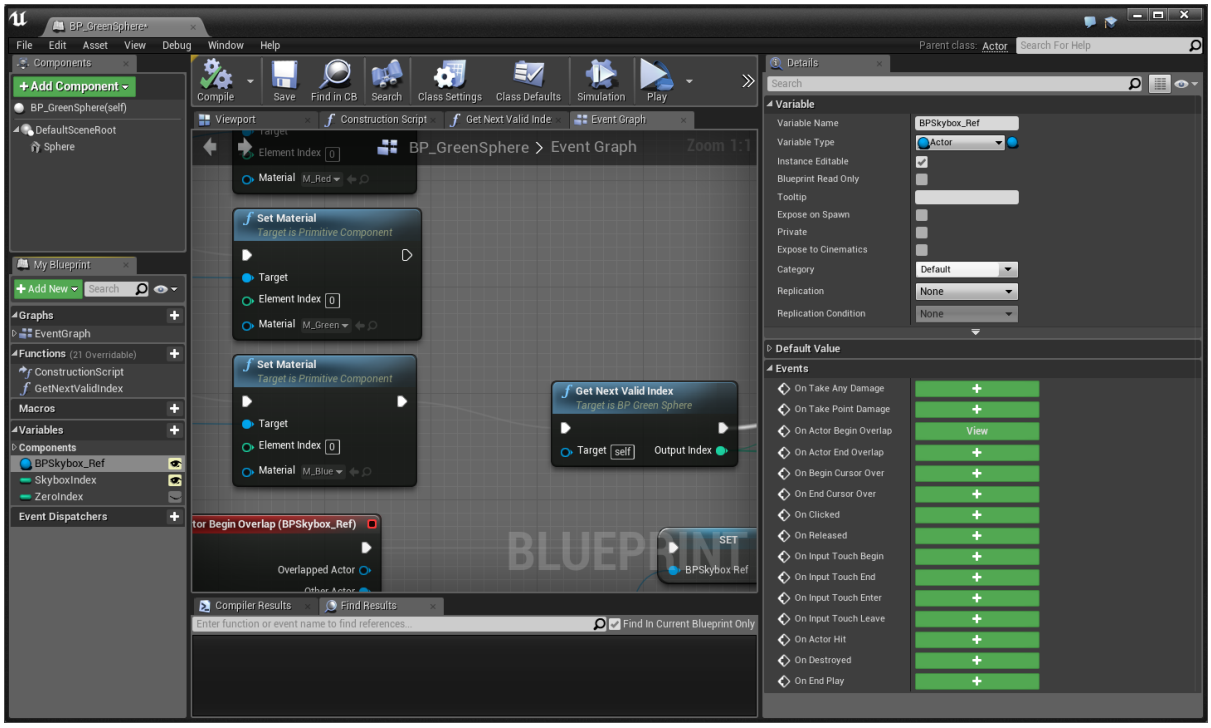
Open the EVENT GRAPH of the BP_GreenSphere and change the EVENT ON FOCUS, EVENT ON BLUR and EVENT ON ACTIVATE so that the SET MATERIAL node is used. The sphere is connected as the target input pin for each of the SET MATERIAL nodes – refer screen shot example below.



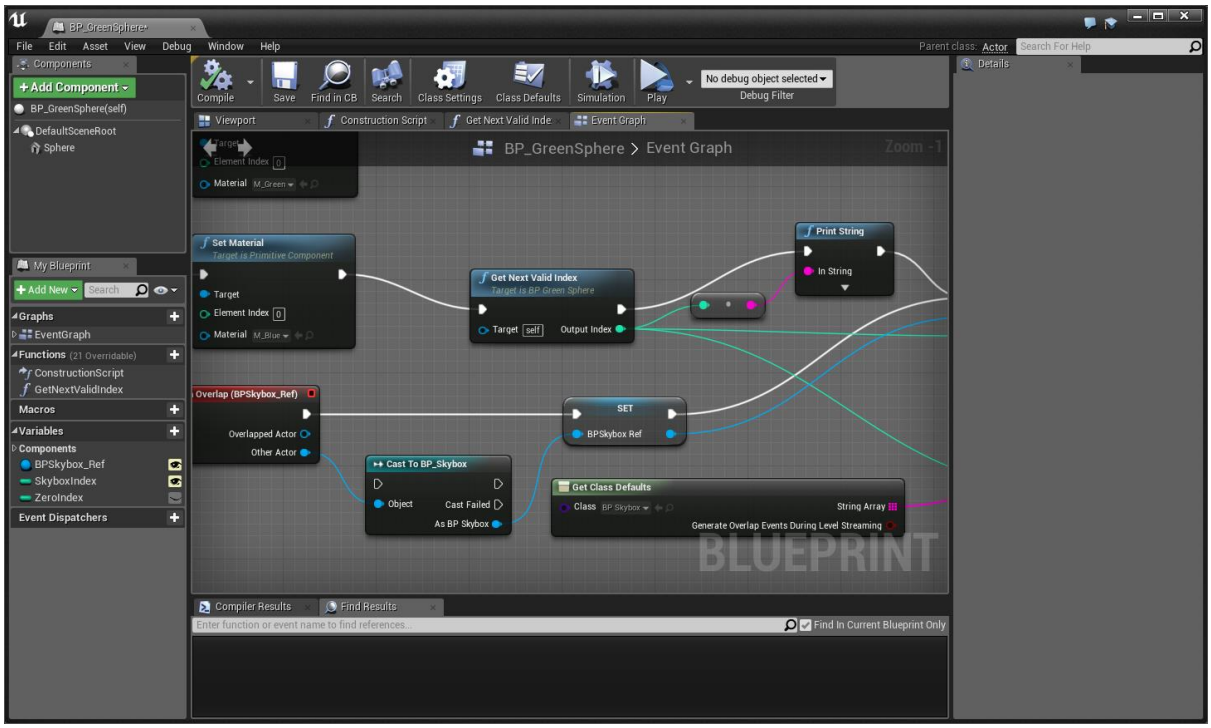
Drag out the white output pin of the SET MATERIAL node (connected from the EVENT ON ACTIVATE node) and search/select the GET NEXT VALID INDEX function call. The target is SELF as this function implementation is part of this blueprint.



Create a new variable and name it “BPSkybox_Ref” … this will be a blueprint skybox reference (the one containing the ‘sky’ material, for which we want to change). In the DETAILS section (top right), change the VARIABLE TYPE to ACTOR and make it editable. In the EVENTS section (just below the VARIABLE and DEFAULT VALUE sections), click the green + button for ON ACTOR BEGIN OVERLAP. This will set up a node like the example shown below.

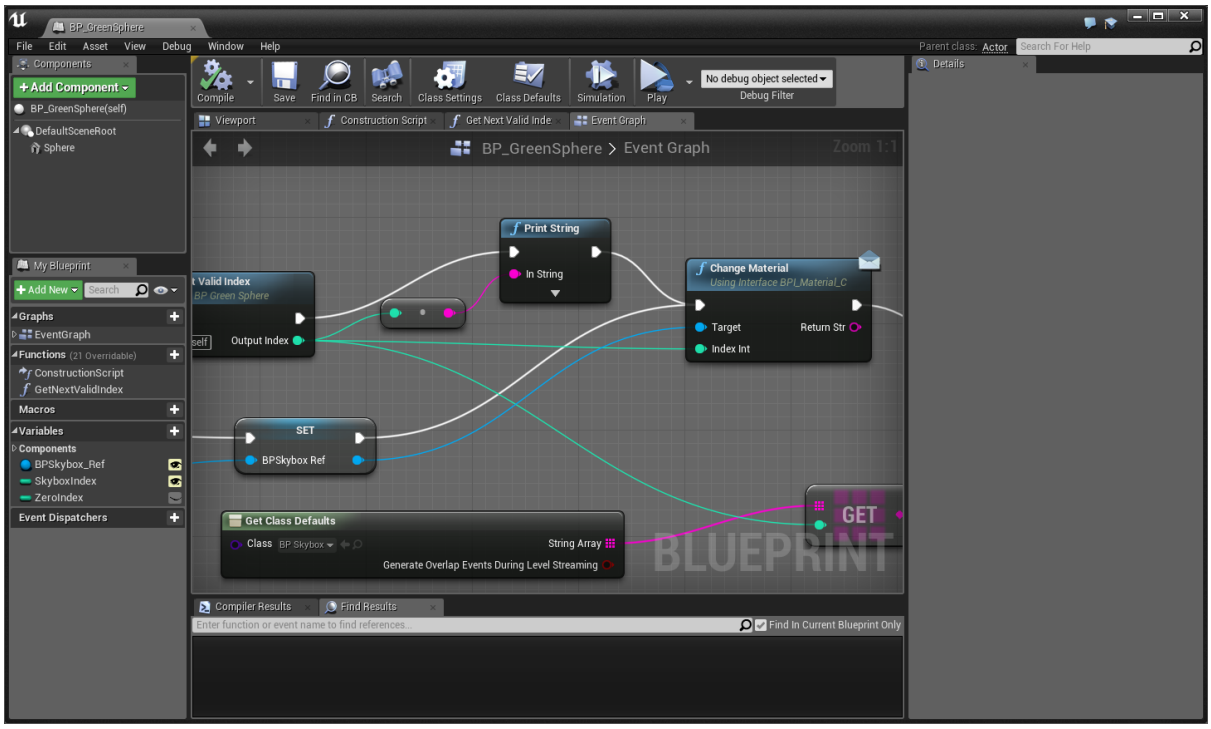


Drag out the white output pin of the ON ACTOR BEGIN OVERLAP node and search/select the SET BP Skybox Ref (SET). From the OTHER ACTOR output pin of the OVERLAP node, select CAST TO BP_SKYBOX. Connect these three nodes in the manner shown below.



The screen shot also shows a PRINT STRING node coming from the OUTPUT INDEX pin of the GET NEXT VALID INDEX node. RMB-click anywhere and search/select GET CLASS DEFAULTS and choose the class BP Skybox (or BP Skybox 1) – the name of your current skybox in the scene. Drag out the STRING ARRAY output pin and search/select the GET (ref) node.

From the SET node of the BP Skybox Ref drag out and search/select the CHANGE MATERIAL (function call) node. Connect the nodes accordingly (refer screen shot below). To print the output of the CHANGE MATERIAL function, create the necessary PRINT STRING node.

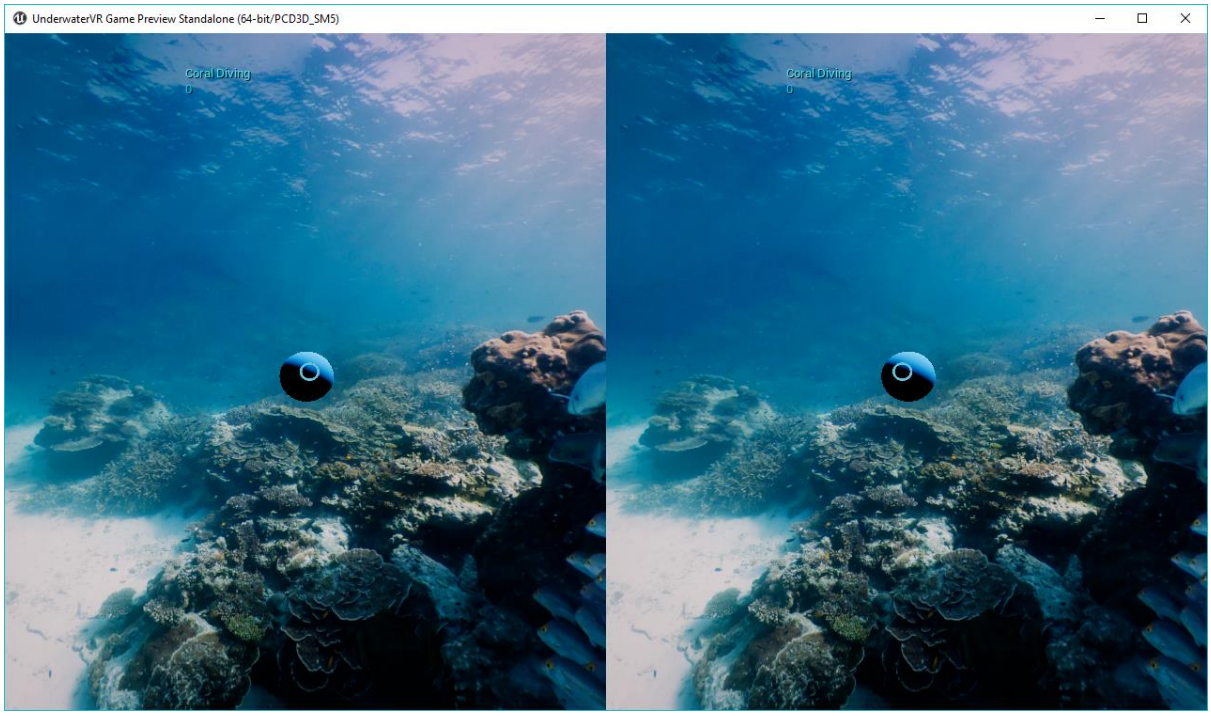


Useful tutorial video on the subject of Get Class Defaults:
http://www.mediarepeats.com/video/?id=ZbwEMnSrgWY

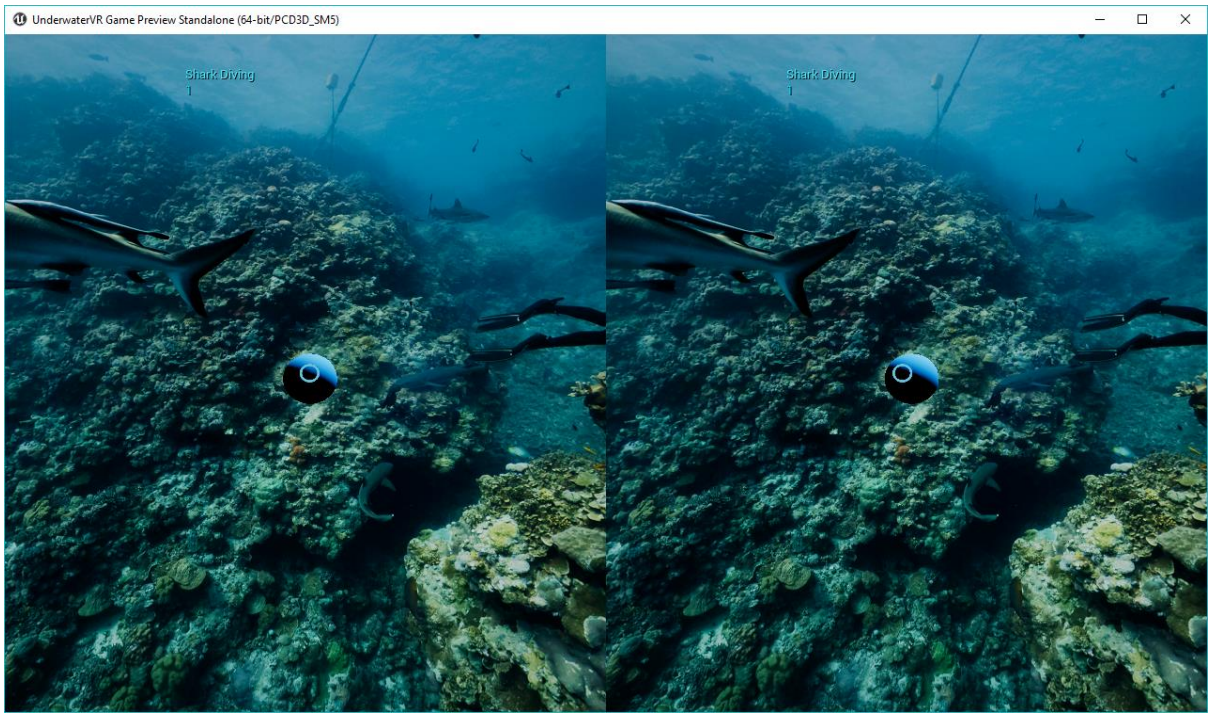
NOTE: The Get Class Defaults node does not currently expose object reference values by design; its current use is intended for exposing value types only. This is because Blueprints currently cannot enforce the read-only state of objects owned by the class default object that might be referenced through the return value, and the class default object's state MUST be treated as read-only. Consider a situation where in one function you use some value in the default object as a basis for initialization; there is nothing in place that can currently prevent another event or function from mutating that object at runtime, which can result in potentially unpredictable behaviour that can be difficult to debug.

Build and test the project. Moving the small reticle over the green interactive sphere should now allow you to change the material mapped to the BP_Skybox blueprint and display the index number and string value for each image.

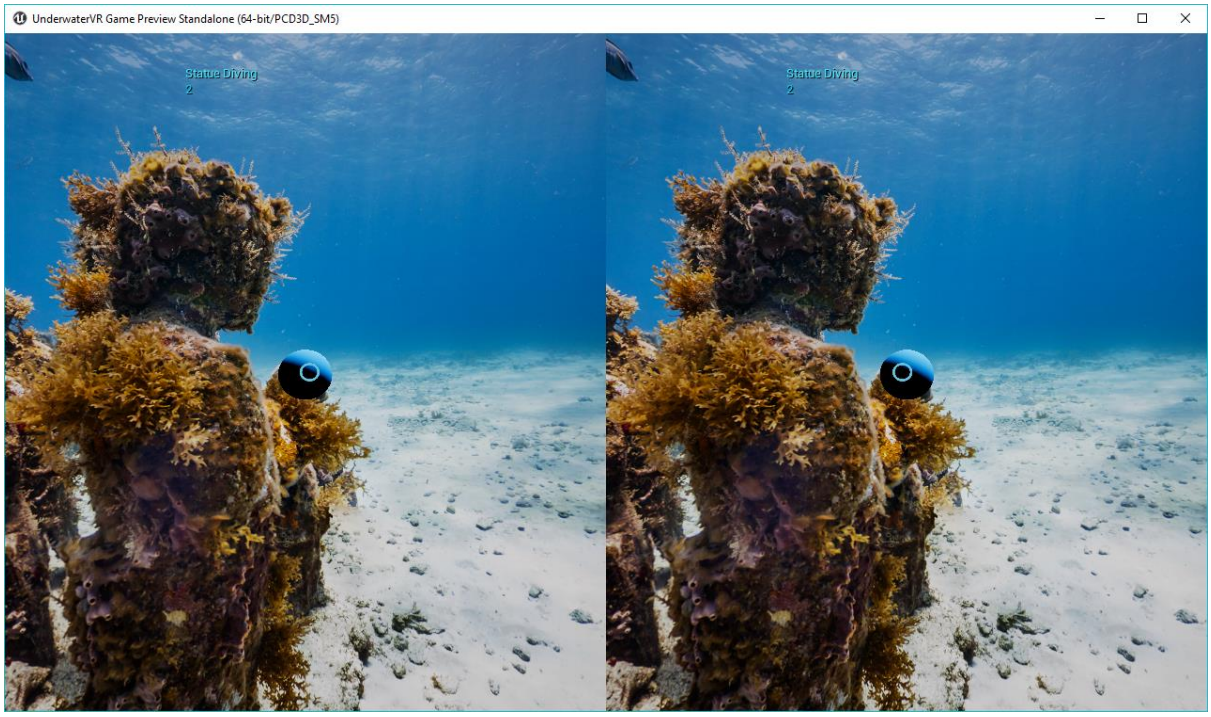
Index [0] --- Coral Diving



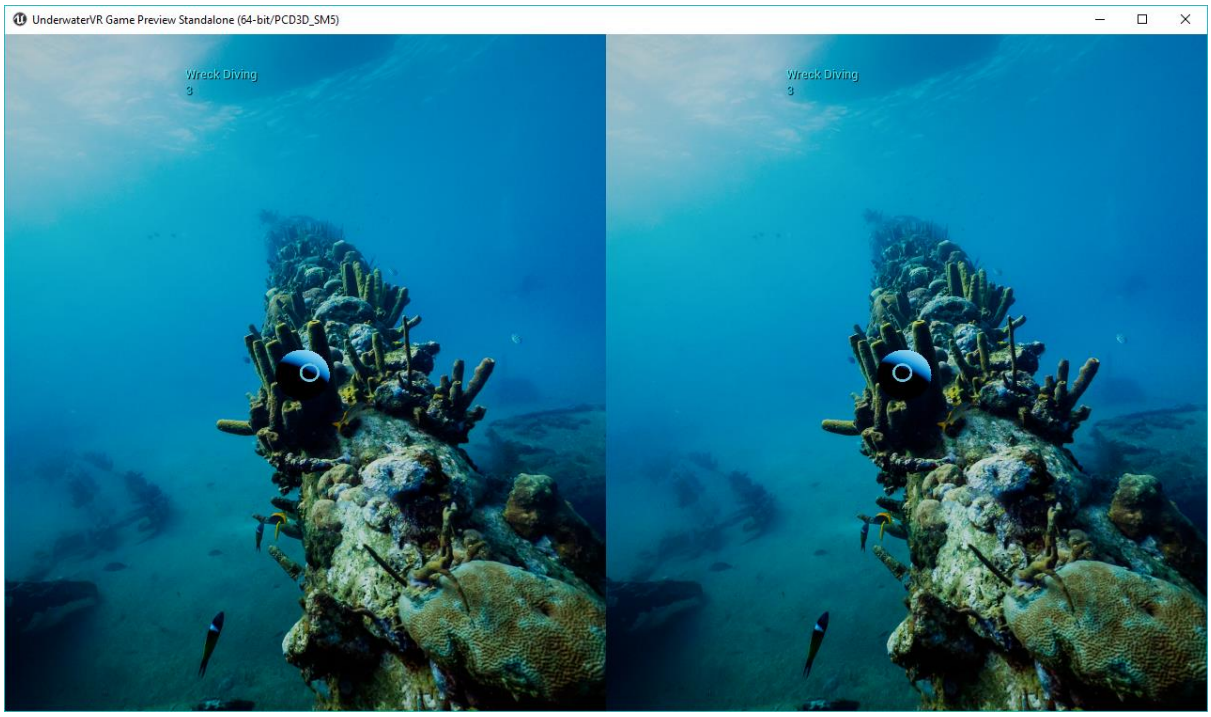
Index [1] --- Shark Diving



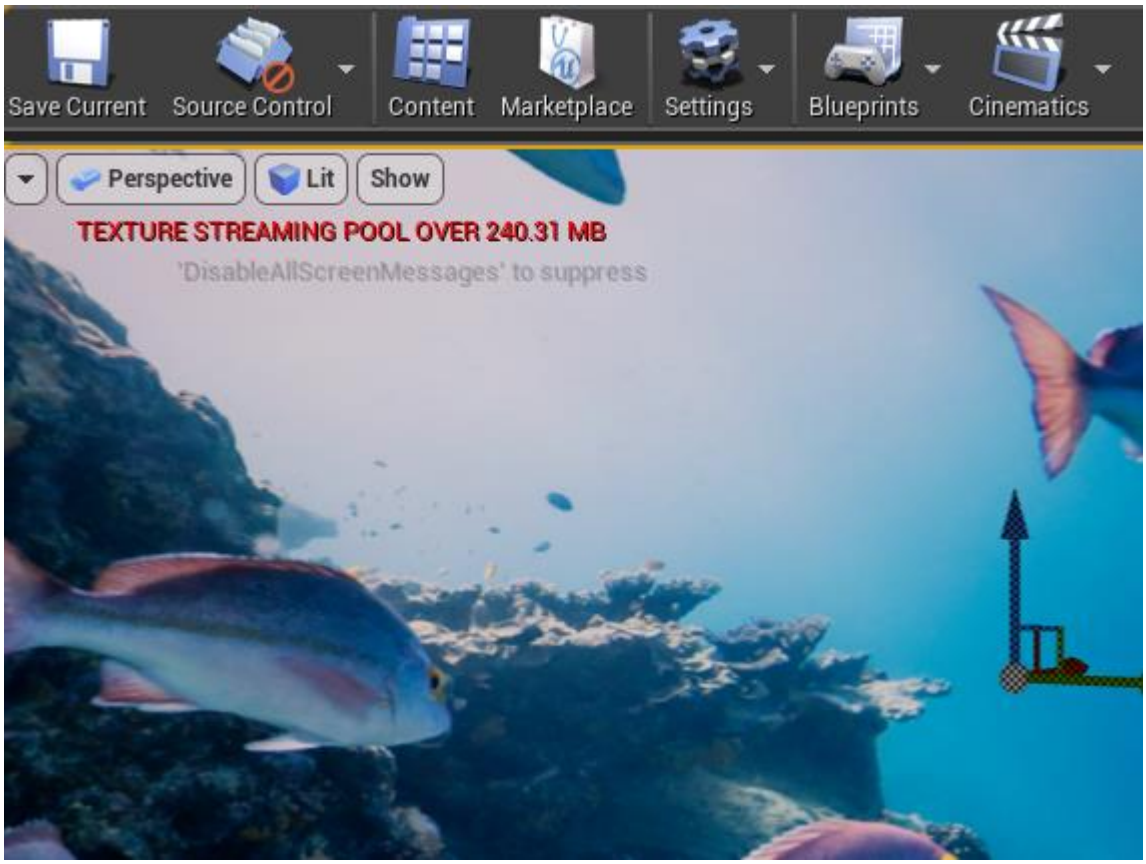
Index [2] --- Statue Diving



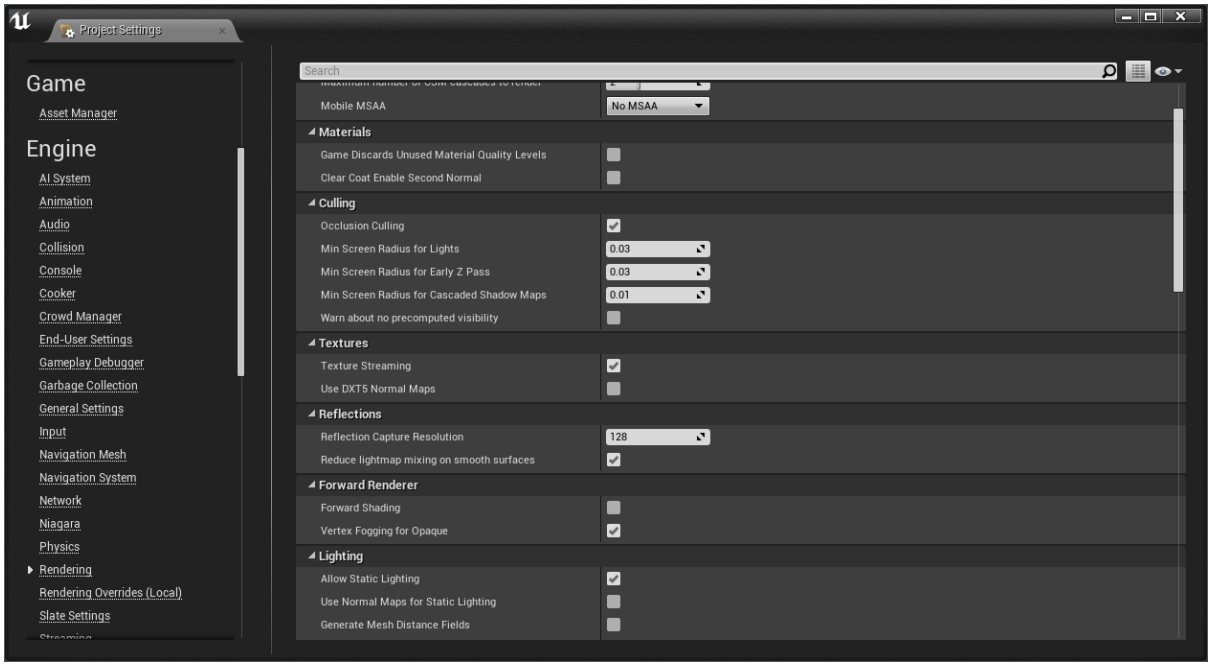
Index [3] --- Wreck Diving



NOTE: Potential texture streaming issues were raised when adding additional materials (beyond 4). This is because the graphics card used in this development had its capacity exceeded. Building and testing the app (containing 6 materials) on a Samsung S7 mobile phone did not cause problems, however, there could be issues with older models.



A possible way of dealing with this is to turn off texture streaming in the PROJECT SETTINGS (ENGINE -> RENDERING -> TEXTURES -> TEXTURE STREAMING). When this is enabled however, the textures will steam in for those that are visible on the screen. This was not attempted in this exercise.



10. Tutorial Videos and Documentation Links

Useful youtube videos

Mitch's VR Lab – Ep02: Introduction to Gaze-based interaction in Unreal Engine

Part 1: https://www.youtube.com/watch?v=X-7Hu0HzbDU (Setting up Gaze-based interaction Intro)
Part 2: https://www.youtube.com/watch?v=7z0zksCCmU0 (How to get the HMD World Location)
Part 3: https://www.youtube.com/watch?v=hqy_zGfEMP8 (Adding look assistance)
Part 4: https://www.youtube.com/watch?v=slIdcuIQWzM (Optimisations of previous code)
Part 5: https://www.youtube.com/watch?v=Wywwtbync70 (Creating a simple puzzle mechanic)
Part 6: https://www.youtube.com/watch?v=KZOvJtrzTKA (Creating a time-based interaction)

Strigifo – Mobile VR Tutorial Series

Part 1: https://www.youtube.com/watch?v=zLWV866RDvo (Android Project Setup)
Part 2: https://www.youtube.com/watch?v=_AEE7v164Ns
(Specific Project settings for Google Cardboard and Google Daydream)
Part 3: https://www.youtube.com/watch?v=p3LYkg8WiFo (Debugging Packaging Issues Update)
Part 4: https://www.youtube.com/watch?v=TeOaThzS51w (Gaze Tracking)
Part 5: https://www.youtube.com/watch?v=M3G5IfCI9zE (Daydream VR Motion Controller)
Part 6: https://www.youtube.com/watch?v=VErMT_vwrk4 (Mobile VR Google GitHub download)
Part 7: https://www.youtube.com/watch?v=L0xQsfR3Tn4 (Mobile VR Movement)
Part 8: https://www.youtube.com/watch?v=z20MihLzJF4 (Blueprint Interfaces)
Part 9: https://www.youtube.com/watch?v=5UepemBBHws (Mobile VR Teleporting)

FRAG Level Design (Oliver Curtis) – Space Skybox Creation Tutorial

Intro: https://www.youtube.com/watch?v=JSRsQpRfDlk (Creating a space skybox)

Line Trace by Channel Node in Unreal Engine
https://www.youtube.com/watch?v=yo9VrxFgUJY

Line Trace for Objects in Unreal Engine
https://www.youtube.com/watch?v=6ZFnPliFSX0

What is a Blueprint Interface?
https://www.youtube.com/watch?v=G_hLUkm7v44