BUILDING A PARACHUTE DROP SIMULATOR

ABSTRACT

A tutorial on how to Install Cesium onto an Unreal project based on tutorial available here: https://cesium.com/learn/unreal/unreal-quickstart/

Grace Malabanti, Joe Scheufele, and Juliette Spitaels

Table of Contents

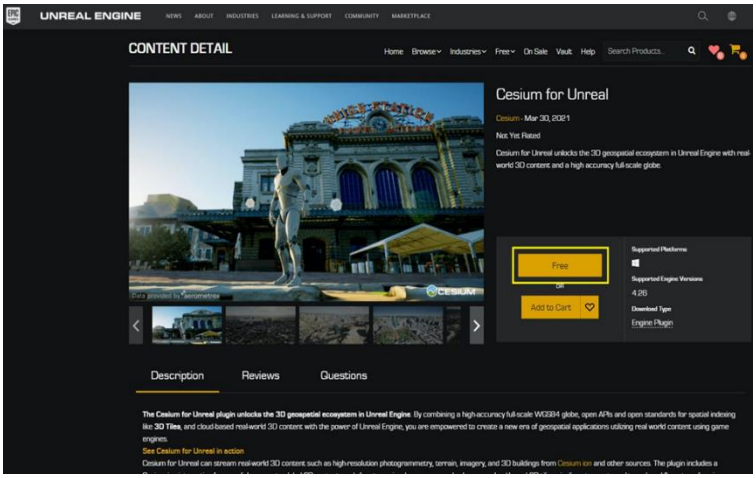
Prerequisites:

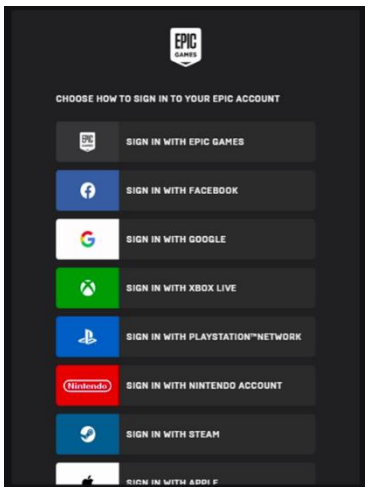
- An installed version of Unreal Engine (at least 4.26 or later). For instructions on how to install Unreal Engine, visit the Unreal Engine download page and refer to the Installing Unreal Engine guide for detailed instructions: Installing Unreal Engine | Unreal Engine Documentation
- A Cesium ion account to stream terrain and building assets into Unreal Engine. Sign up for a free Cesium ion account if you don't already have one: Sign Up
- A recent version of Visual Studio, which is a licensed software. Information about downloading this for Unreal Engine here: Setting Up Visual Studio for Unreal Engine

Set Up Unreal Engine and Cesium

Step 1: Install the Cesium for Unreal Plugin

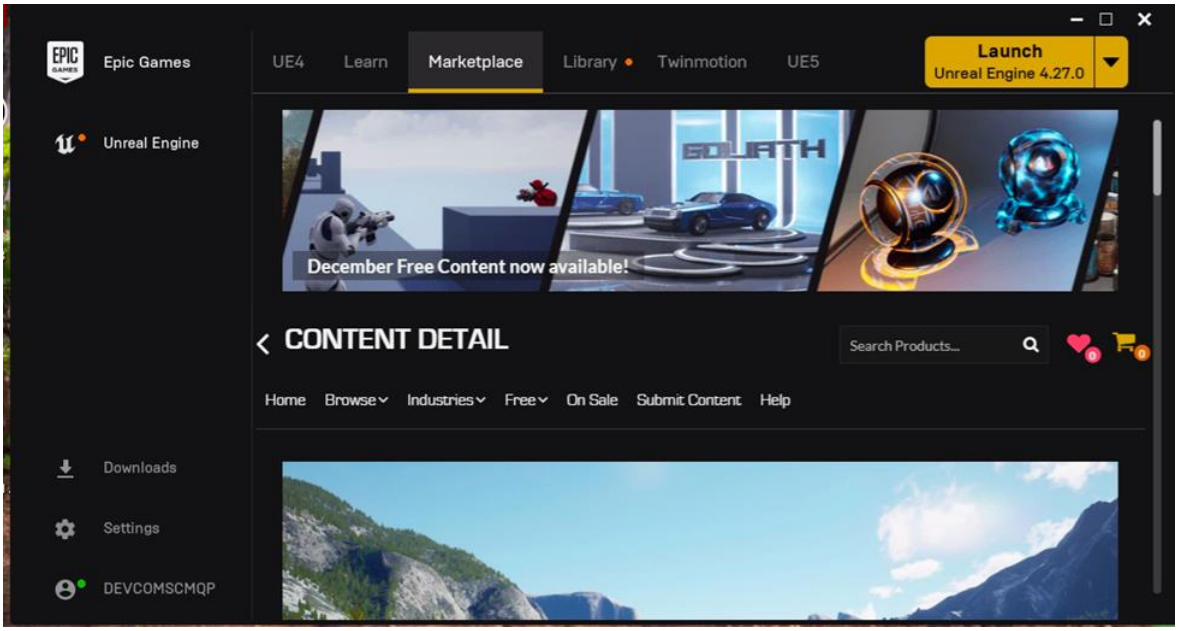
Open the Cesium for Unreal plugin page on the Unreal Engine Marketplace: Cesium for Unreal in Code Plugins - UE Marketplace. Sign in if needed and click on the **Free** button to install the plugin on your Unreal Engine account. If authorized, use the DEVCOMSCMQP@gmail.com account to sign in with Google.



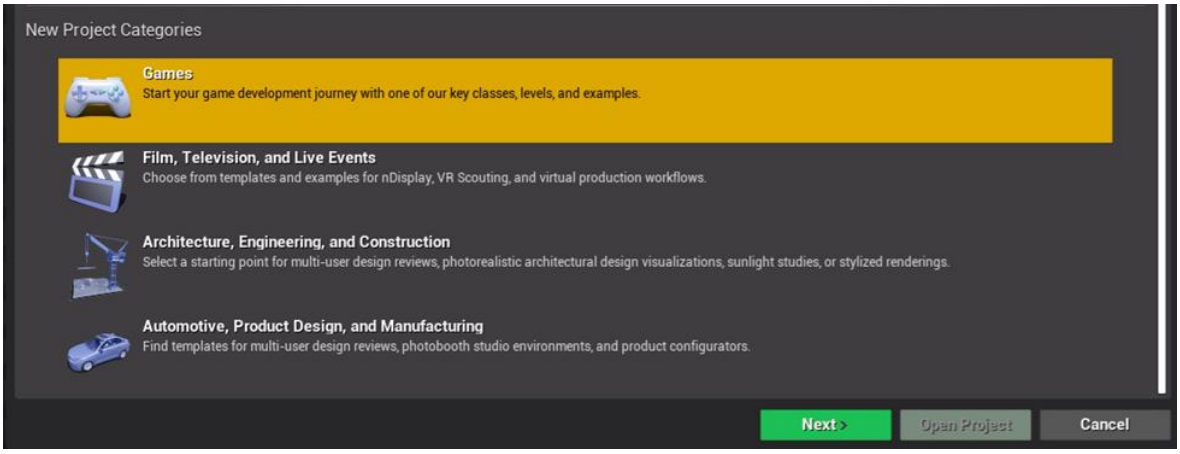


Step 2: Create the Project and Level

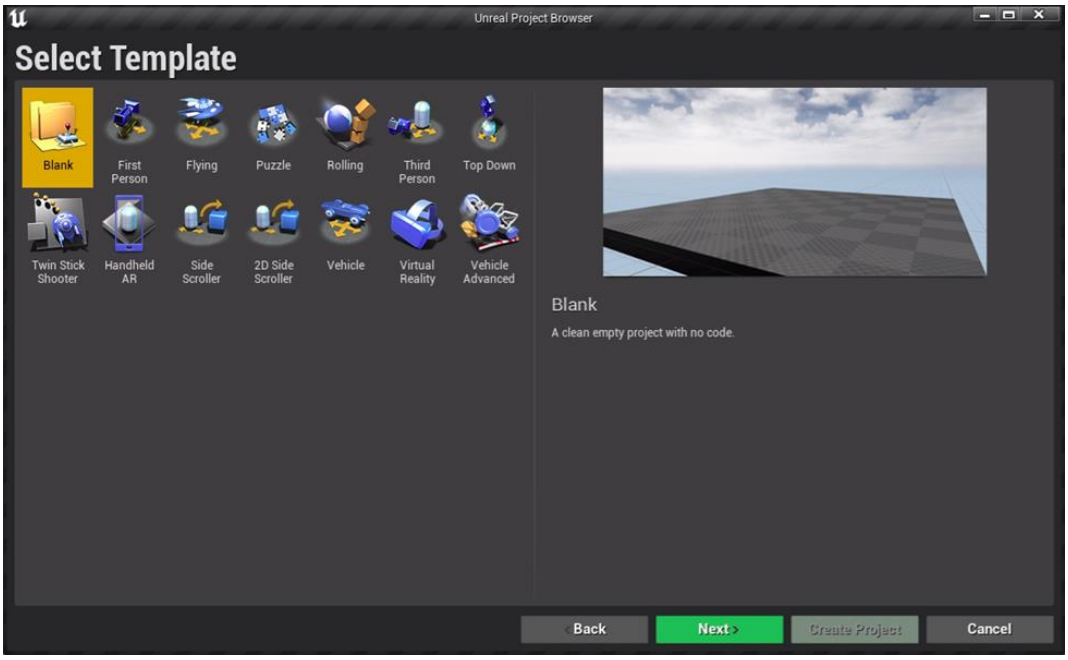
Click the **Launch** button in Unreal Engine to create a new project.



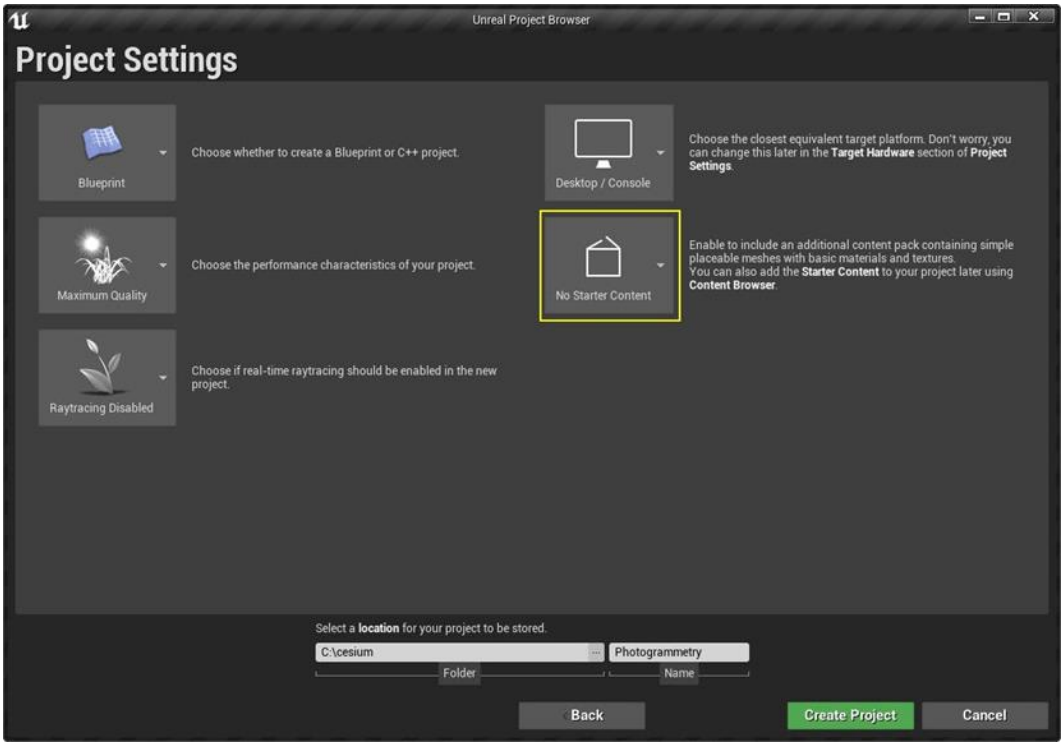
Select **Game** as the New Project Category and click the green **Next** button.



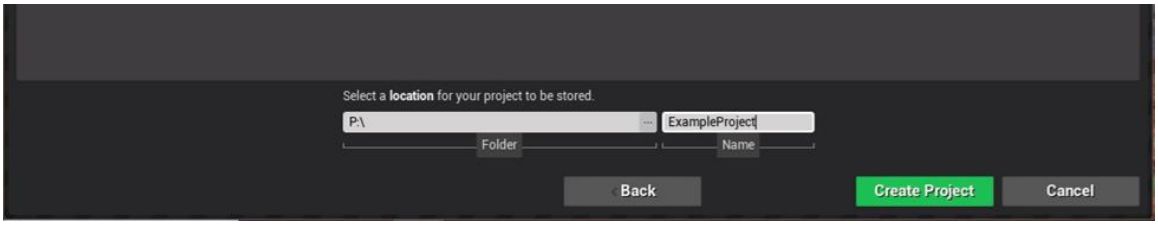
Now choose **Blank** as the Template. Again, click the green **Next** button.



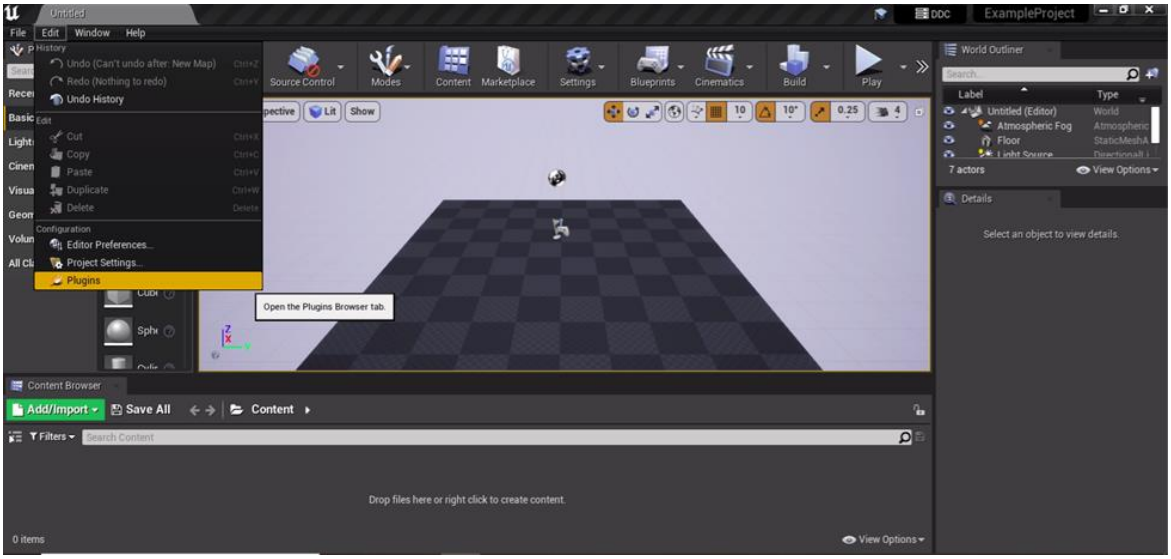
Choose **No Starter Content** to avoid cluttering the level.



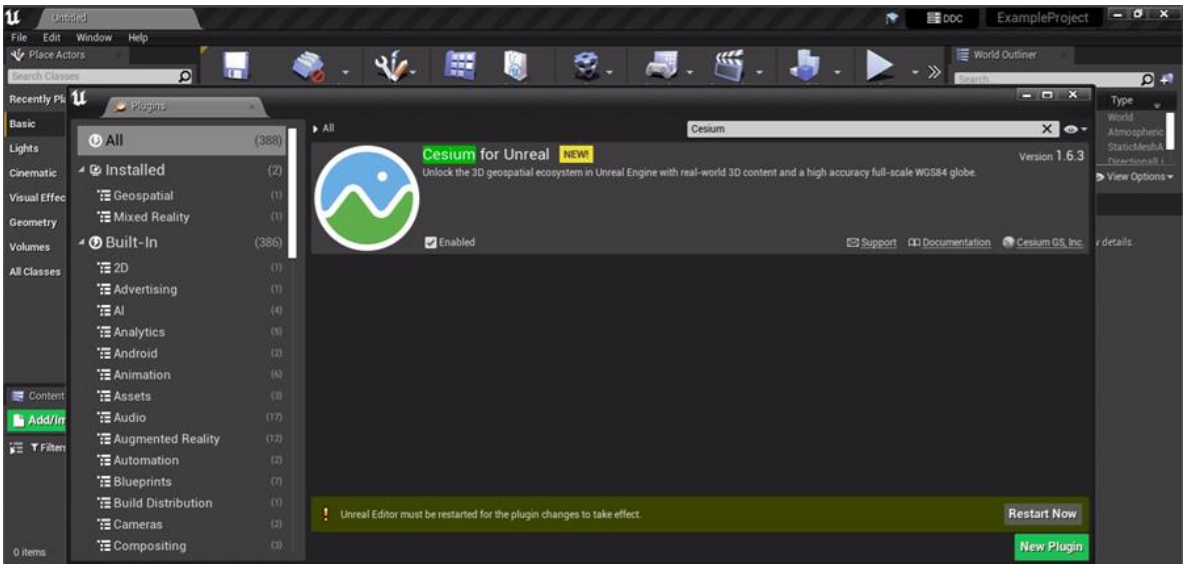
Name your project in the field at the bottom of the window. Here, you will also select the location where your file will save. Note, these files are about 4 GB, so select a location that will accommodate that size. Additionally, if using the remote desktop, ensure you are saving to a permanent location, like the P drive (pictured below). Then click the green **Create Project** button.



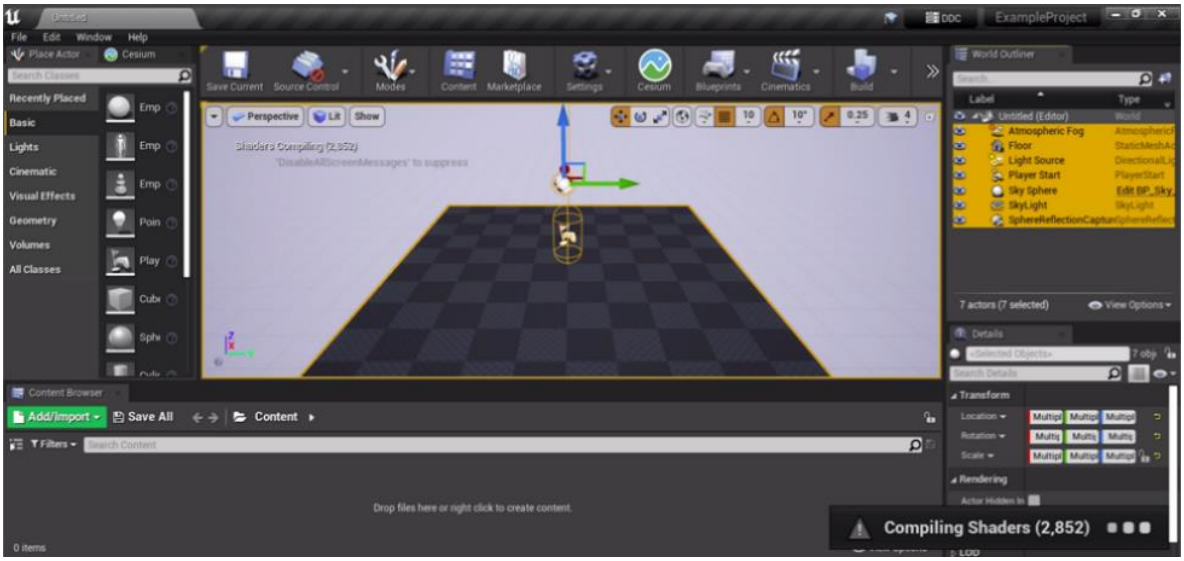
Next you will activate the Cesium for Unreal plugin. Go to Edit → Plugins in the top left of the editor window.



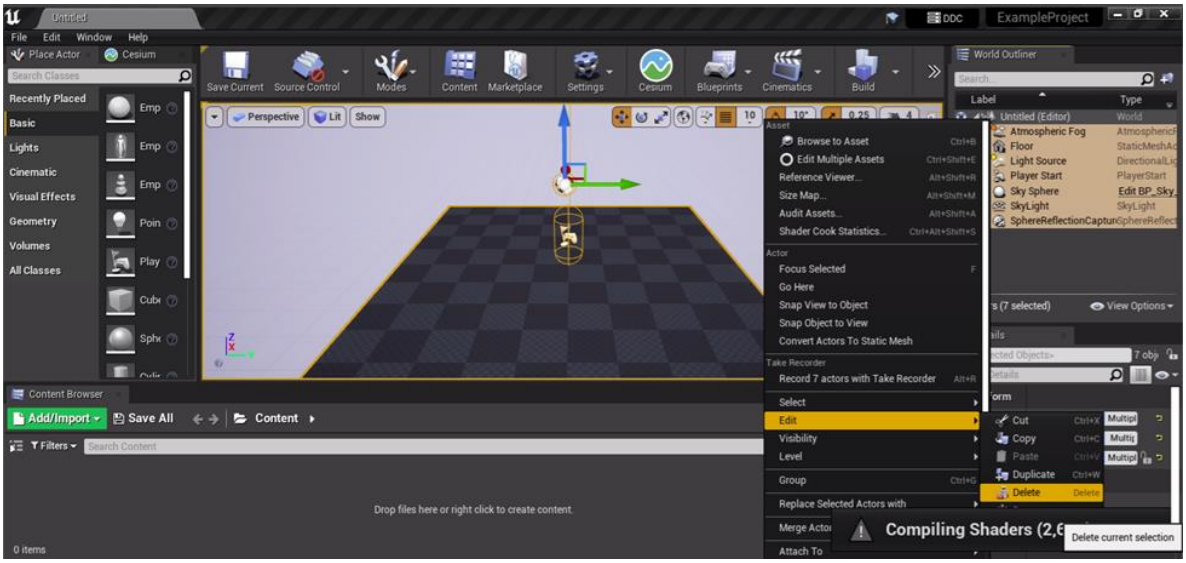
In the new Plugins window, search for “Cesium” in the search bar at the top-right corner. Click the **Enabled** checkbox for the plugin, so it is checked. A yellow banner will likely appear at the bottom, asking you to restart the engine. Click the **Restart Now** button. Your project will automatically reopen, and you can now exit out of the plug-ins window



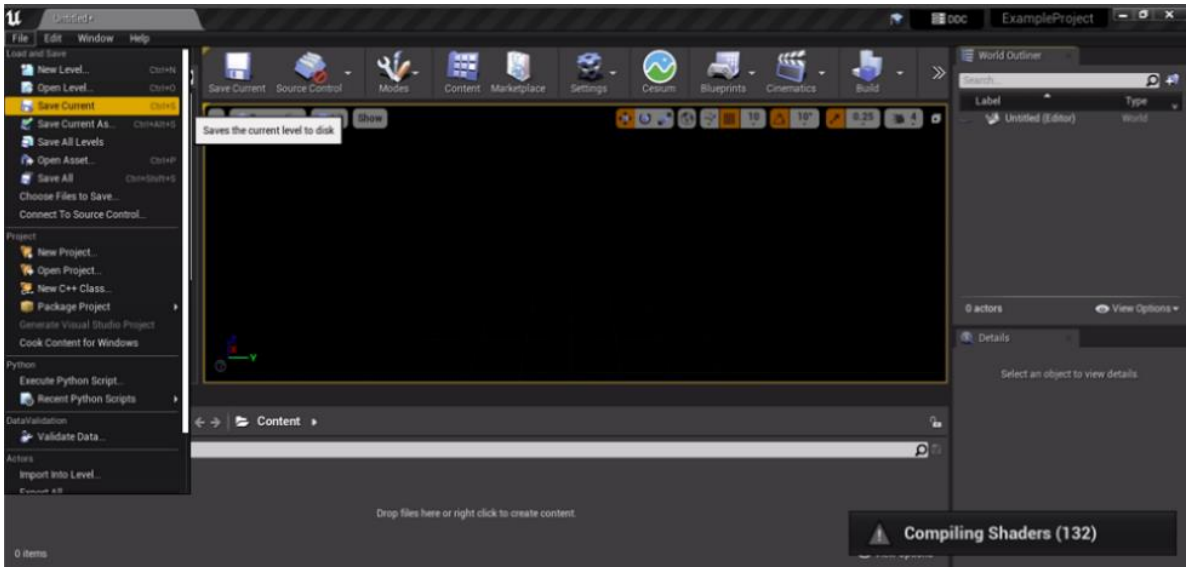
The World Outliner on the right side of the editor lists all the objects you have in your scene. Select all of them except the **Untitled (Editor)** by using the shift-click command.



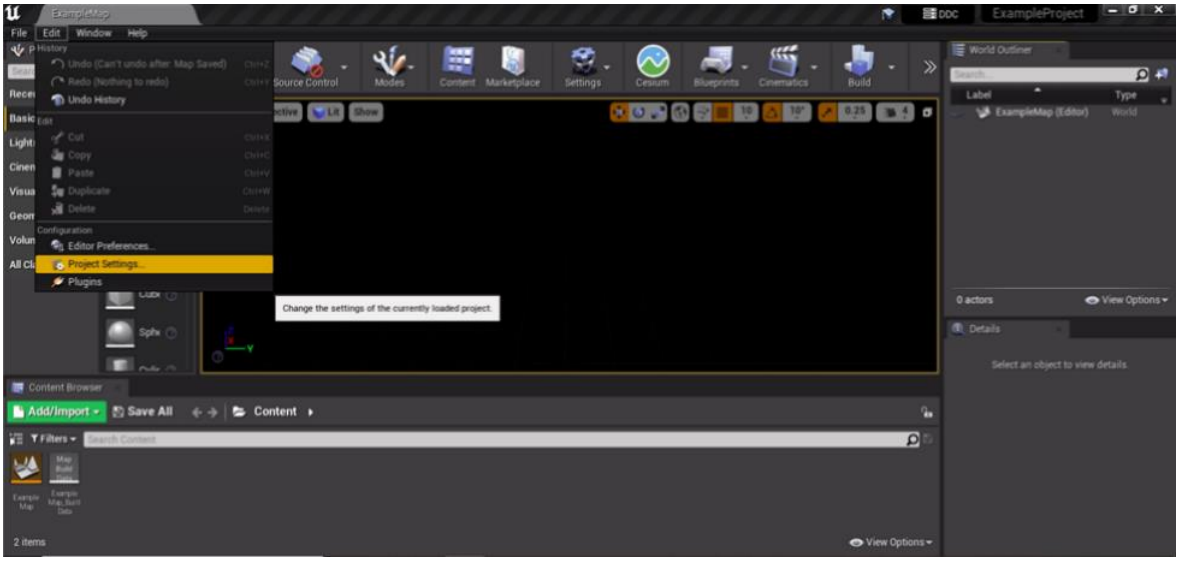
Right click on your selected objects → Edit → Delete, to remove all the objects.



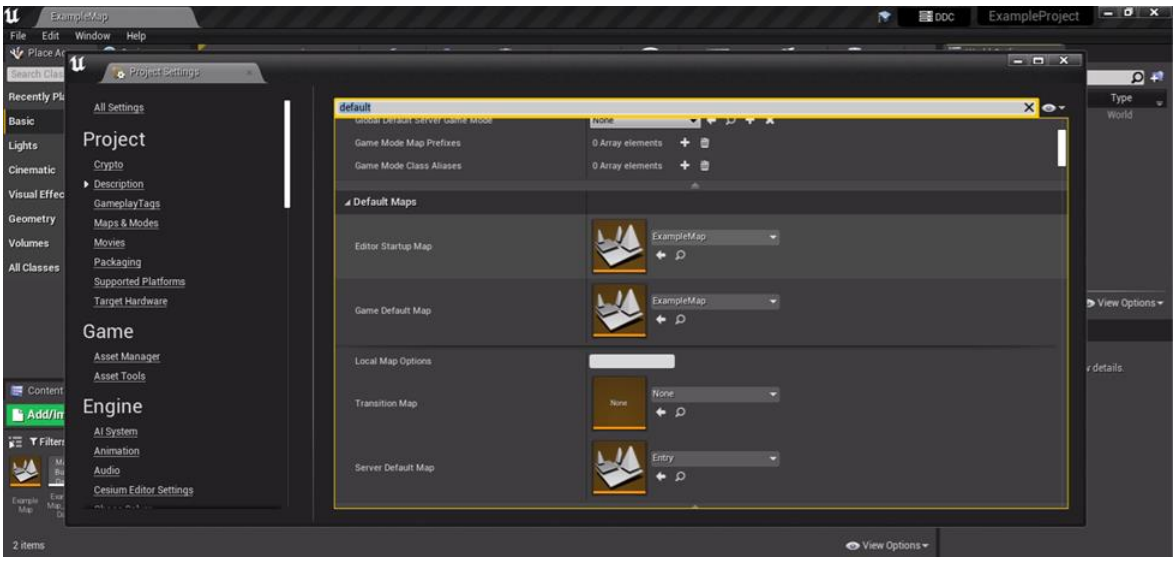
A message will pop-up warning about dependencies. Click **Yes All** to clear the message and delete the objects. You should now have a completely black viewport in the editor. Now you want to save your level by first clicking File → Save Current in the top left of the editor window.



Name your level in the bottom bar of the popup window, here we used the name “ExampleMap.” Theme click the grey **Save** button. This will save a file inside of your project. Go to Edit → Project Settings in the top left of the editor.

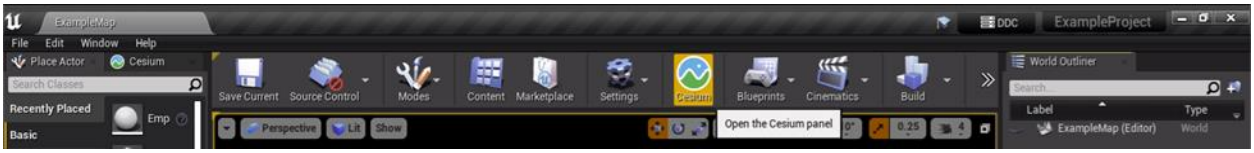


In the new popup window, search for “default.” Set the level you saved above as the Editor Startup Map and the Game Default Map. This ensures your level will be re-opened automatically when Unreal Editor is restarted. Once set, close the Project Settings window, it will automatically save your changes.

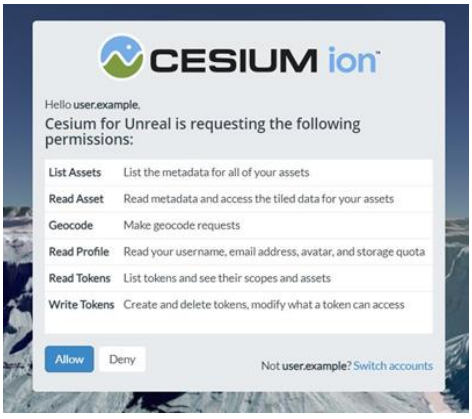


Step 3: Connect to Cesium

Open the Cesium panel by clicking the icon in the toolbar.



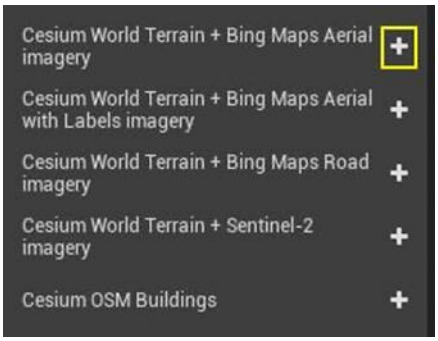
The Cesium panel will show up on the left side of the editor window. Connect to Cesium ion with the Connect button. A pop-up browser will show, asking you to allow Cesium for Unreal to access your assets with the currently logged-in account from Cesium ion. Select **Allow**. You may be asked to login again. If so, select the first option, **Sign in with Epic Games.**



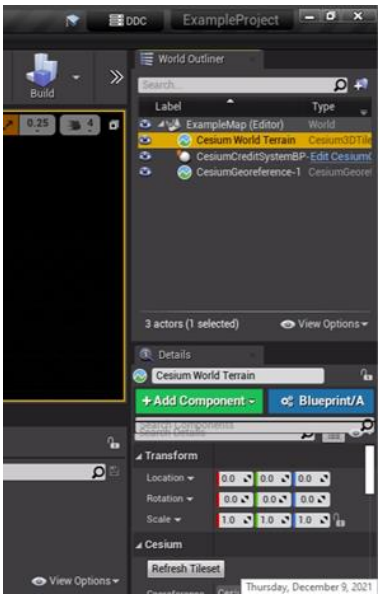
Now head back to Unreal Engine to continue with the next steps.

Step 4: Create a Globe

In this section, you will begin to populate the scene with assets from Cesium ion. Now that you've connected to Cesium, the window will look different, displaying many options for editing. From the Quick Add Cesium Ion Assets section at the bottom of the window, click the plus button next to **Cesium World Terrain + Bing Maps Aerial imagery.**

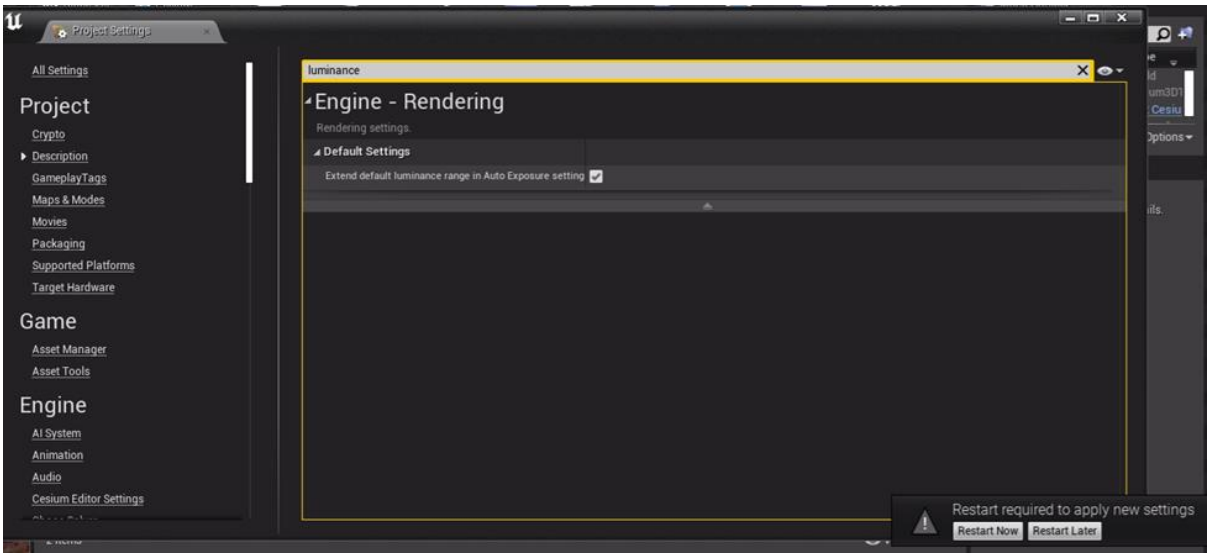


This step will generate new Cesium World Terrain and CesiumGeoreference actors in the World Outliner, on the left. You should also see a white sphere with arrows in your editing viewport.



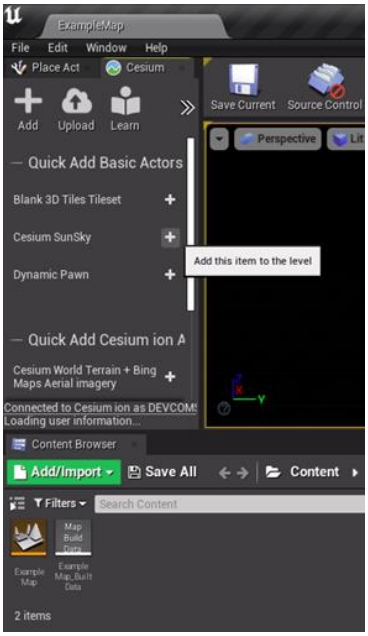
Step 5: Add Lighting with CesiumSunSky

The CesiumSunSky Actor controls the sun in a project. To being, go to Edit → Project Settings. In the pop-up, search for “luminance.” Make sure the option “Extend default luminance range in Auto Exposure settings” is enabled.

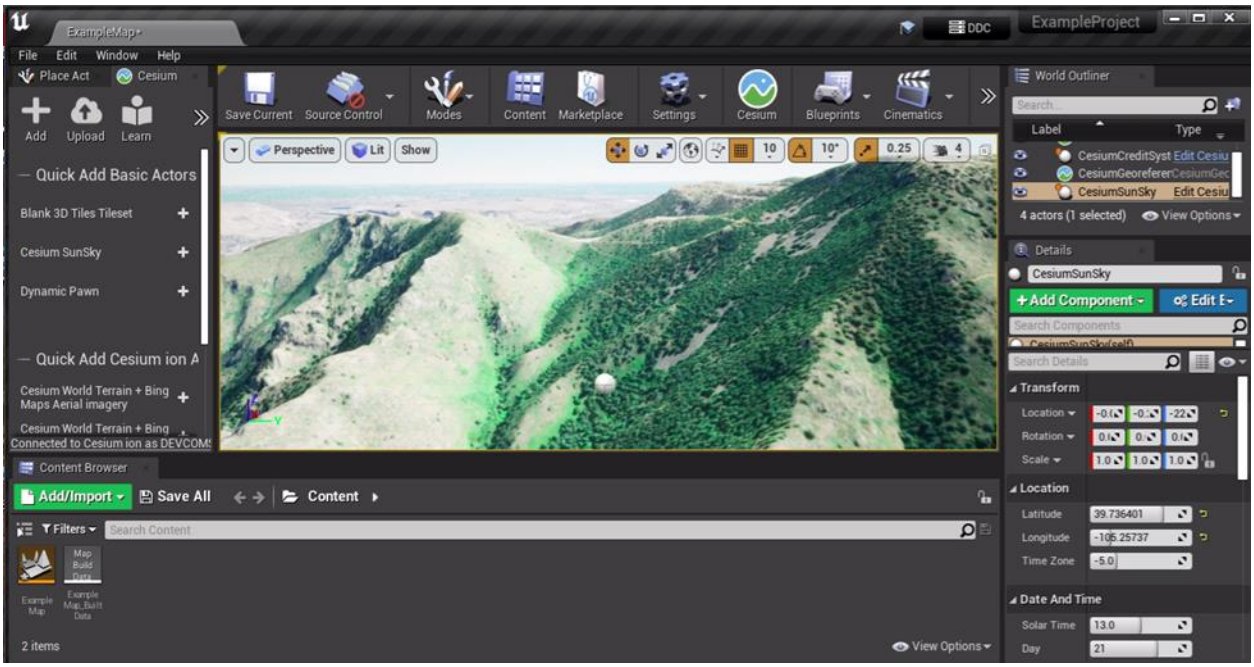


Restart Unreal Engine after enabling this option, as indicated by the popup in the corner.

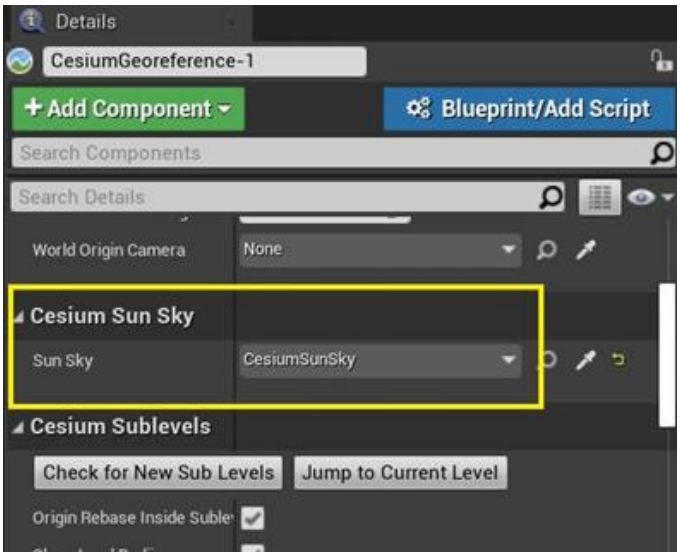
Return to the Cesium panel on the left. Add a CesiumSunSky actor to the scene by clicking the plus button.



The scene may appear white for a moment as the auto exposure adjusts. When it has settled, you should see a sky and atmosphere lighting the terrain.



Note: There may be a black strip near the horizon. To fix this, select the CesiumGeoreference actor, look for SunSky in the Details panel, and ensure it is set to your new CesiumSunSky actor.



Creating a Flight Path

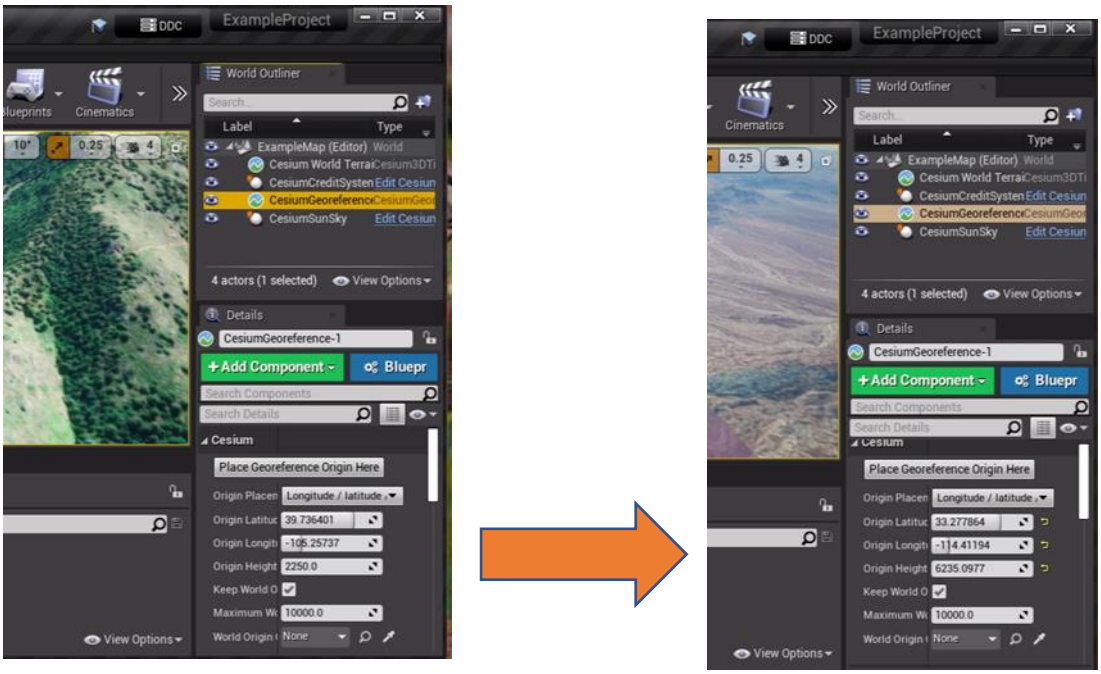
(Based on tutorial available here: https://cesium.com/learn/unreal/unreal-flight-tracker/)

In this section we will import a flight path and create a spline for an object and camera to fly along. For this tutorial, we will recreate a parachute test drop in Arizona.

Step 1: Adjust the Unreal Level

To position your view to the parachute test site. In the World Outliner on the left, select the CesiumGeoreference object. And you want to change your location in this tab to be:

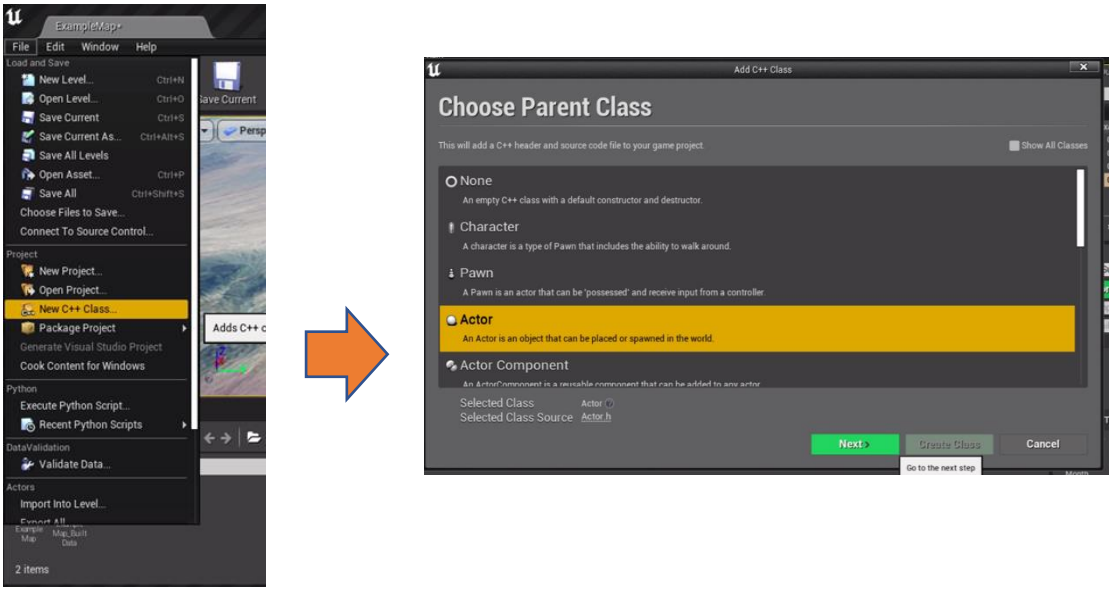
Origin Latitude = 33.277864 Origin Longitude = -114.41194 Origin Height = 6235.0977



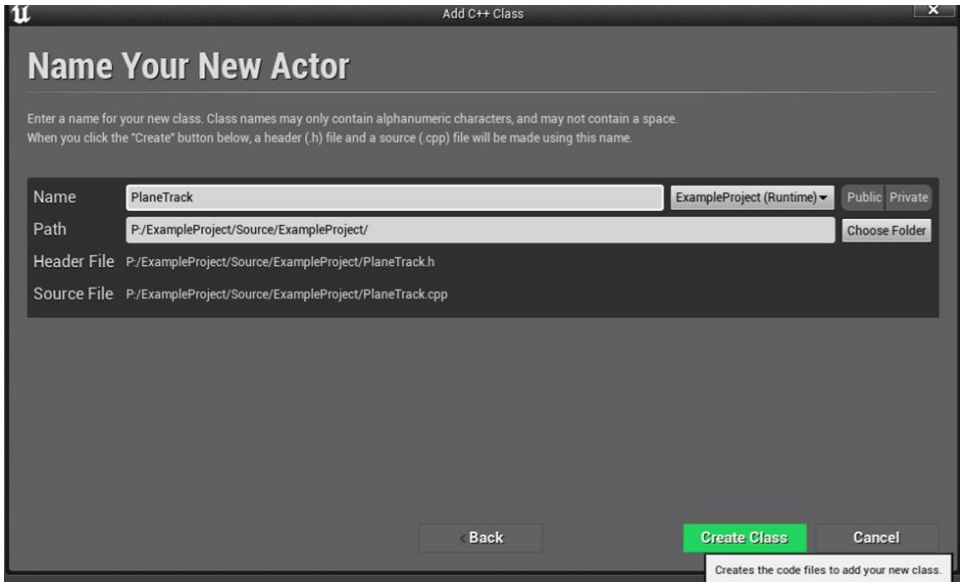
Step 2: Add the PlaneTrack Class

The PlaneTrack class will contain logic to process the drop data and generate position points for the spline which represents the parachute's path.

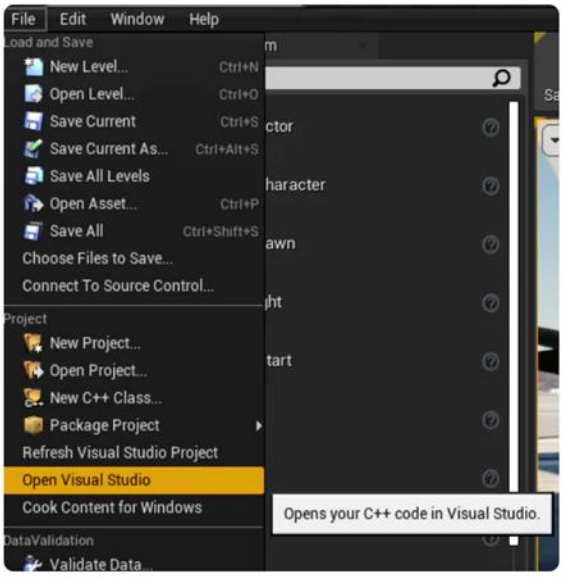
Add a new C++ class by going to File → New C++ Class… at the top left corner of the Unreal Editor. Select Actor as the parent class. And click the green **Next** button.



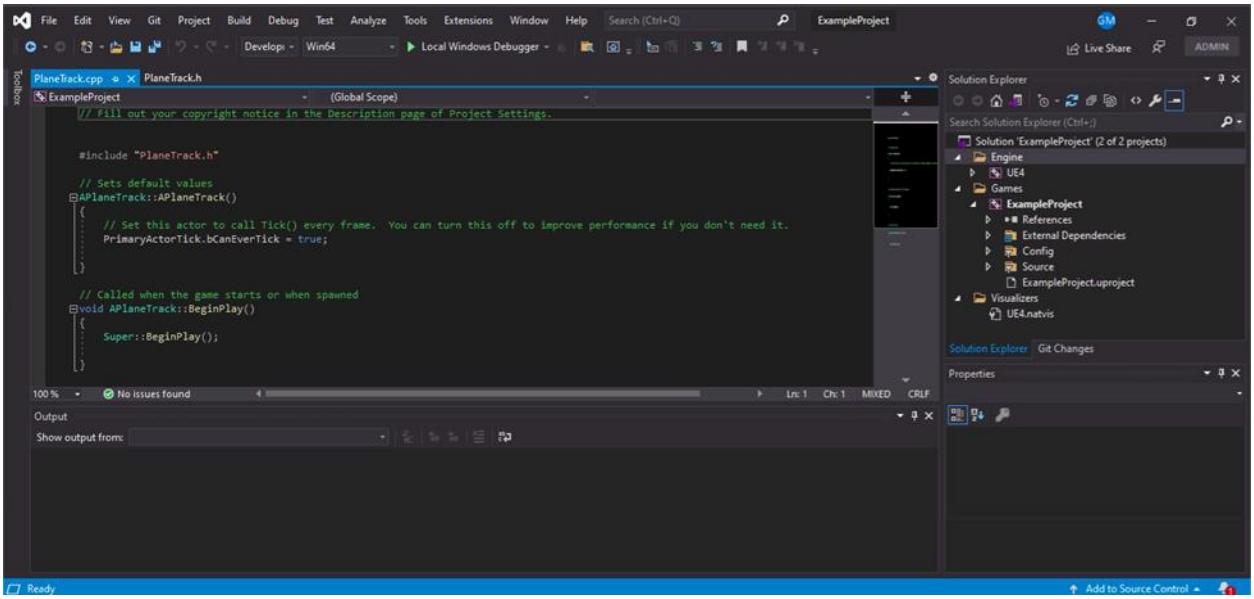
On the following page, set the name of the new class to "PlaneTrack". Then click the green **Create Class** button. Visual Studio should automatically open the file.



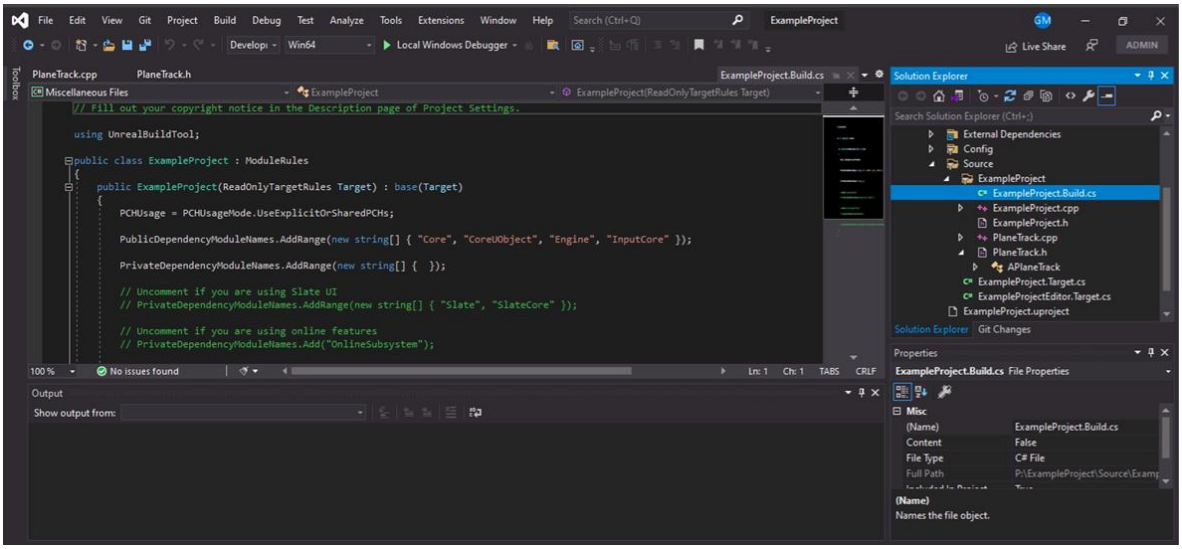
If Visual Studio does not open, navigate to File → Open Visual Studio. You will need to log in using your licensed account.



Visual Studio will open with associated Unreal Engine code.



On the right Panel, click Games → Source → YourProjectName there will be a file with the file type '.Build.cs' , named according to your project. Click that file to open the code.

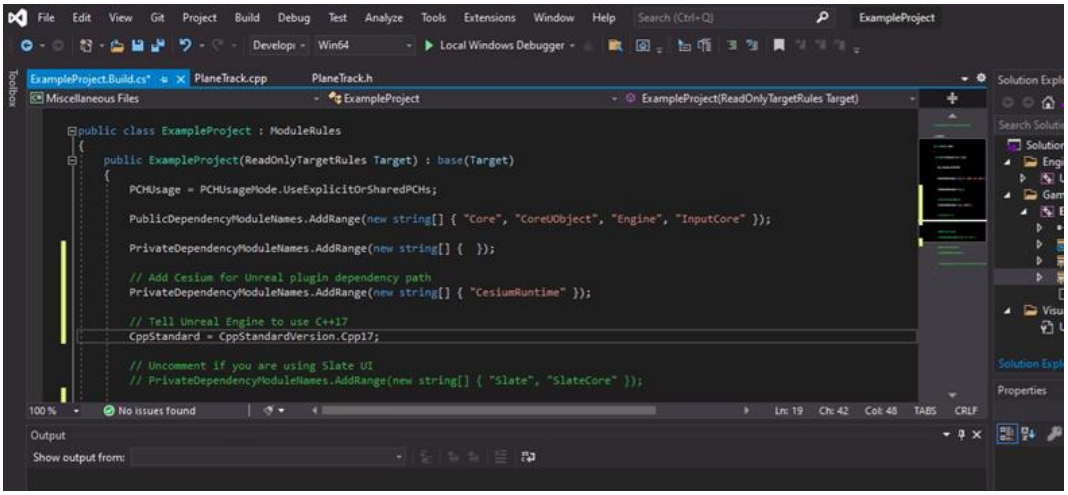


Add the following code snippet to the project, below the line:

"PrivateDepenencyModuleNames.AddRange(new string[] { });":

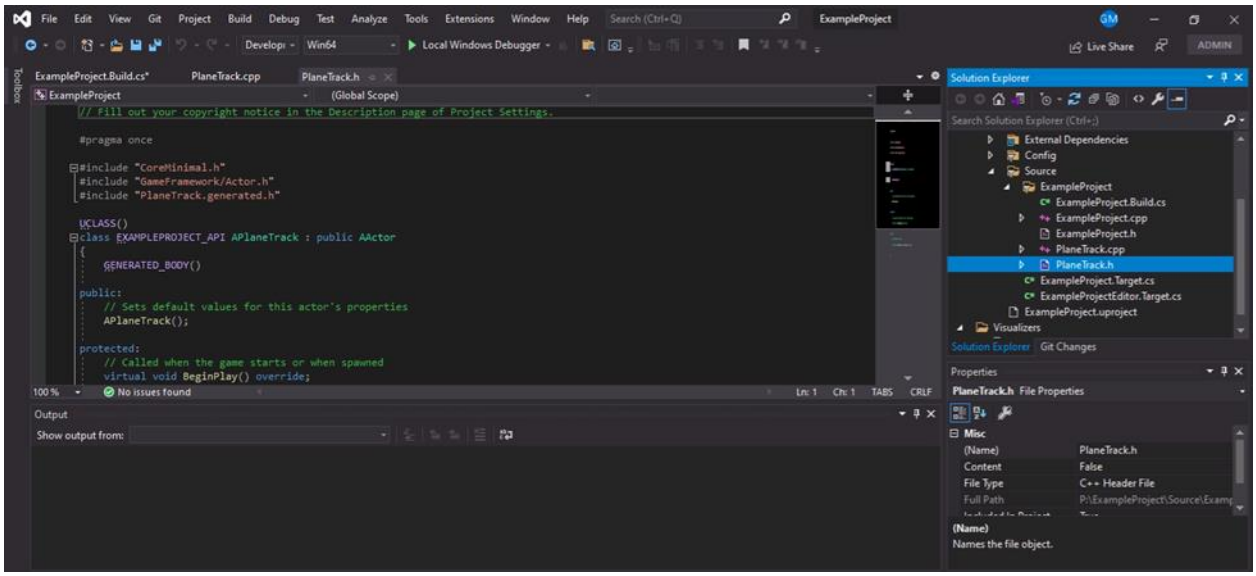
```
// Add Cesium for Unreal plugin dependency path

PrivateDependencyModuleNames.AddRange(new string[] { "CesiumRuntime" });

// Tell Unreal Engine to use C++17

CppStandard = CppStandardVersion.Cpp17;
```

After you paste in the code, it should like this:



Next you need to add some member variables to the PlaneTrack class to store the drop data, spline, and convert the data to the appropriate coordinate system.

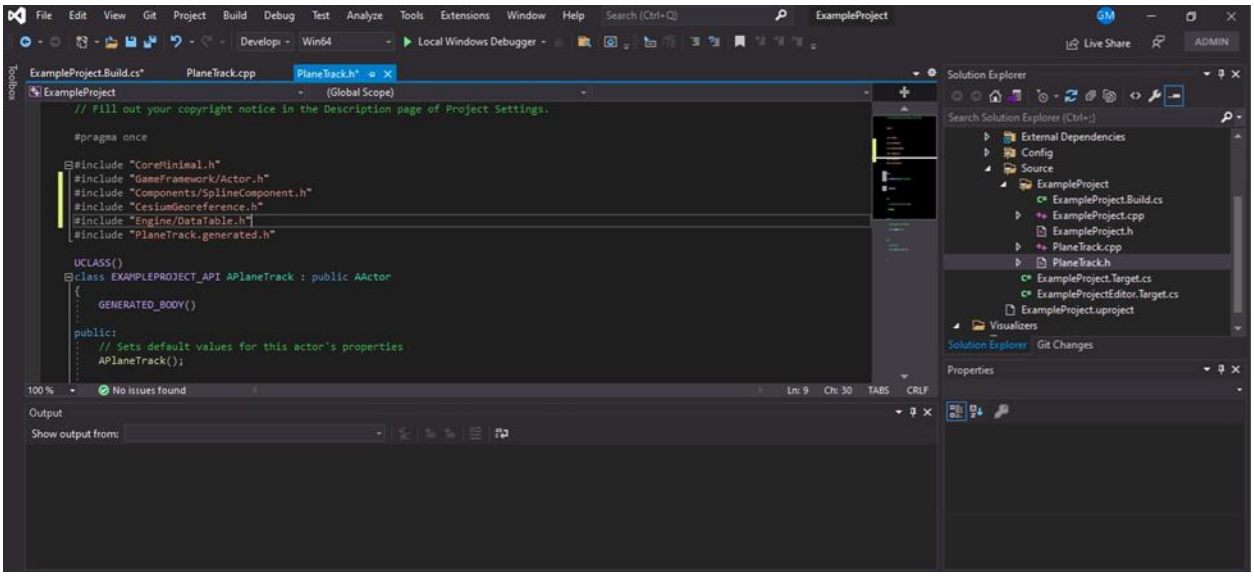
Now open a new .h file, PlaneTrack.h, located in the source folder.



At the top of the file, import the necessary libraries by copying and pasting in the code below:

```
// Add import paths. Make sure they go above the PlaneTrack.generated.h line

#include "Components/SplineComponent.h"

#include "CesiumGeoreference.h"

#include "Engine/DataTable.h"
```

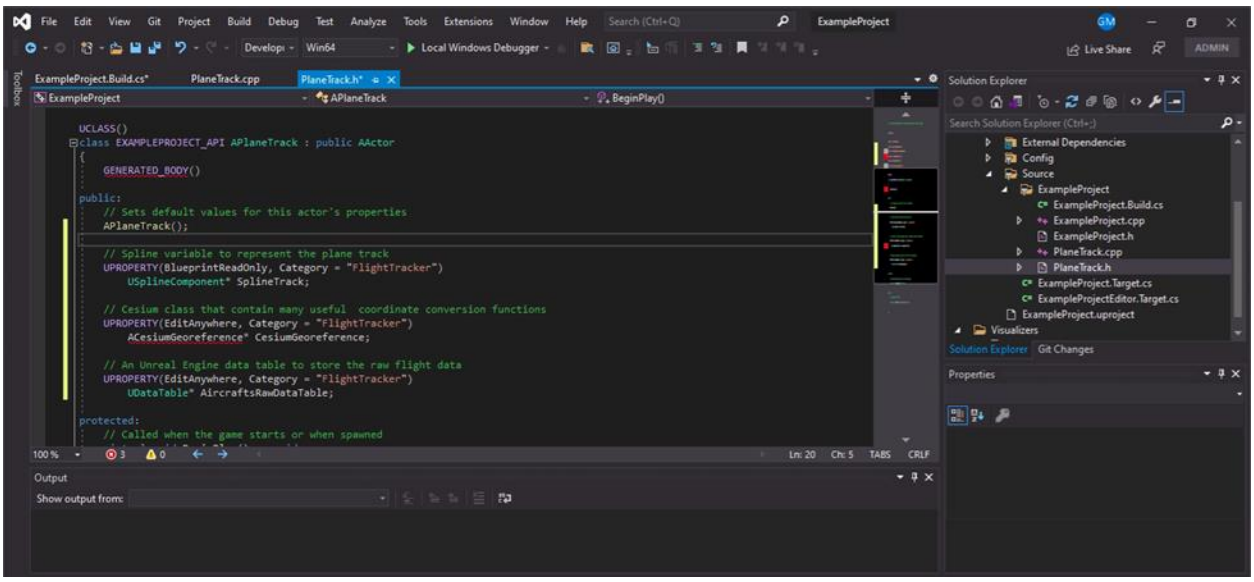
Once finished it should look like this:



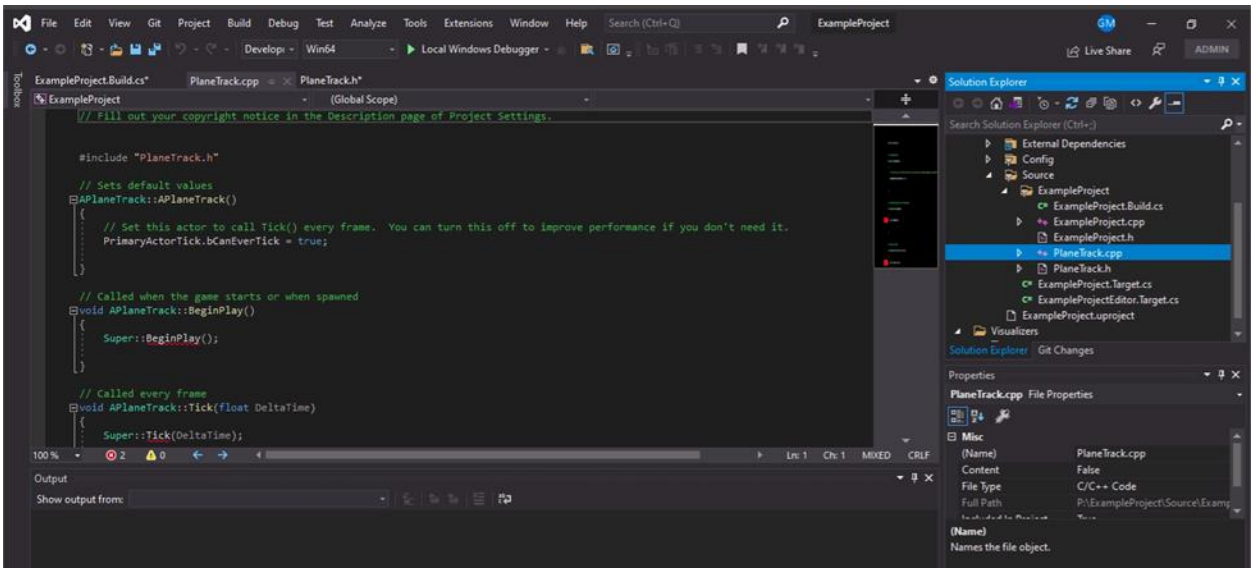
You will also need to add the below code snippet within the class under APlaneTrack();

```
public:

 // Spline variable to represent the plane track

 UPROPERTY(BlueprintReadOnly, Category = "FlightTracker")

   USplineComponent* SplineTrack;


 // Cesium class that contain many useful  coordinate conversion functions

 UPROPERTY(EditAnywhere, Category = "FlightTracker")

   ACesiumGeoreference* CesiumGeoreference;


 // An Unreal Engine data table to store the raw flight data

 UPROPERTY(EditAnywhere, Category = "FlightTracker")

   UDataTable* AircraftsRawDataTable;
```

Once completed your code should look like this:



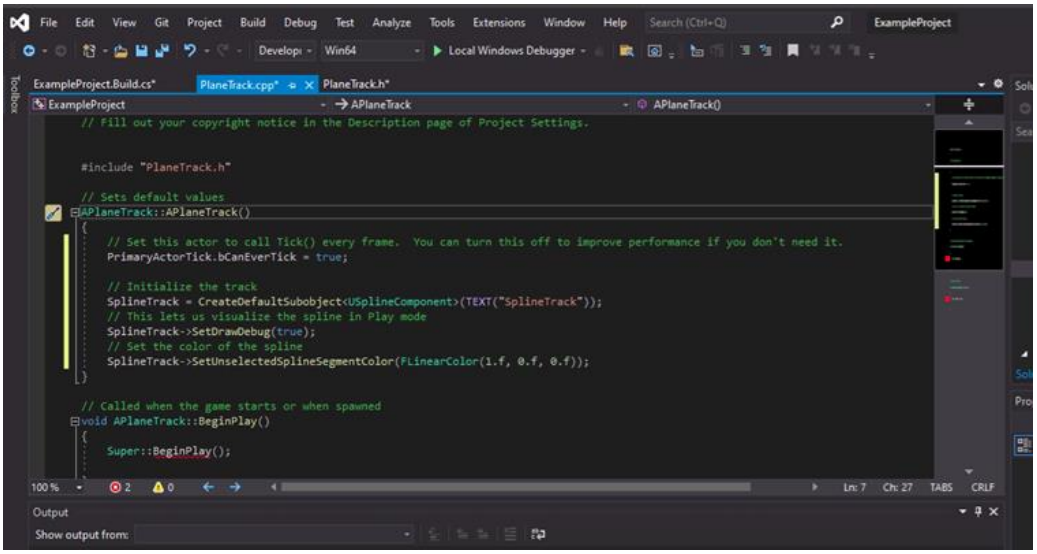
Navigate to and open the file PlaneTrack.cpp. It should initially look like the image below. Navigate to the APlaneTrack::APlaneTrack() line in the code. This is known as a constructor, and we will modify it in the next step.



Change the contents of the APlaneTrack constructor by replacing the existing code with the code below:

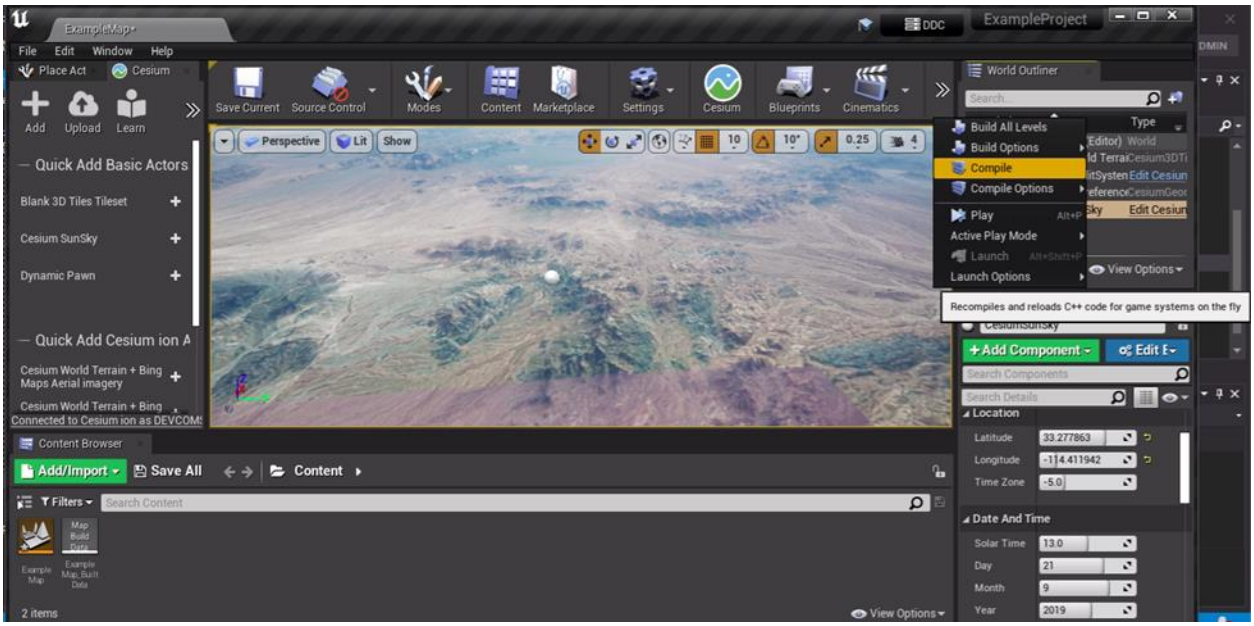
```
APlaneTrack::APlaneTrack()

{

  // Set this actor to call Tick() every frame.  You can turn this off to //improve performance if you don't need it.

  PrimaryActorTick.bCanEverTick = true;



  // Initialize the track

  SplineTrack = CreateDefaultSubobject<USplineComponent>(TEXT("SplineTrack"));

  // This lets us visualize the spline in Play mode

  SplineTrack->SetDrawDebug(true);

  // Set the color of the spline

  SplineTrack->SetUnselectedSplineSegmentColor(FLinearColor(1.f, 0.f, 0.f));

}
```

Once you've made this change your file should look like this:



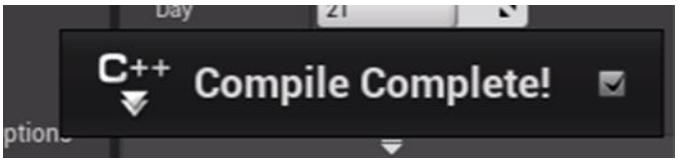
Now save each of the C++ files you edited (there should be three) in the Visual Studios Editor. Do this by clicking the blue save button in the top left and saving to the default location.

After you save your files, return to the Unreal Engine Editor, and click the **Compile** button at the top tool panel.



Depending on screen size, you may see the icon in the toolbar here:

If the code is well-formatted and correct, you will see the “Compile Complete” message at the bottom right corner of the Unreal Engine main editor. In our instructions, compiling the code refers to this step.

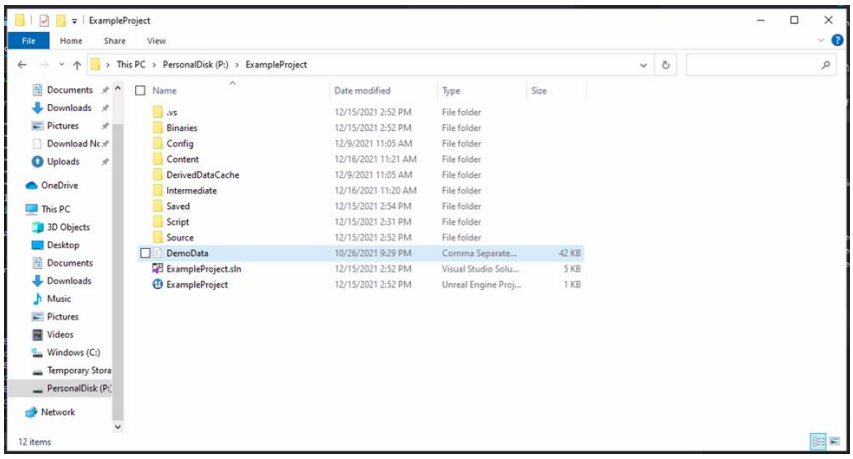


Step 3: Bring in Real-World Flight Data

Next, you are going to use parachute test-drop data for this demo. The height in Cesium for Unreal is in meters relative to the WGS84 ellipsoid. The data is provided in the required format, and available on the shared drive and here: DemoData.csv. Then download the data and drop a copy of the CSV file into your project folder.

	A	B	C	D
1	id	latitude	longitude	height
2	1	33.27786	-114.412	6235.098
3	2	33.278	-114.412	6202.197
4	3	33.27805	-114.412	6191.097
5	4	33.27811	-114.412	6180.897
6	5	33.27817	-114.412	6169.897
7	6	33.27823	-114.412	6160.198
8	7	33.27829	-114.412	6152.398
9	8	33.27837	-114.412	6145.198
10	9	33.27848	-114.412	6138.398
11	10	33.27859	-114.412	6131.499
12	11	33.27871	-114.412	6124.899
13	12	33.27884	-114.412	6118.099
14	13	33.27896	-114.412	6111.799
15	14	33.2791	-114.412	6105.399
16	15	33.27922	-114.412	6098.999
17	16	33.27935	-114.412	6092.9
18	17	33.27947	-114.412	6086.7
19	18	33.27959	-114.412	6080.6
20	19	33.27971	-114.412	6074.3
21	20	33.27983	-114.412	6068.1
22	21	33.27995	-114.412	6061.901
23	22	33.28008	-114.412	6055.401
24	23	33.28022	-114.412	6049.101
25	24	33.28036	-114.412	6042.801
26	25	33.28051	-114.412	6036.501
27	26	33.28065	-114.412	6030.401
28	27	33.28079	-114.412	6024.101
29	28	33.28094	-114.412	6018.101

DemoData

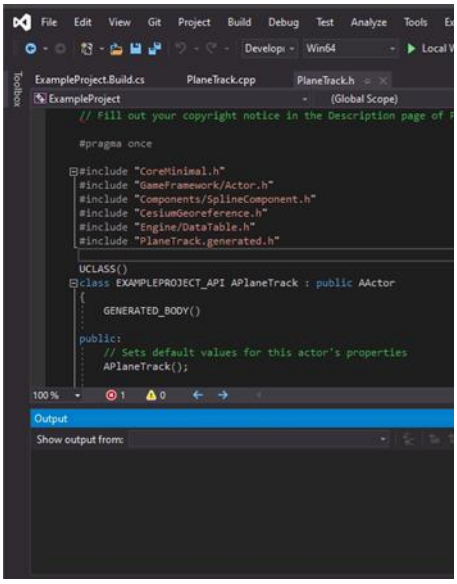


For the PlaneTrack class to access the data to perform coordinate conversions, you will use the Unreal Engine DataTable to store the data inside of the project. In this step, you will create a data structure to represent the structure of the drop data.

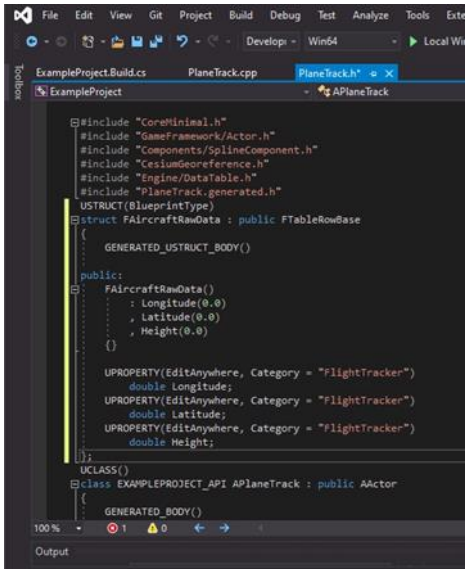
In the PlaneTrack.h file, insert this snippet of code directly below the imports to define the flight database structure.

```
USTRUCT(BlueprintType)

struct FAircraftRawData : public FTableRowBase

{

  GENERATED_USTRUCT_BODY()

  public:

    FAircraftRawData()

      : Latitude(0.0)

      , Longitude(0.0)

      , Height(0.0)

    {}

  UPROPERTY(EditAnywhere, Category = "FlightTracker")

    double Latitude;

  UPROPERTY(EditAnywhere, Category = "FlightTracker")

    double Longitude;

  UPROPERTY(EditAnywhere, Category = "FlightTracker")

    double Height;

};
```

Before you add the code, it should look like:

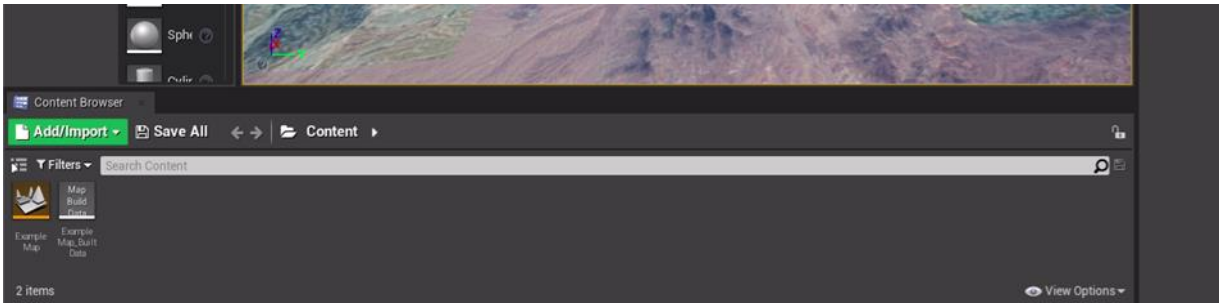


After you add the code it should look like:

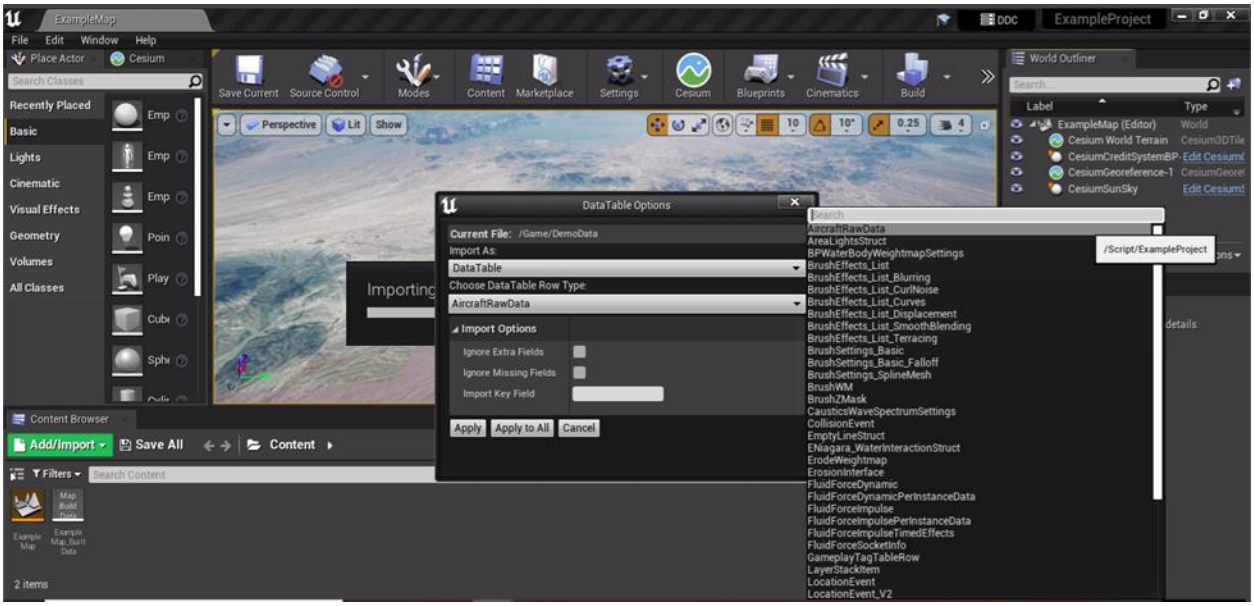


Next, save and compile the code. Remember, if the code is well-formatted and correct, you will see the “Compile Complete” message at the bottom right corner of the Unreal Engine main editor.

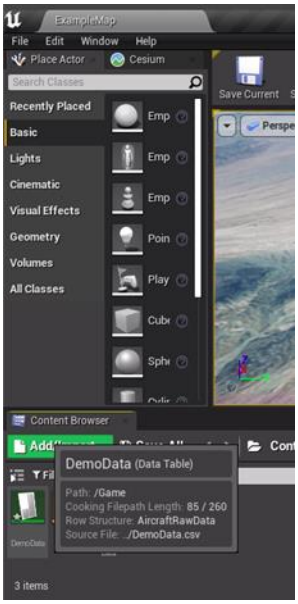
Next you will drag-and-drop the .csv data file into the Unreal Engine Content Browser. This is the panel at the bottom of your editor.



After you drop in the file, a pop-up window will appear. In the **Choose DataTable Row Type** dropdown select **AircraftRawData** (typically the first option):



Click **Apply**. Then to check the data uploaded correctly, double-click on the file in the Content Browser to open the data table:



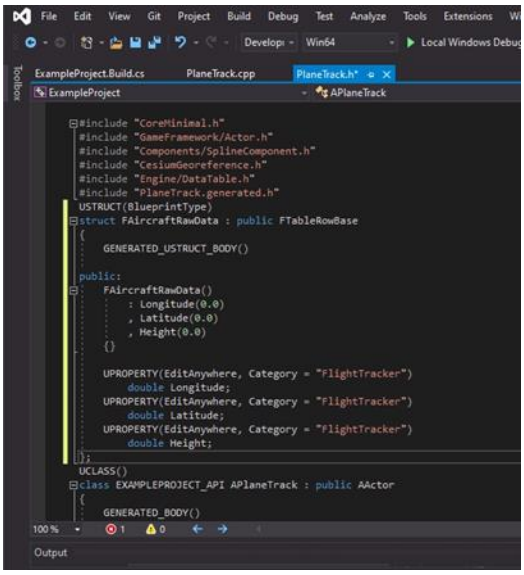
Step 4: Add Positions to the Flight Track

In this step, you will add some more code to the PlaneTrack class to complete the rest of the functionality needed to create the spline path.

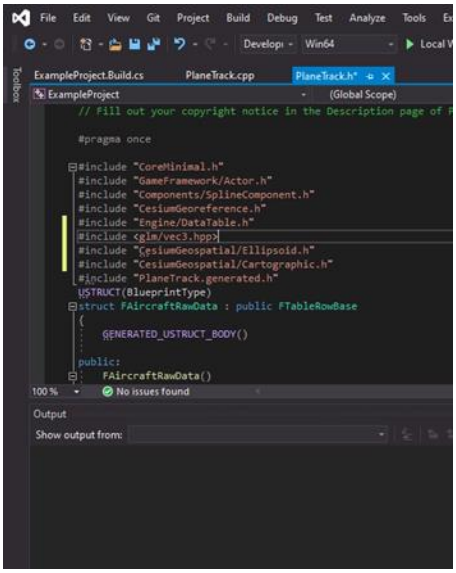
Add the following imports to the PlaneTrack.h file, above the PlaneTrack.Generated.h line.

```
// Imports should be placed above the PlaneTrack.Generated.h line.

#include <glm/vec3.hpp>

#include "CesiumGeospatial/Ellipsoid.h"

#include "CesiumGeospatial/Cartographic.h"
```

Before you add the code it should look like:



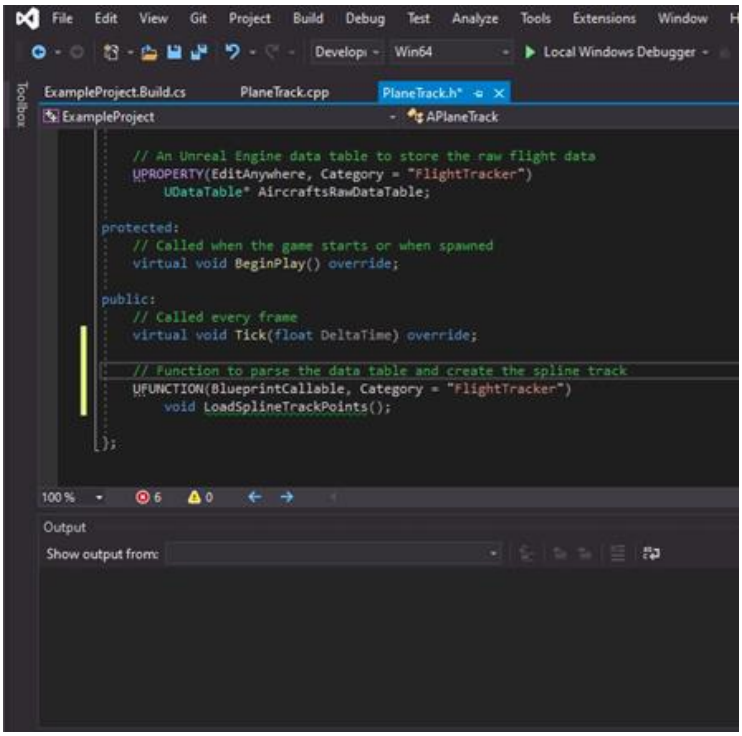
After it should look like:



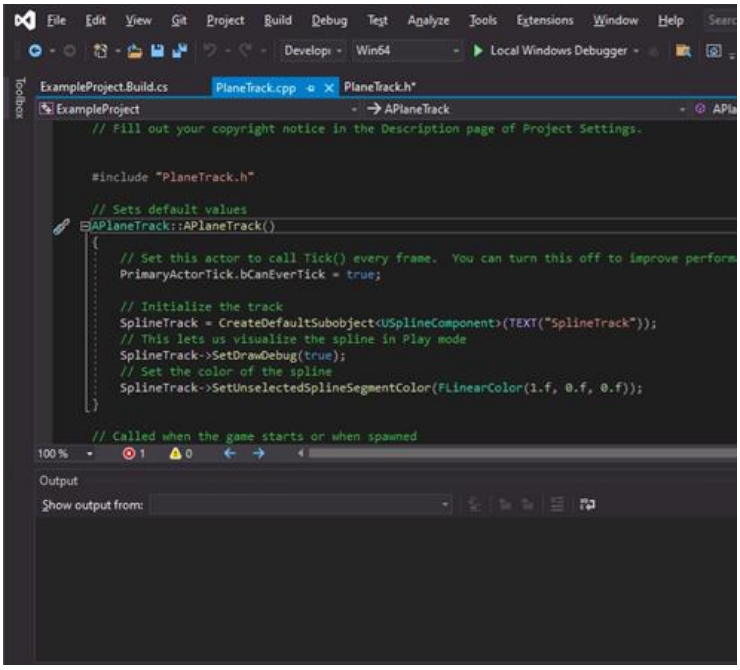
While still in PlaneTrack.h, add the following function at the end of the APlaneTrack class definition.

```
public:

 // Function to parse the data table and create the spline track

 UFUNCTION(BlueprintCallable, Category = "FlightTracker")

  void LoadSplineTrackPoints();
```

After, it should look like:



Switch to PlaneTrack.cpp file. Before you make any changes, it should look like:

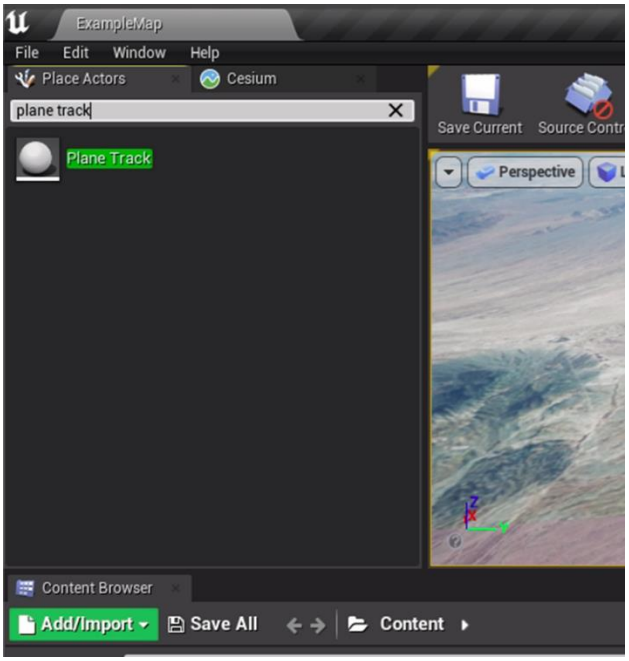


Switch to PlaneTrack.cpp. Add the following code snippet at the bottom of the file to create the body of LoadSplineTrackPoints:

```
void APlaneTrack::LoadSplineTrackPoints()
{
    if (this->AircraftsRawDataTable != nullptr && this->CesiumGeoreference != nullptr)
    {
        int32 PointIndex = 0;
        for (auto& row : this->AircraftsRawDataTable->GetRowMap())
        {
            FAircraftRawData* Point = (FAircraftRawData*)row.Value;
            // Get row data point in lat/long/alt and transform it into UE4 points
            double PointLatitude = Point->Latitude;
            double PointLongitude = Point->Longitude;
            double PointHeight = Point->Height;

            // Compute the position in UE coordinates
            glm::dvec3 UECoords = this->CesiumGeoreference-
>TransformLongitudeLatitudeHeightToUe(glm::dvec3(PointLongitude, PointLatitude,
PointHeight));
            FVector SplinePointPosition = FVector(UECoords.x, UECoords.y, UECoords.z);
            this->SplineTrack->AddSplinePointAtIndex(SplinePointPosition, PointIndex,
ESplineCoordinateSpace::World, false);

            // Get the up vector at the position to orient the aircraft
            const CesiumGeospatial::Ellipsoid& Ellipsoid =
CesiumGeospatial::Ellipsoid::WGS84;
            glm::dvec3 upVector =
Ellipsoid.geodeticSurfaceNormal(CesiumGeospatial::Cartographic(FMath::DegreesToRadia
ns(PointLongitude), FMath::DegreesToRadians(PointLatitude),
FMath::DegreesToRadians(PointHeight)));

            // Compute the up vector at each point to correctly orient the plane
            glm::dvec4 ecefUp(upVector, 0.0);
            const glm::dmat4& ecefToUnreal = this->CesiumGeoreference-
>GetEllipsoidCenteredToUnrealWorldTransform();
            glm::dvec4 unrealUp = ecefToUnreal * ecefUp;
            this->SplineTrack->SetUpVectorAtSplinePoint(PointIndex, FVector(unrealUp.x,
unrealUp.y, unrealUp.z), ESplineCoordinateSpace::World, false);

            PointIndex++;
        }
        this->SplineTrack->UpdateSpline();
    }
}
```

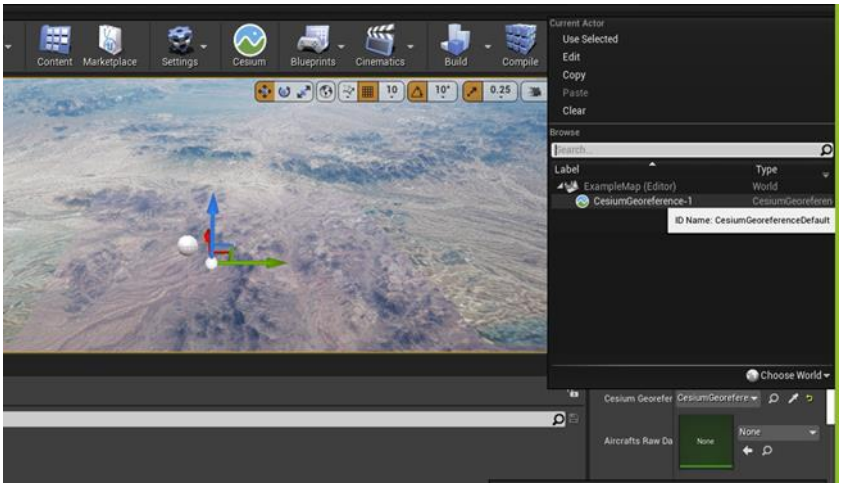
Save and compile the code. Make sure you get the “Compile Completed” notification.

Navigate back to the Unreal Engine to add the flight track to the scene. In the Place Actors panel, search for "Plane Track"

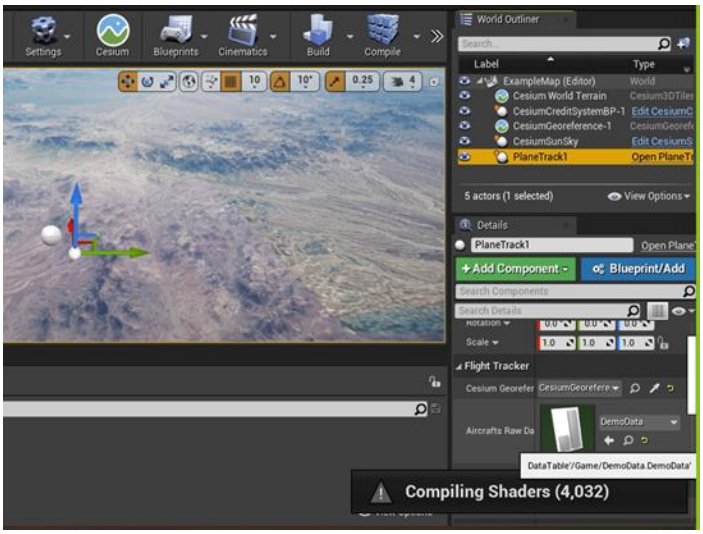


Click and drag the actor into the viewport, then drop it into the scene.

Select the PlaneTrack actor. In the Details panel, look for the Flight Tracker category. Set the Cesium Georeference variable to the Cesium Georeference variable in your scene



Then set the Aircrafts Raw Data Table variable to the data table added in step 3.

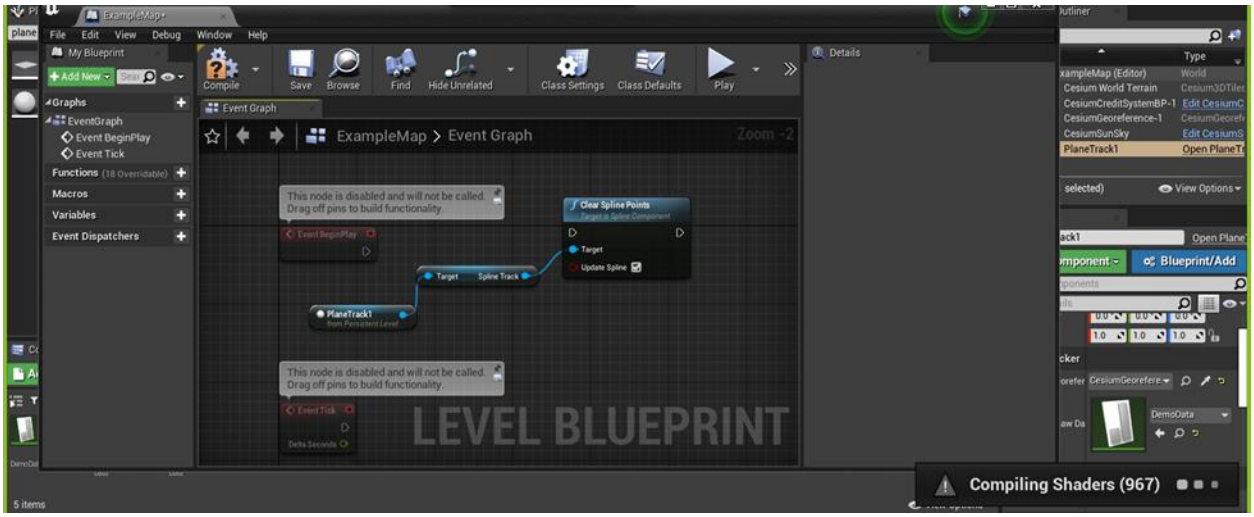


Navigate to and click on the Open Level Blueprint button in the toolbar at the top of the screen.

Near the Event BeginPlay node, Drag-and-drop the PlaneTrack actor from the World Outliner.

Right click on the PlaneTrack actor to bring up a menu of additional nodes, which are the code building blocks of Blueprint. Search for the Clear Spline Points function node and add it by clicking.

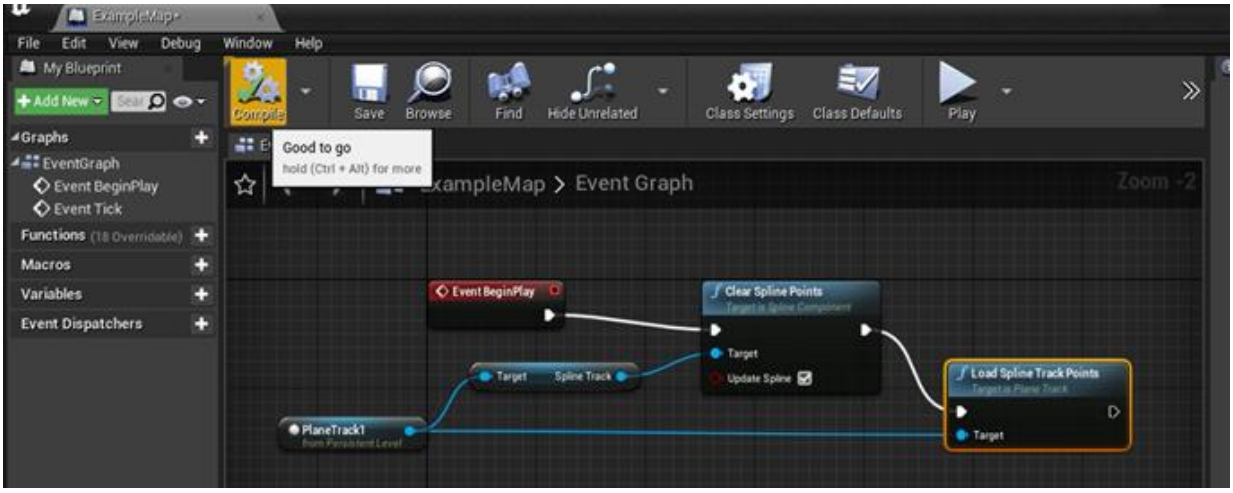
Connect the blocks by clicking and dragging the arrow from the PlaneTrack node to the Clear Spline Points node, and an additional node, Target, will automatically populate to help connect them.



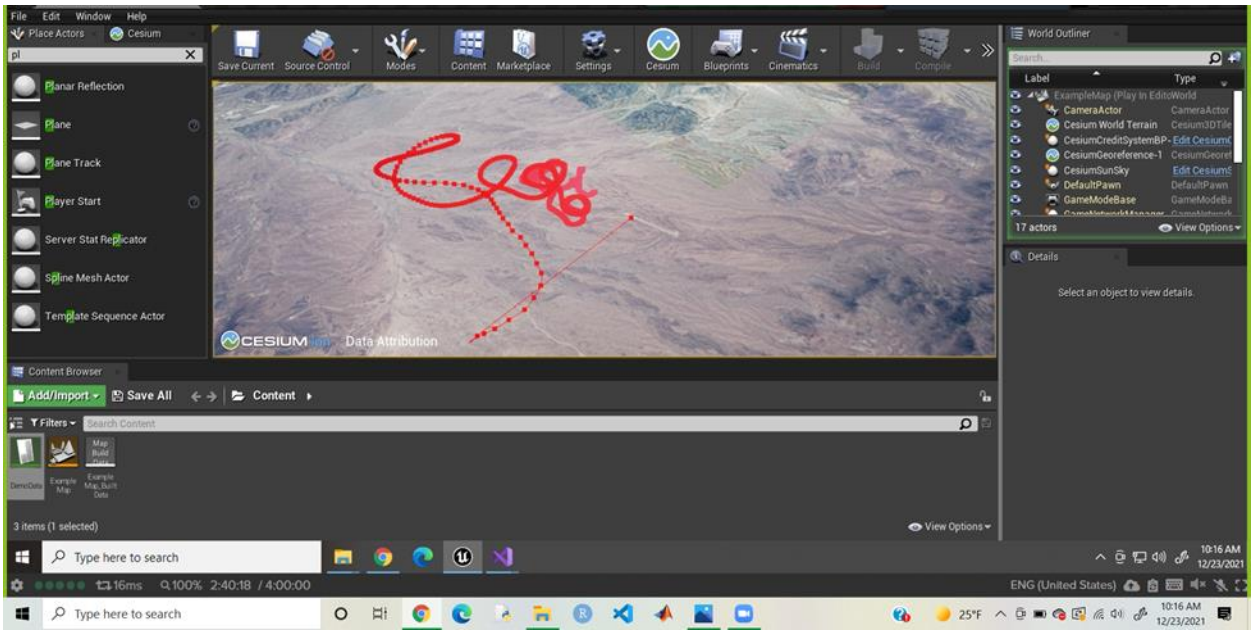
From the PlaneTrack object node, drag another connection and search for "Load Spline Track Points."

Connect the Clear Spline Points and the Load Spline Track Points nodes, and connect the Event BeginPlay node to the Clear Splint Points node. The final Blueprint network looks like this:

Click Compile at the top left corner of the Blueprint Editor, the following is the completed connections.



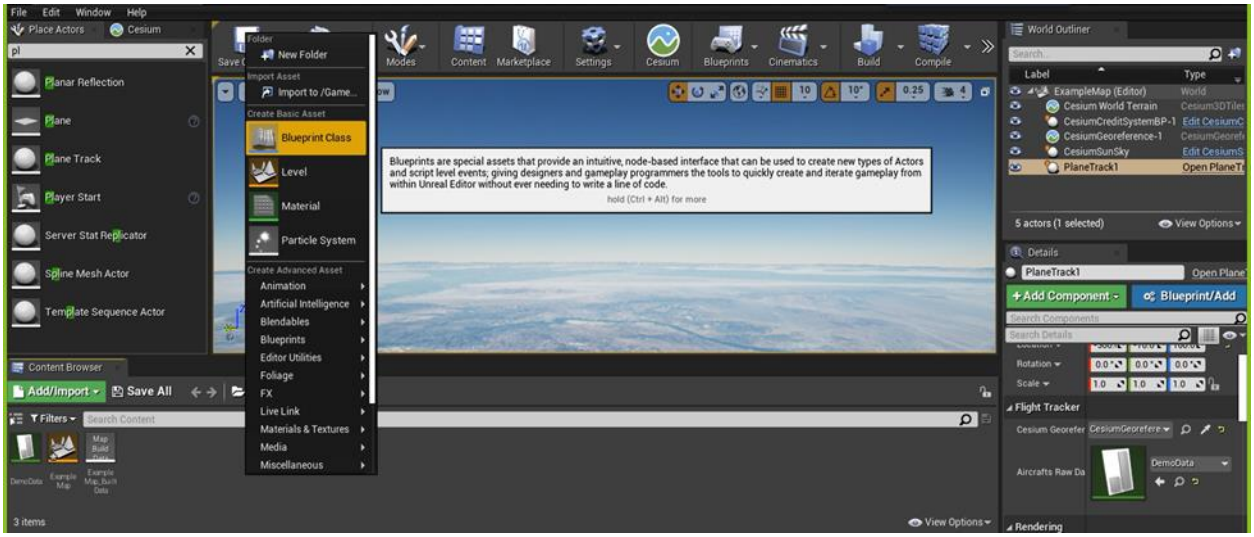
To check if everything is set up correctly, click the Play button in the top panel of the main editor. Since spline visualization is off by default, you can turn it on by selecting the viewport, pressing the` key (usually under the Esc key) on your keyboard and entering in the ShowFlag.Splines 1 command. You should be able to see the data points connected by a spline curve that starts in the air and spirals towards the ground.



Step 5: Add the Aircraft

The final step to complete the flight tracker is adding an actor that follows the spline path.

Right-click in a blank area of the Content Browser and select Blueprint Class.



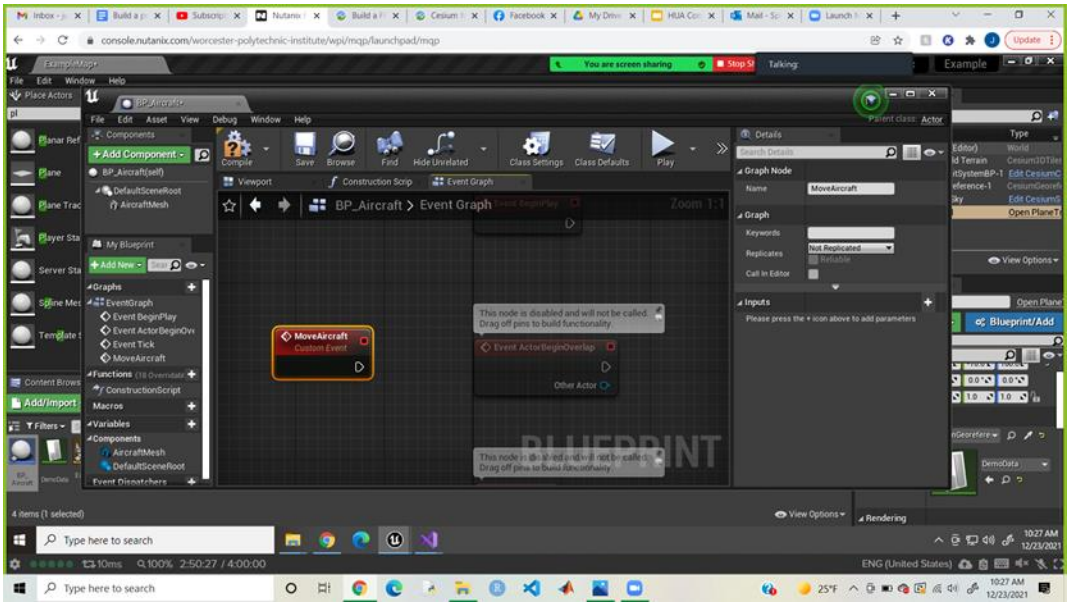
When prompted to pick the Parent Class, select Actor. Name the class “BP_Aircraft” (BP stands for Blueprint).

Double-click on the new Blueprint class to access the Blueprint Editor. Click on the green **Add Component** button at the top left corner

Search for “Static Mesh”. Add it to the component list and name it “AircraftMesh”.

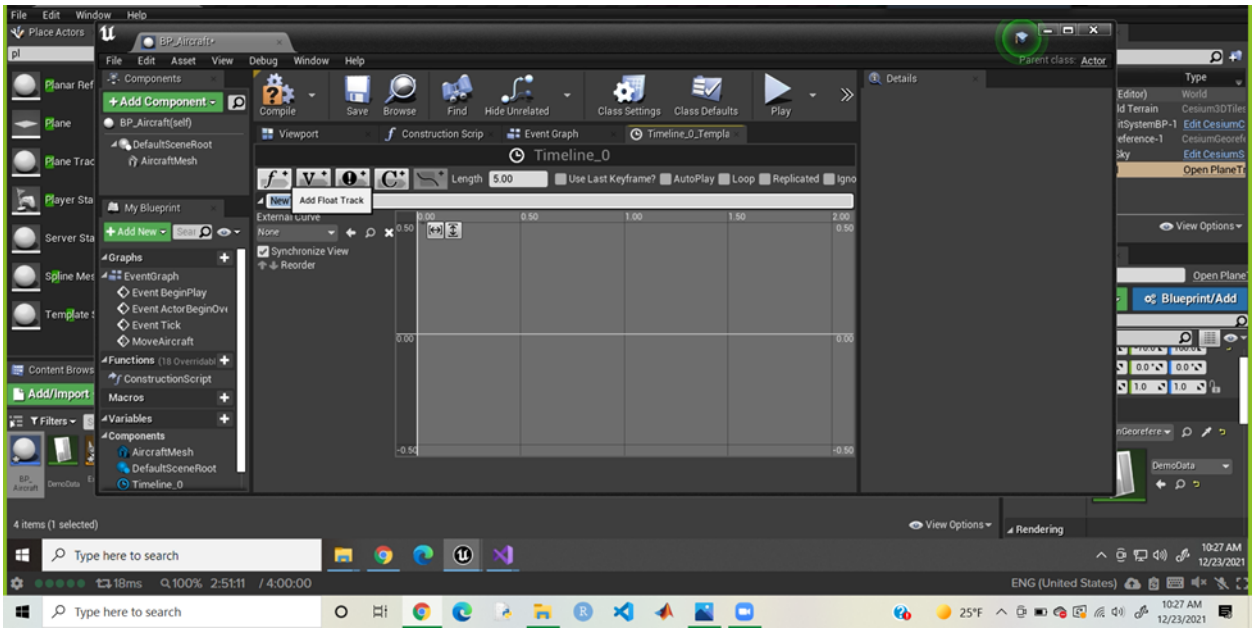
Head to the Event Graph by clicking the Event Graph tab at the top of the Blueprint Editor.

Right-click anywhere in the Event Graph and search for “Custom Event”. Name this event “MoveAircraft”. The following is what your editor should look like:



Right-click again in the Event Graph and search for “Add Timeline”.

Double-click on the Timeline node to open the Timeline Editor. Then create a float curve by clicking on the **Add Float Track** button at the top left corner of the editor and give it a name. In this tutorial, the Float Track is called “Alpha”.

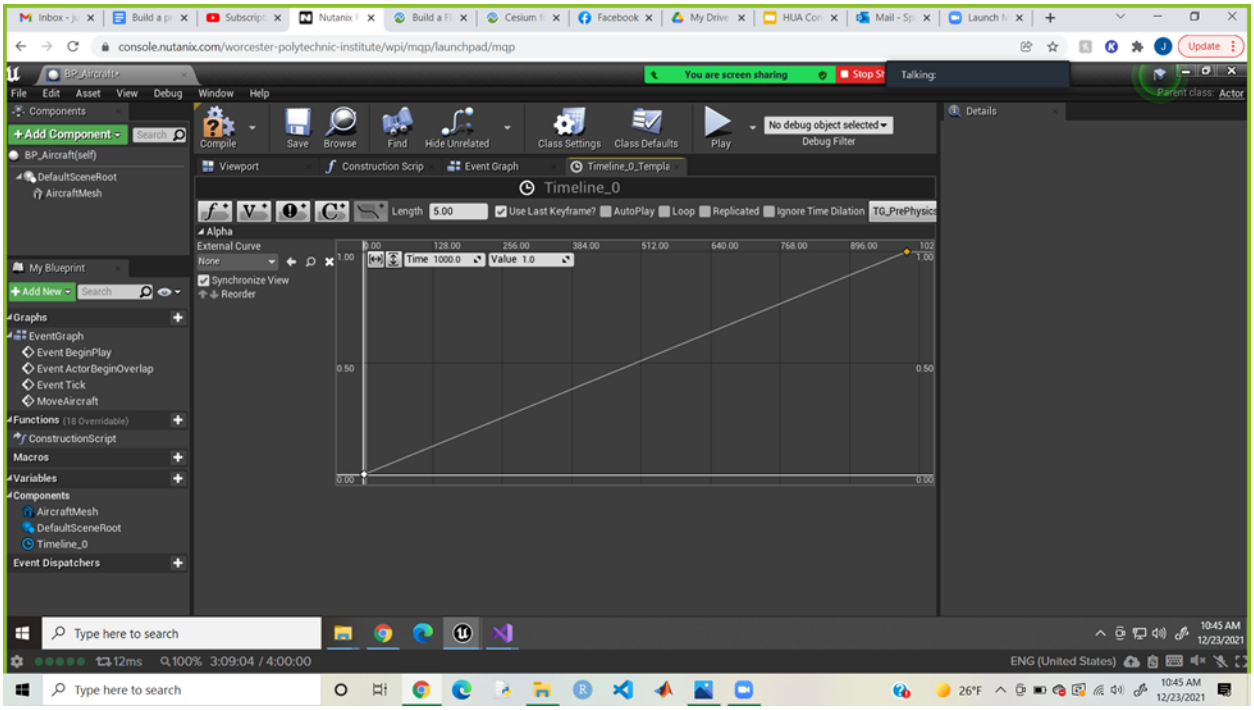


Add keyframes to the timeline by right-clicking on the curve and select Add key to Curve.

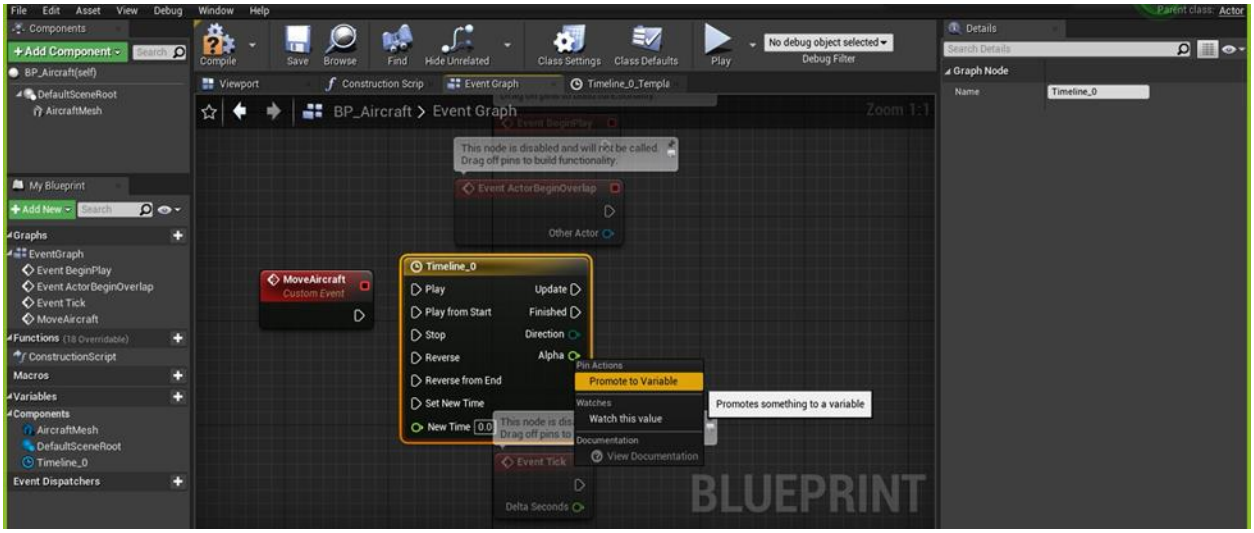
The first key point is at Time = 0, Value = 0. To assign precise values to a key point, select it and type in the values at the top left corner. Then add a second key point by again right-clicking on the curve and select Add key to Curve. This key point should be set at Time = 1, Value = 1000:

To adjust the view so you can see the whole curve, click the **Zoom for Fit Horizontal** and **Zoom to Fit Vertical** buttons in the top left-hand corner of the viewport.

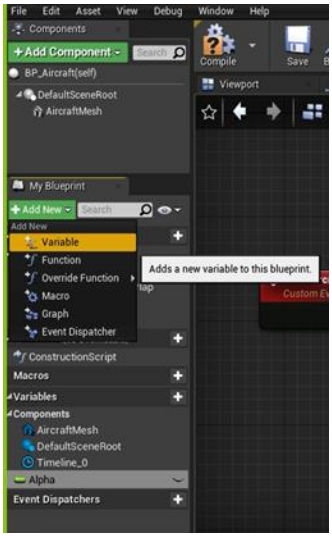
Then, at the top of the timeline window, set **Length** as 5 and check **Use Last Keyframe** if it is not already checked. Then, in the toolbar, select **Save**.



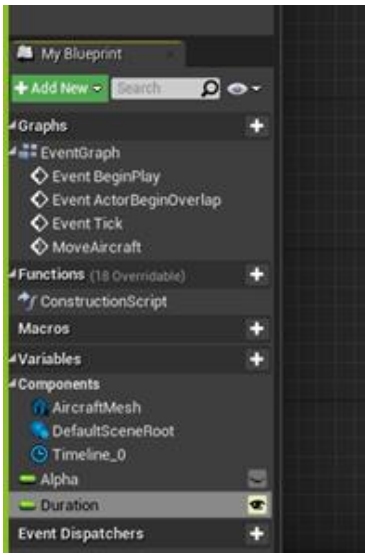
Head back to the Event Graph tab and right-click the Alpha output pin of the Timeline node and select Promote to variable. This will add a new variable to the left panel under Components:



Navigate to the My Blueprint panel, click the **Add New** button, and select **Variable** from the drop-down menu. Name the variable “Duration.” Give it a type of float.



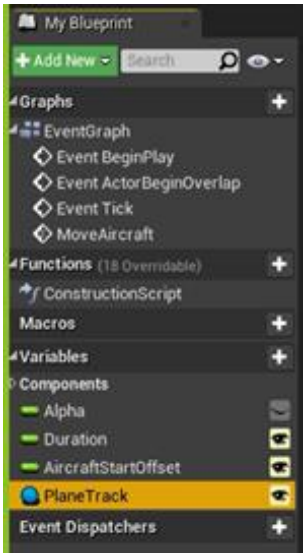
On the right side in the My Blueprint panel, click on the closed eye symbol next to duration to open the eye and make it public and editable in the main Unreal Engine editor.



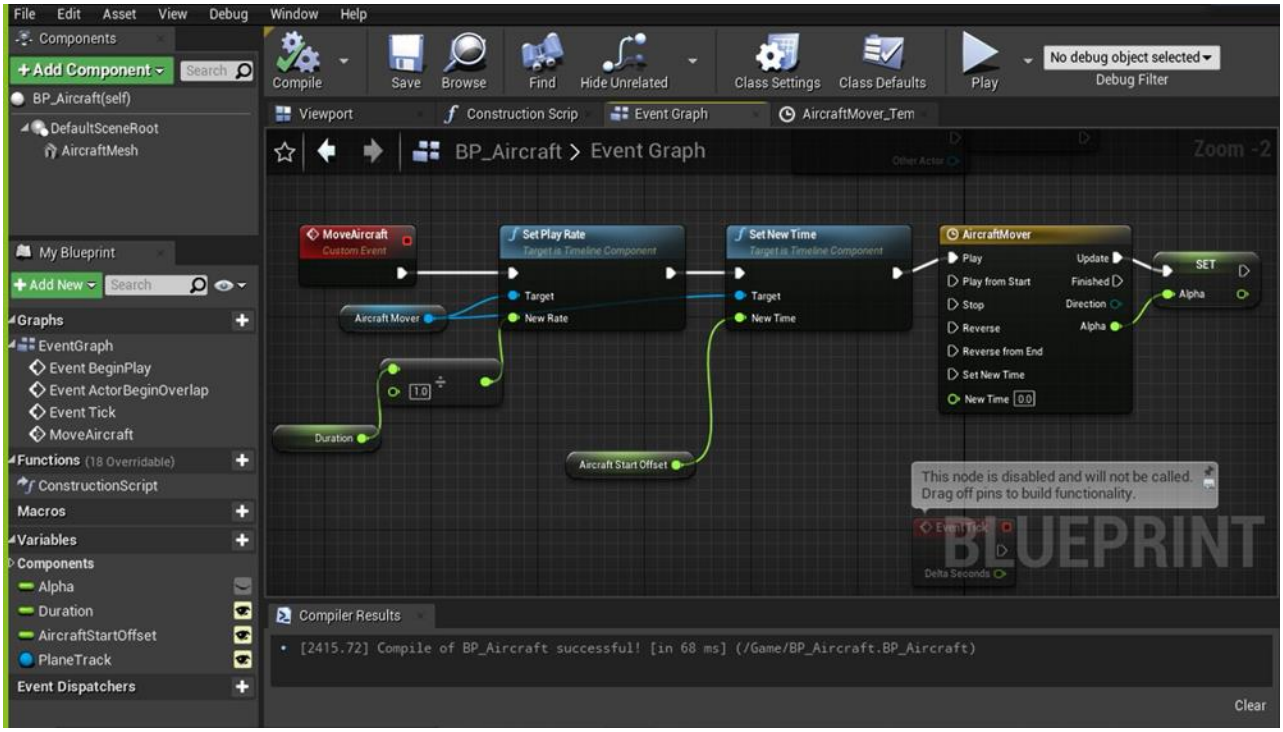
Similarly, add the following remaining variables to the Plane Blueprint class:

- **AircraftStartOffset**: type float; public visibility; The Slider Range and Value Range should be between 0 and 1, since the timeline goes between 0 and 1. These variables can be edited in the Details panel in the Blueprint Editor.
- **PlaneTrack**: type PlaneTrack with Object Reference; public visibility.

The final Components list looks like this:



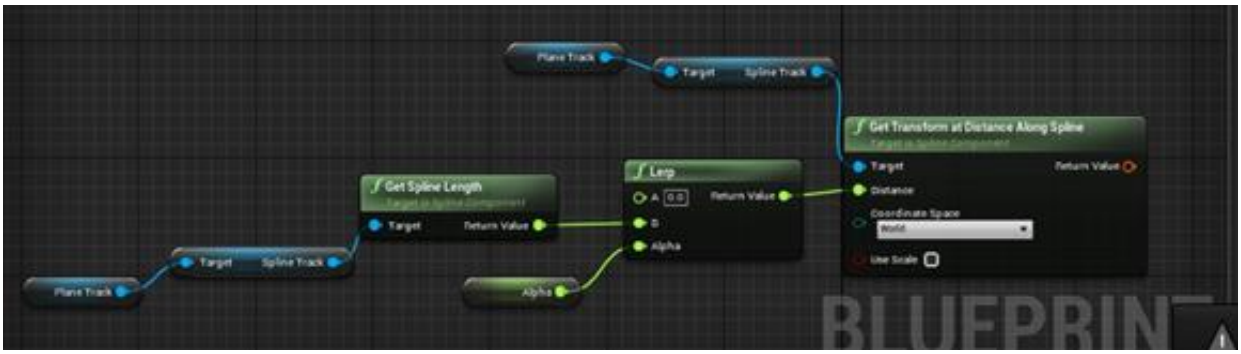
Right click on Timeline and rename it to “AircraftMover.” Then complete the rest of the Move Aircraft custom event as follows:



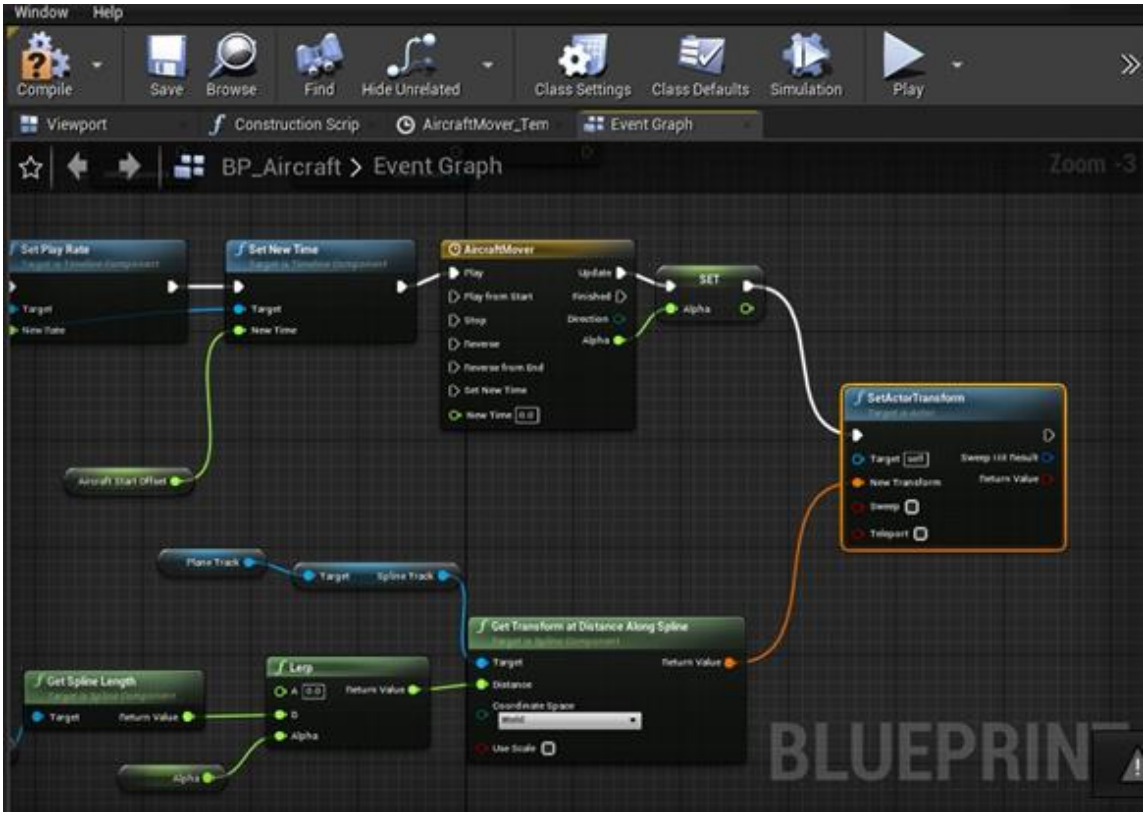
Note: when searching for a function make sure to uncheck ***Context Sensitive****. To use a variable, drag-and-drop it into the Event Graph or right-click anywhere in the graph and search for the variable’s name. To call a function that corresponds to a variable, drag a new connection from the variable node and search for the function’s name.*

Compile the Blueprint.

Next, in the same Event Graph, create the following node network below the Move Aircraft function:



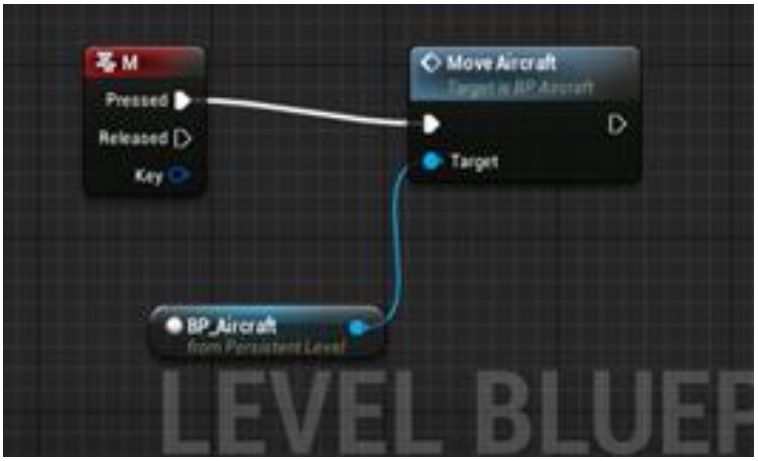
Lastly, connect the two Blueprint networks together with the Set Actor Transform node:



Compile the Blueprint.

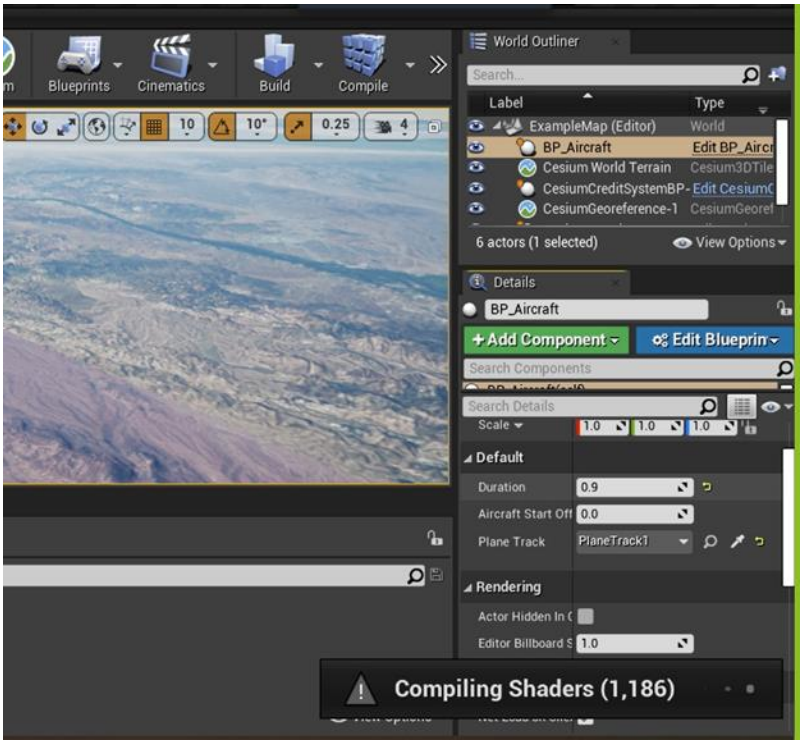
Navigate back to the main editor. Add your aircraft object to the scene by drag-and-dropping the BP_Aircraft Blueprint into the scene.

Head back to the Level Blueprint. In the Event Graph, add a Keyboard node (M is used here). Connect this Keyboard node to the Move Aircraft event as follow:



Compile the Blueprint.

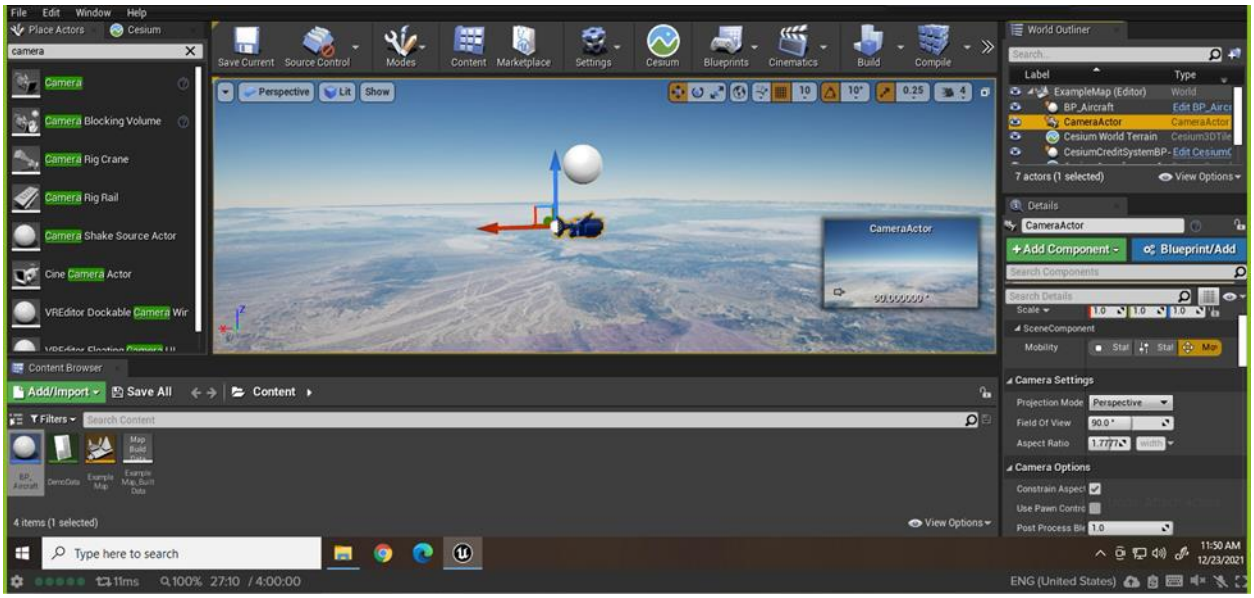
With the BP_Aircraft actor selected, head to the Details panel. Set PlaneTrack to PlaneTrack1 from the drop down. Set Duration to 0.9. Leave AirplaneOffset at 0.0. The smaller the duration the slower the flight.



With the viewport focused on the aircraft, click the Play button and press M (or the key that you chose to connect the Move Aircraft event to) to start the flight.

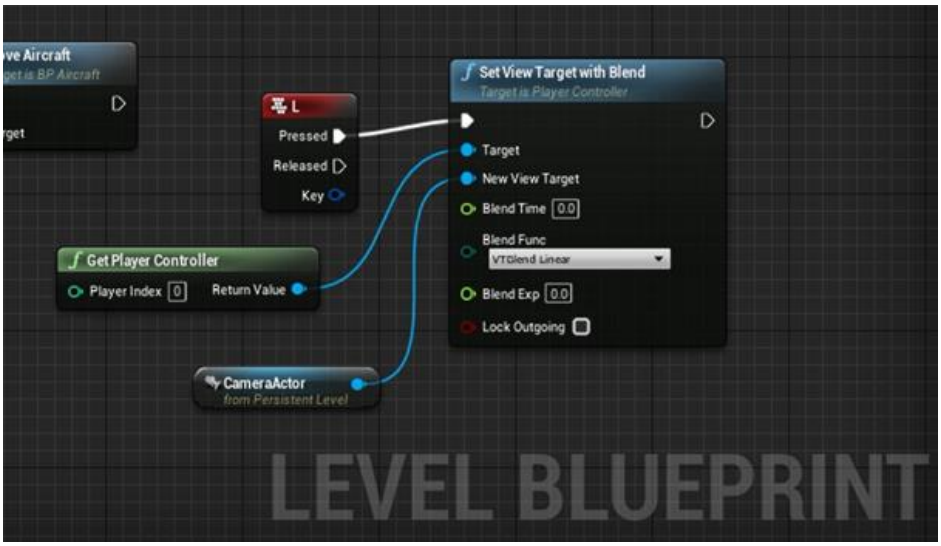
Step 6: Adding a Camera

Next, we will mount a camera to the parachute so we can follow the parachute as it drops. Using the Place Actor panel, add a Camera Actor to the scene.



In the World Outliner panel on the right, drag-and-drop the Camera Actor onto the BP_Aircraft actor so that it becomes BP_Aircraft actor's child. Then, click on the Camera Actor in the World Outline. In the Details panel, navigate to Transform. Change the green value of Rotation to -45.

Next, open the Level Blueprint. Add a custom keyboard event to change the view to the camera as seen below.



Save and compile the blueprint.

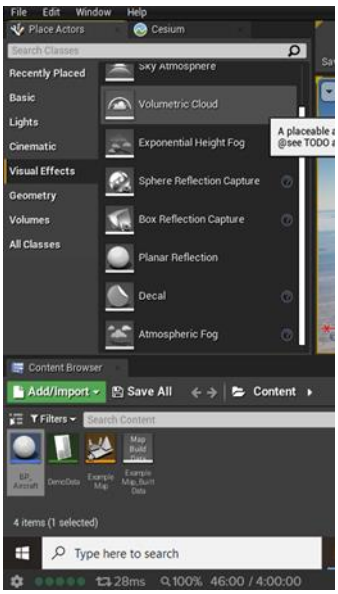
Environment Customizations

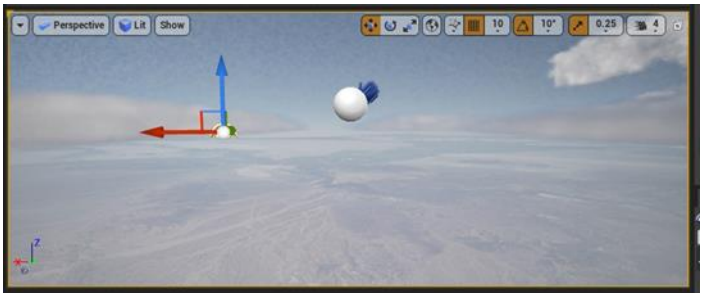
Time of Day

You can customize what time of day the game is in. Click on CesiumSunSky and in the details panel navigate to **Timezone**, there you can type in a precise value for time or click and drag the slide to adjust time of day.

Fog

There are multiple kinds of fog that can be added the a scene: Volumetric Cloud (our preference), Atmospheric Fog, and Exponential Height Fog. In the Place Actors panel select **Visual Effects** and you will see the various types of fog. Click and drag the desired fog into the viewport.





Handling the Data

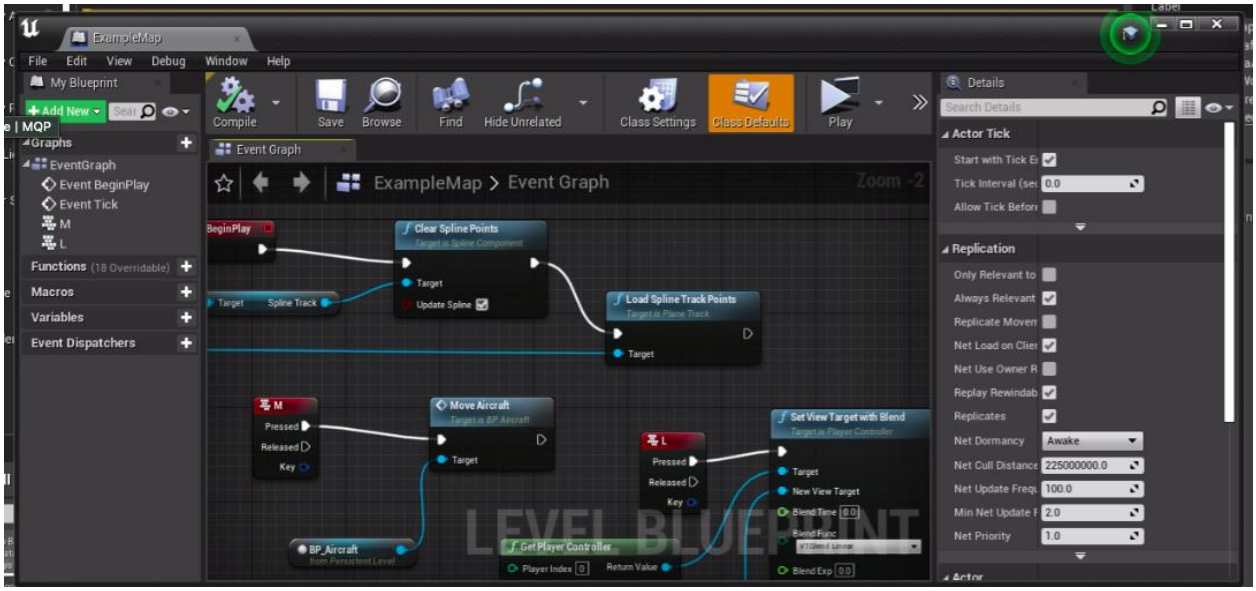
The goal of the project is to be able to collect data in the form of images taken from the drop and their corresponding location (latitude, longitude, and altitude). There are several ways to collect this data explained below.

Manual Data Collection

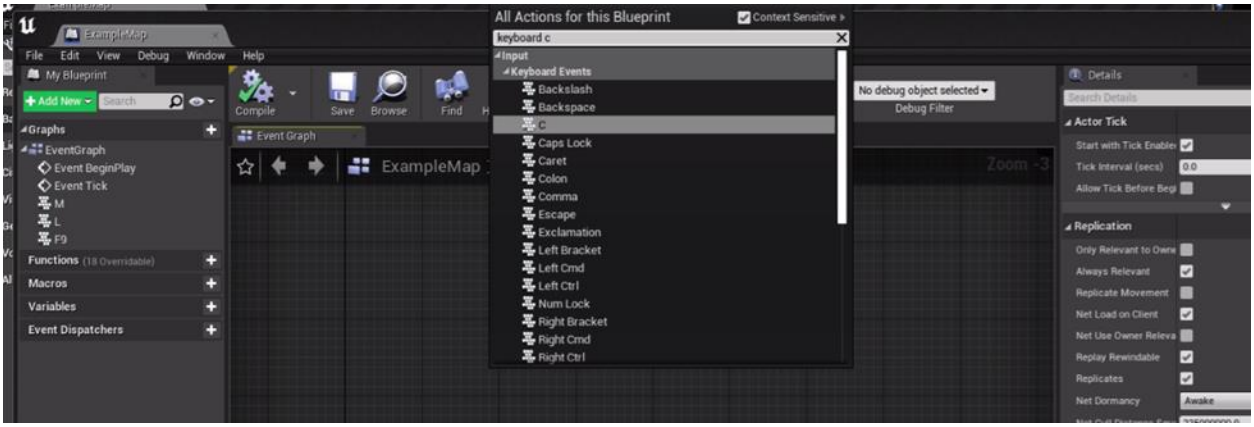
We will create a button command to take a screenshot and capture its location when we press the C key. This allows you to capture a screenshot and location at varying frequencies and however many times you wish.

To begin, open the **Level Blueprint.**

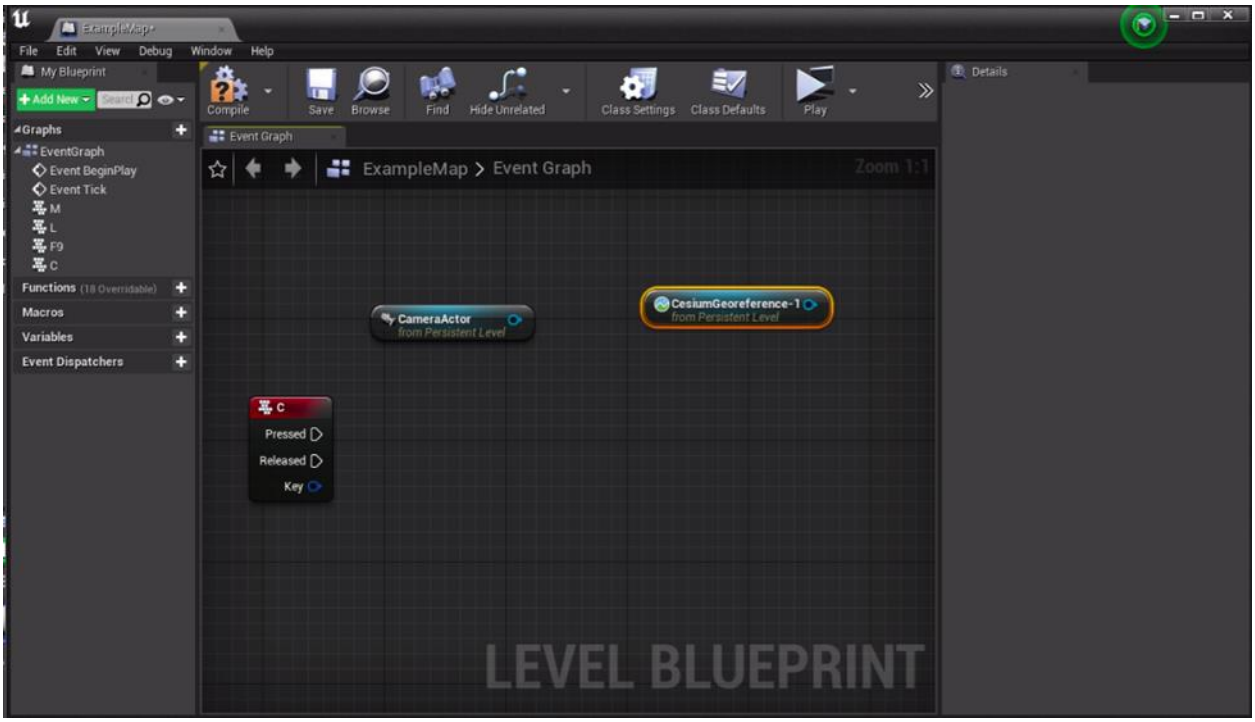
It will look like this:



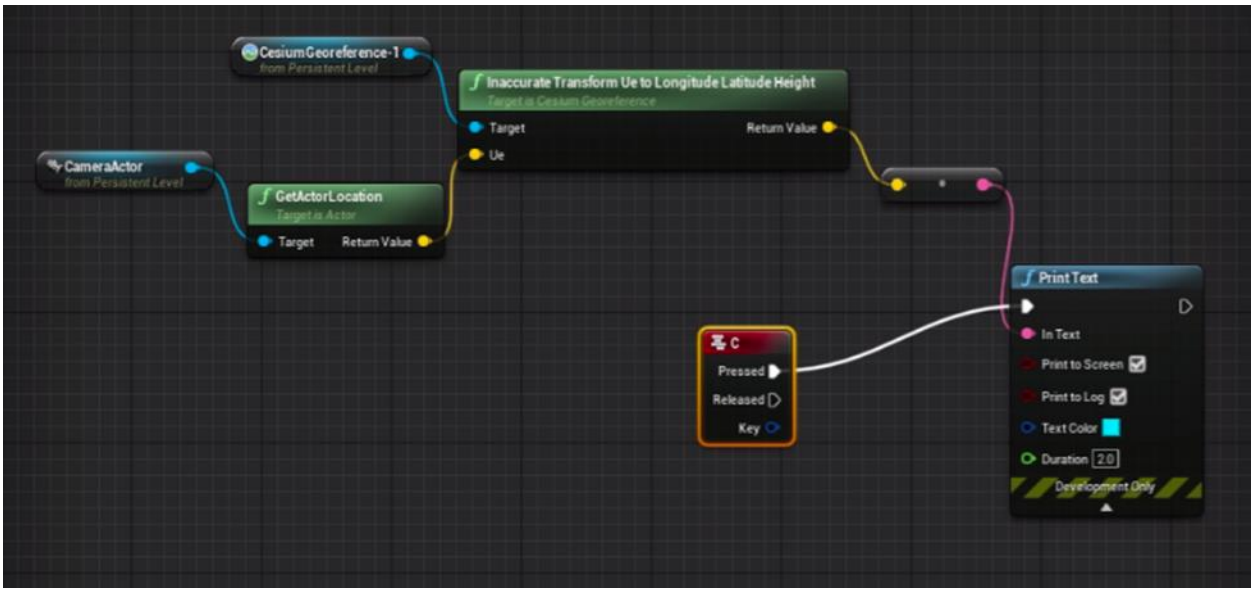
Right click and drag the background to move to a clear part of the blueprint. Then, Right click in the open space and search for ‘Keyboard C’. This will be the trigger key for an image and location datapoint.



Back in the main window editor, click and drag ‘CesiumGeoReference-1’ from the World Outliner to the blueprint window. Click and drag in the ’CameraActor’ in as well. They will appear as shown below:

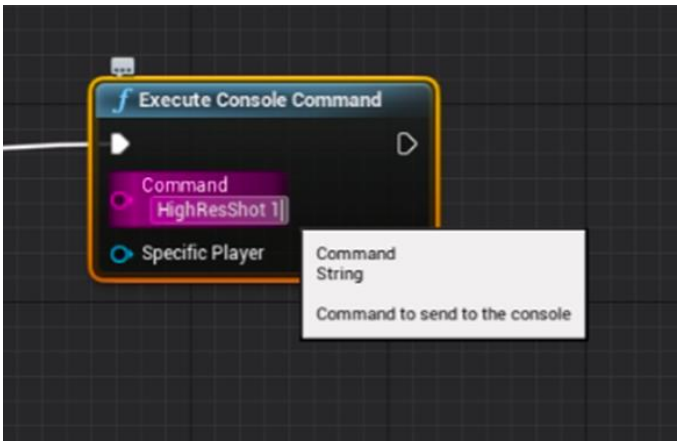


Right-click in the level blueprint to search and add the rest of the following nodes. Your final blueprint should like this:



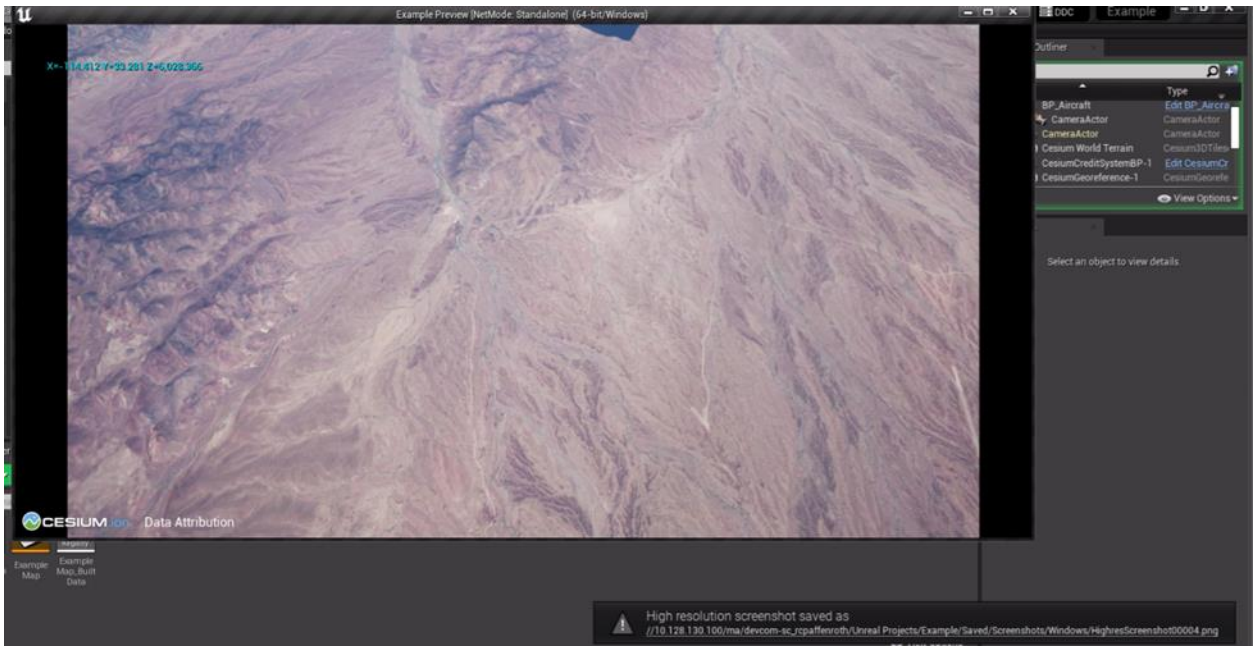
Make sure to click **Save** and then **Compile** at the top left corner of the blueprint window. This code builds the location collection function.

Next, we will build the code to capture screenshots with the same key, C. Click and drag the background to move to a clear spot in the blueprint. Right-click and search for ‘Keyboard C’ again. Then right-click and search for ‘execute console command.’ Connect the two nodes. In the Command text box within ‘Execute Console Command’ type in ‘HighResShot 1’



Click **Save** then **Compile** in the top left corner of the blueprint window. Return to the main editor and click **Save** there too. Then to test that it is working. Hit the play button in the top right corner to open the play window.

Then click ‘L’ to move to the camera view and then click ‘M’ to begin the drop. To take a screenshot click ‘C’ and you will see a message in the bottom right and top right corners seen here:



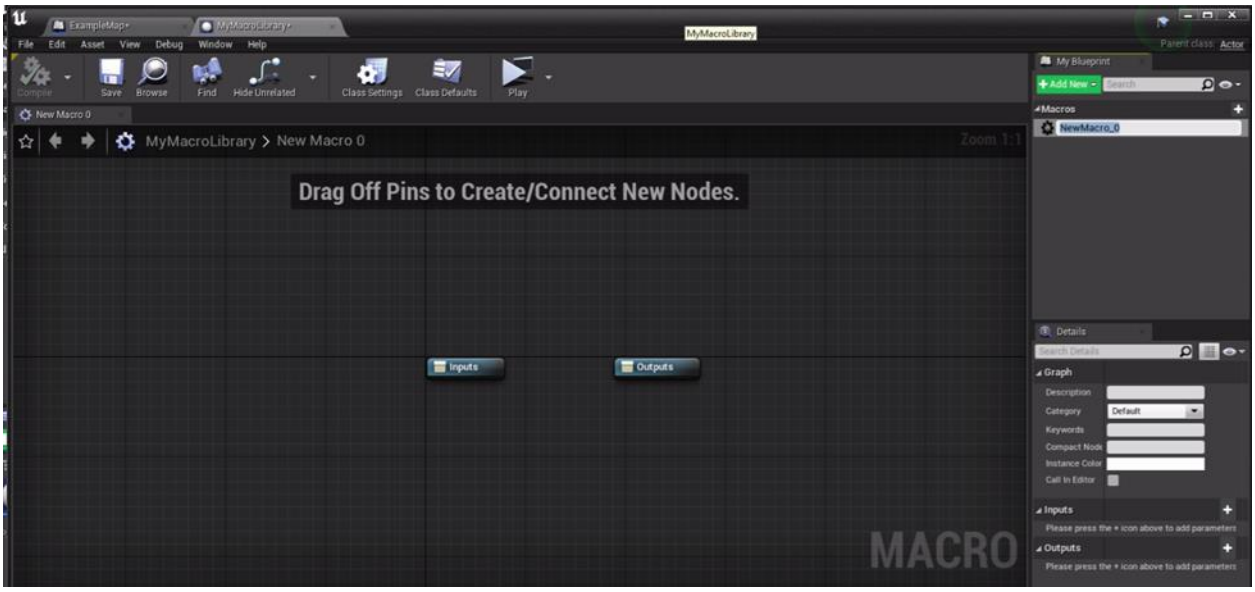
The message in the bottom right will indicate where the images are saved to. You'll see small blue text in the top left corner of the play window, in the previous image, that indicates the location of the parachute at the time the picture was taken.

Automatic Data Collection

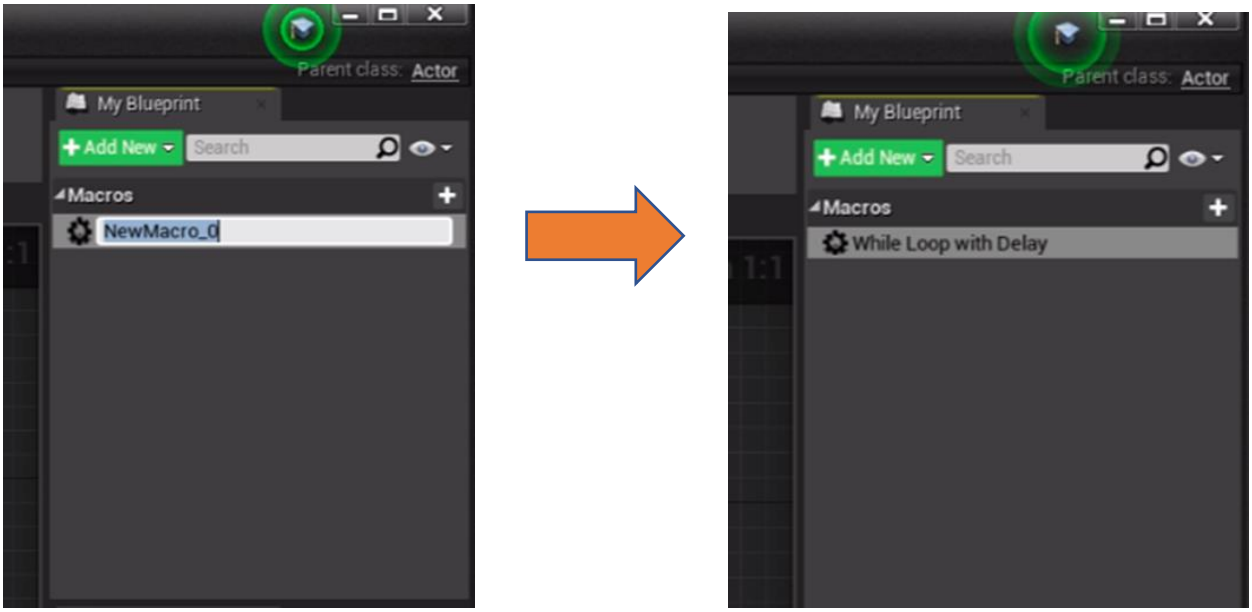
You can also add in a blueprint to automatically collect data at selected intervals.

To do this, first open the level blueprint. In the blueprint, click and drag the background to move to a clear part of the blueprint. Right click and search for 'Keyboard F' and select it. Next we have to customize a while loop. Navigate back to the main editor page. On the bottom of the editor, right click in the Content Browser and go to Blueprints → Blueprint Macro Library.

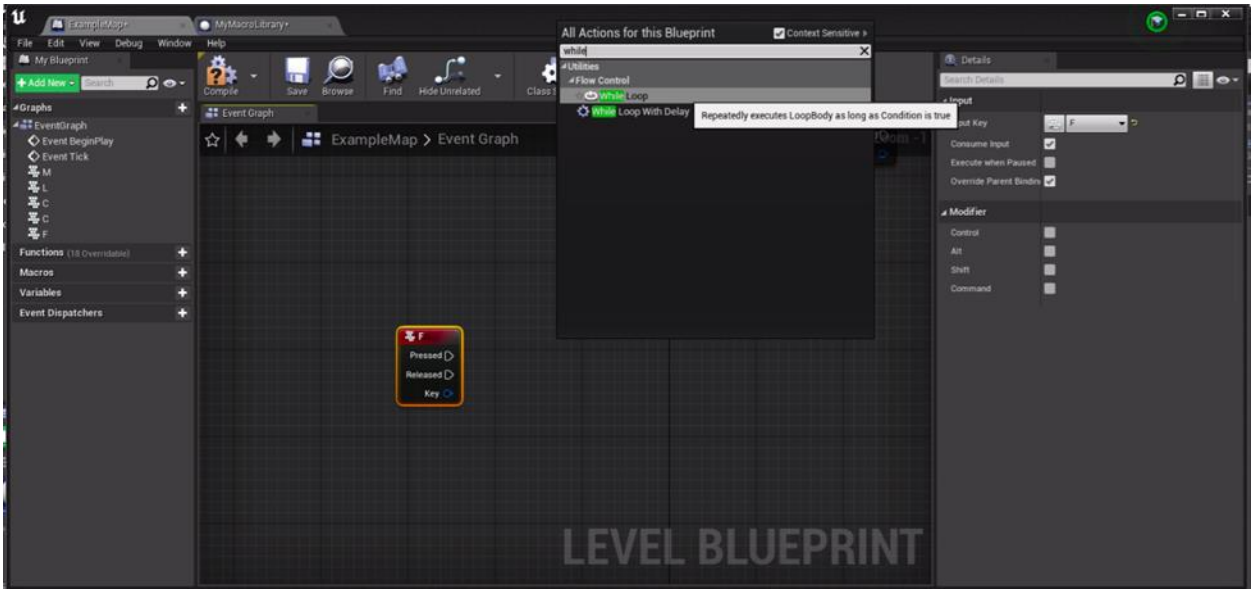
A dialog window will pop up, select **Actor** as the parent class. An object will be added to the content browser, rename it ‘MyMacroLibrary’. Then double click on **MyMacroLibrary** and it will open a new tab in the level blueprint window.



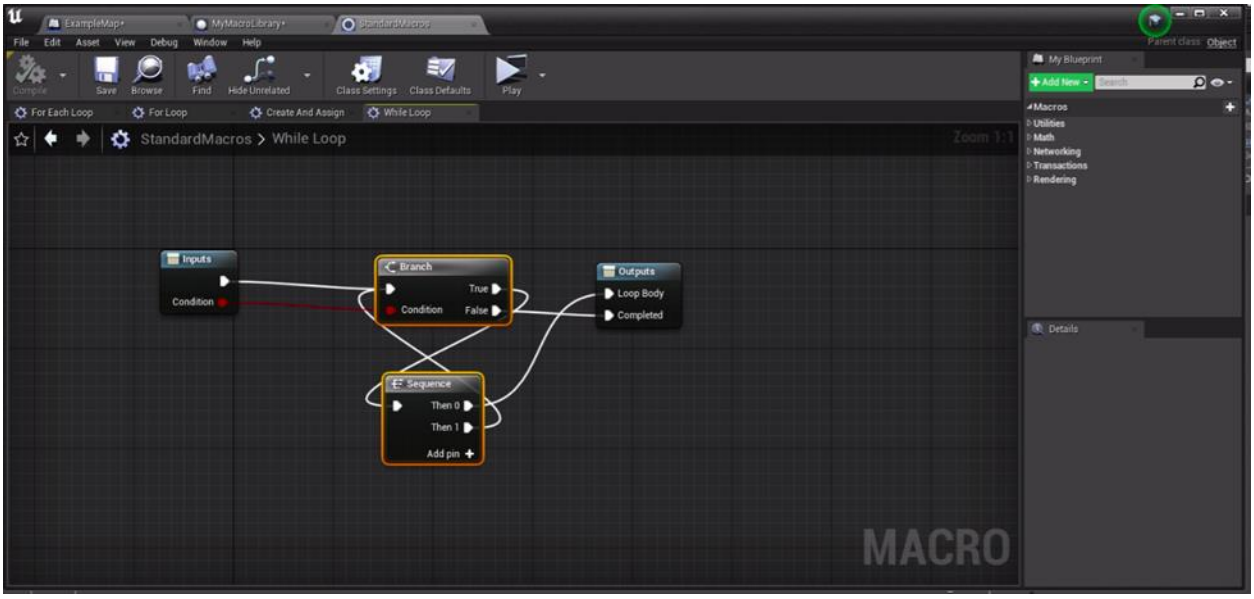
In the top right corner, there is a Blueprint tab. This custom node is considered a macro. By default, it is named ‘NewMacro_0’, but rename it to ‘While Loop with Delay’



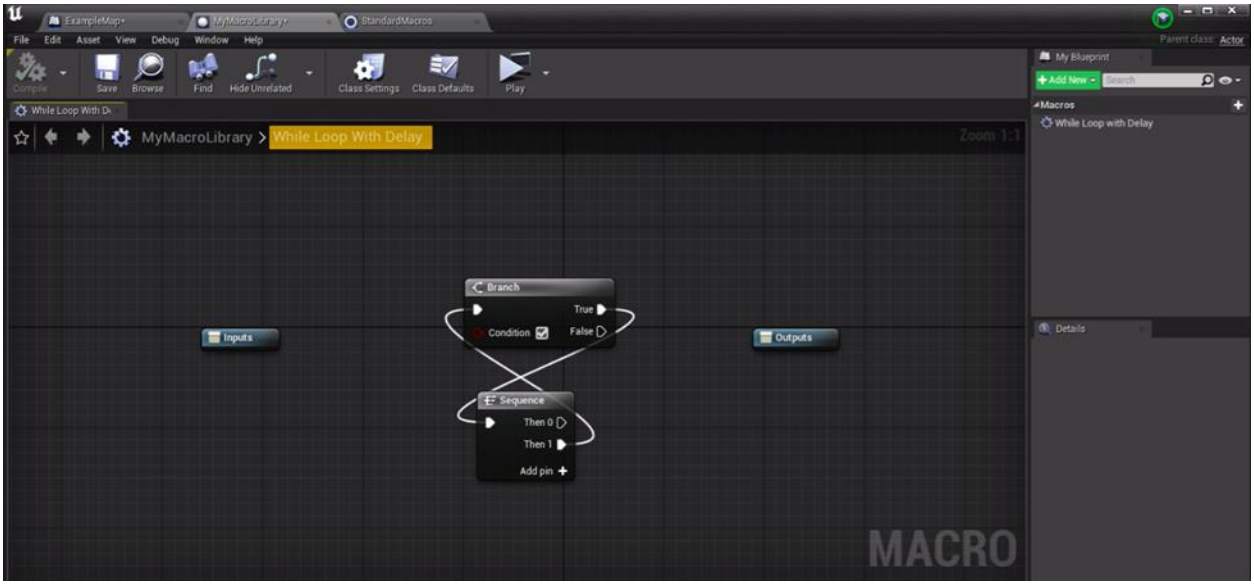
Then navigate back to the ExampleMap tab at the top of the window. Right click in the blueprint and search for ‘While Loop’



The While Loop node will appear. Double-click on it to open a new tab with the blueprint for this function. Drag and select everything between the Input and Output nodes and copy them:

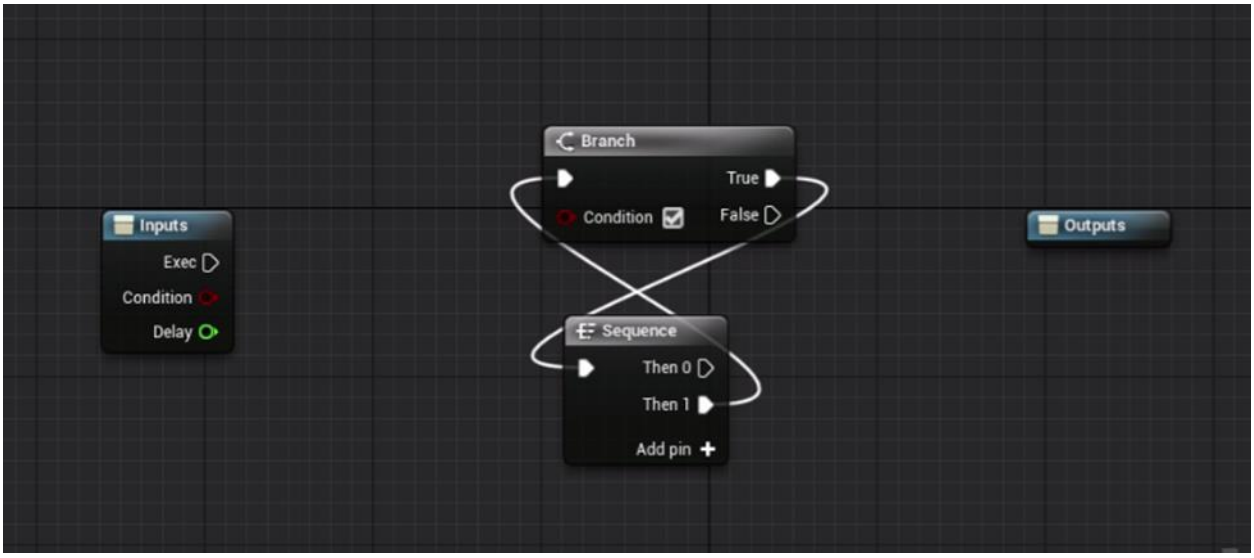


Then navigate to the MyMacroLibrary tab at the top of the window. Drag the Input and Output nodes further away from each other to create more space between them. Then paste in the node we copied earlier.



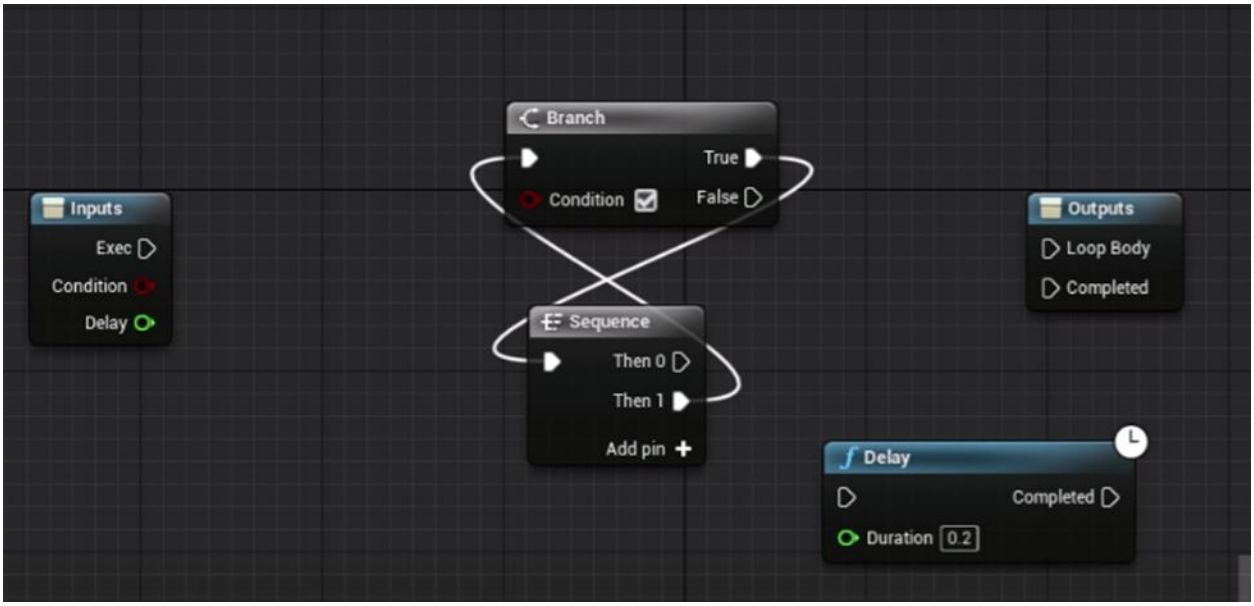
Click **While Loop with Delay** in the Blueprint tab on the right-hand side. Then navigate to the Details panel below it. Click **New Parameter** next to inputs. Rename the new input as ‘Exec’ and set the type to Exec with the white identifier. Then we will add two more inputs. The next one will be named ‘Condition’ with type Boolean with the red identifier. The last one will be named ‘Delay’ with type Float with the green identifier.

The inputs will automatically appear on the input node:

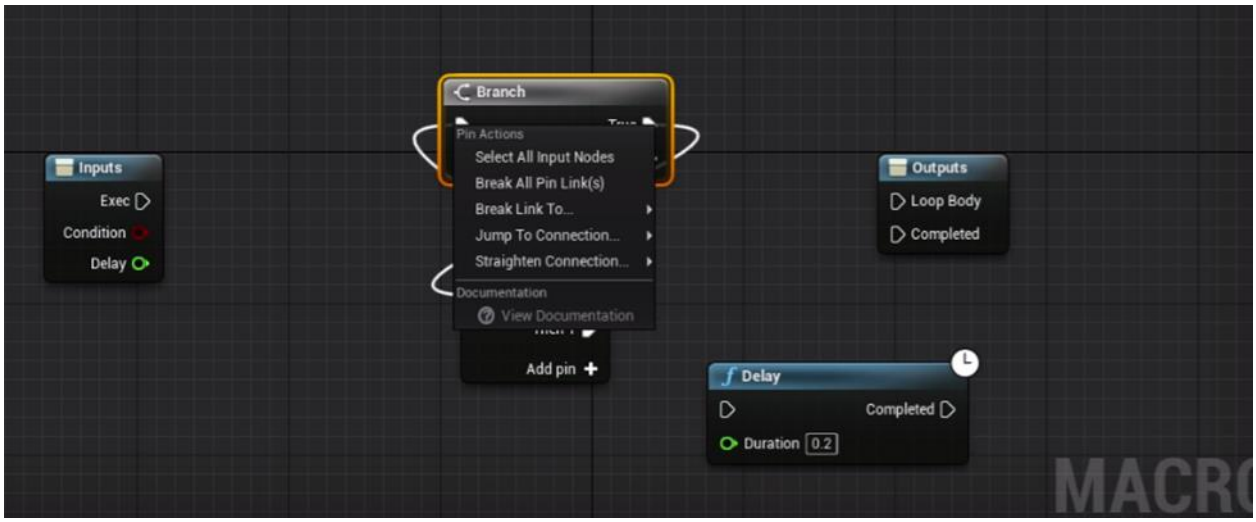


Now we will add parameters to the output node, which is also in the Details panel. The first one will be named ‘Loop Body’ with type Exec with the white identifier. The second one will be named ‘Completed’ with the type of Exec with the white identifier. The parameters will automatically appear of the output node:

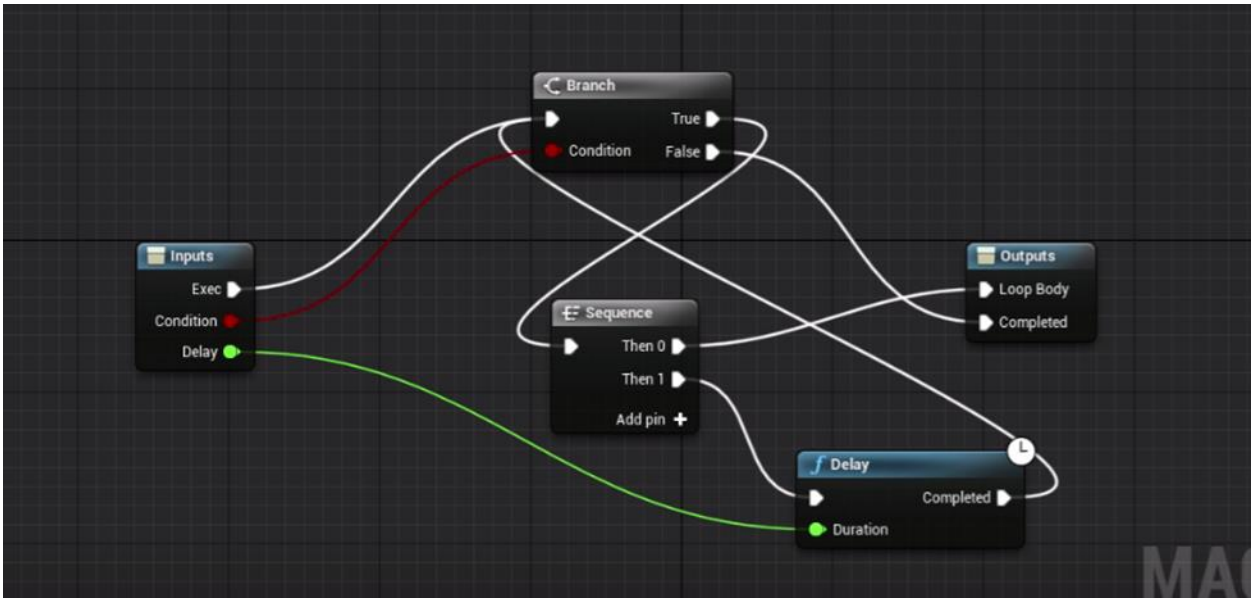
To the right of the Sequence node, right-click and search for Delay and select it.



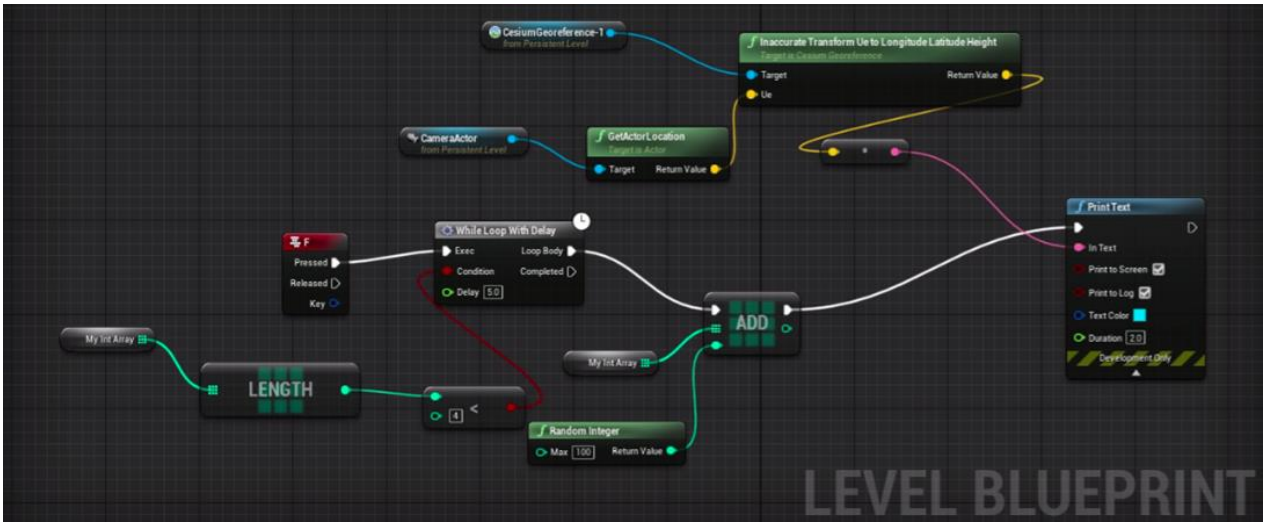
Next, we will break all the connections. Right click on the input pin for Branch and select **Break All Pin Link(s).** Repeat this for the input pin next to False on the Branch node.



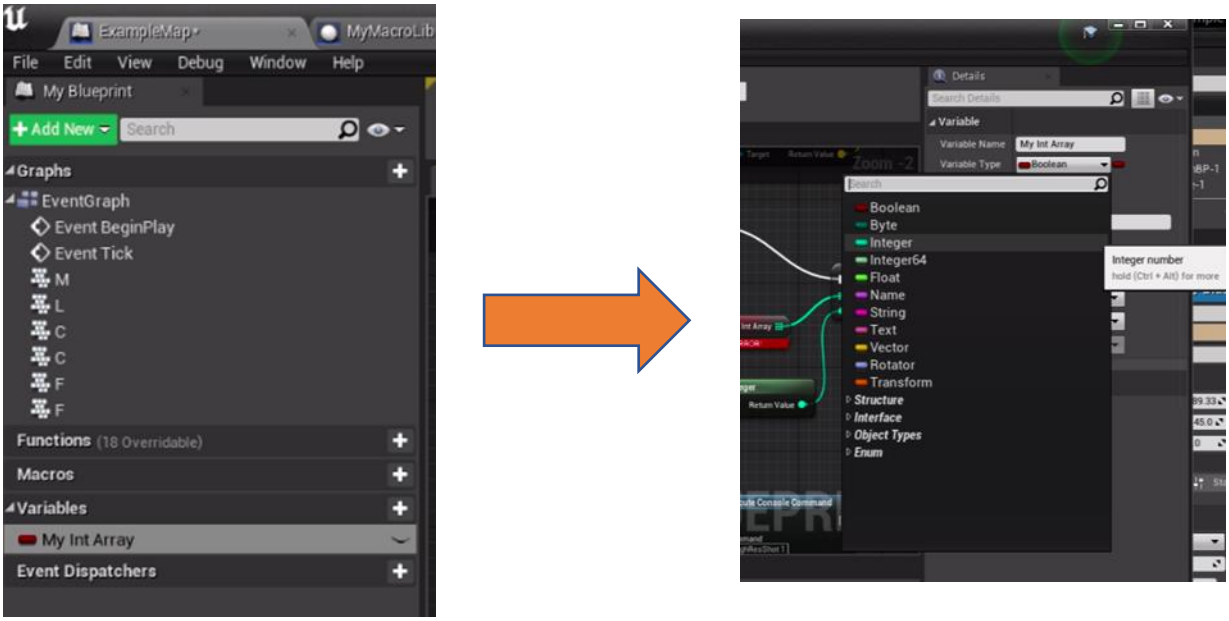
Then reconnect the nodes as seen below: Then click save in the top right.



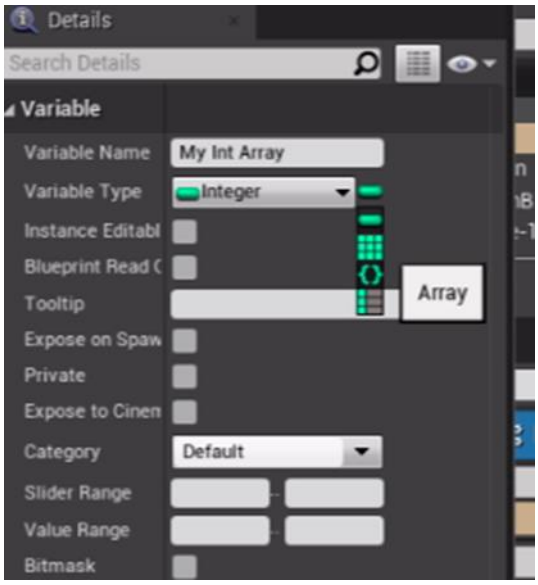
Navigate back to the ExampleMap tab and delete the While Loop node we had inserted earlier. Right-Click and search for ‘While Loop with Delay’ and select it. Complete the rest of the blueprint as follows:



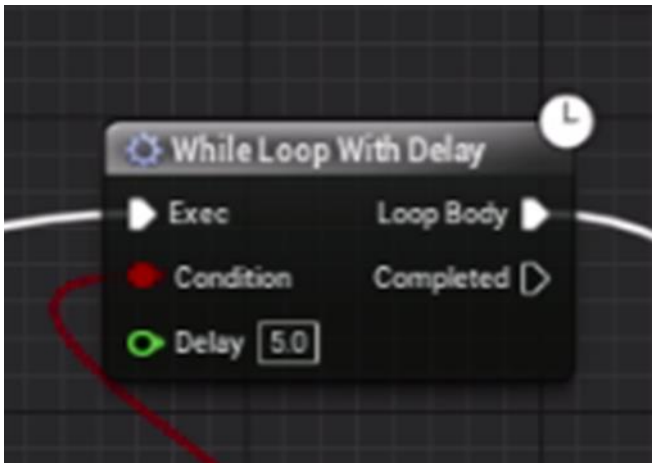
To create the array ‘My Int Array’ you must add a variable in the left-hand side and name it ‘My Int Array’. Then on the right-hand side the details panel will appear, set the type as integer.



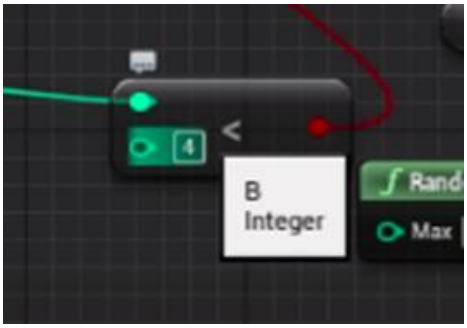
Next to the variable type, select the blue pill shaped icon and a drop down will appear. Select the 3x3 square icon to set it as an array. This variable can now be dragged and dropped into the blueprint.



This code will log the location every x seconds. To increase the time between collections, change the value of Delay in the While Loop with Delay. The picture shows it set to a 5 second delay. You can also adjust how many pictures it takes.

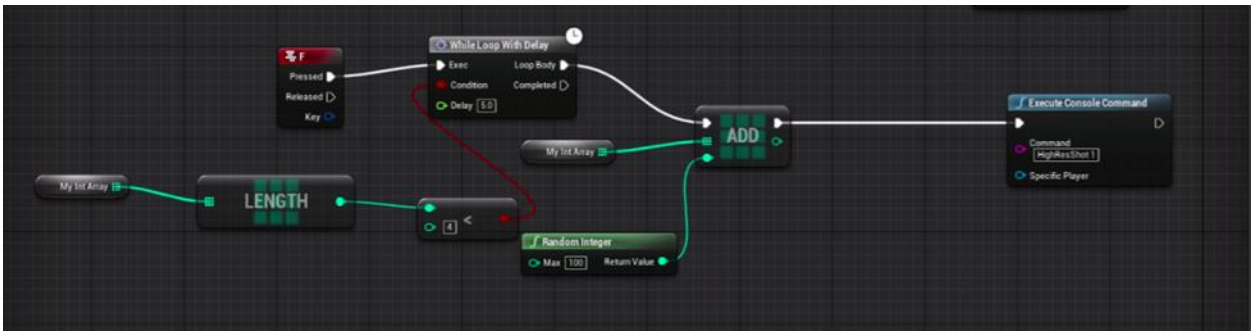


In the node shown below, the value entered is one more than the number of pictures taken. So, it is set to take 3 images.



Next, we will add in the nodes to take screenshots every time the location is logged.

Create the following section of the Blueprint. Make sure the keyboard control is the same as the location, in this case “Keyboard F.” Also make sure the values for the Delay and Number of Images are the same as the values for the location that we set above.

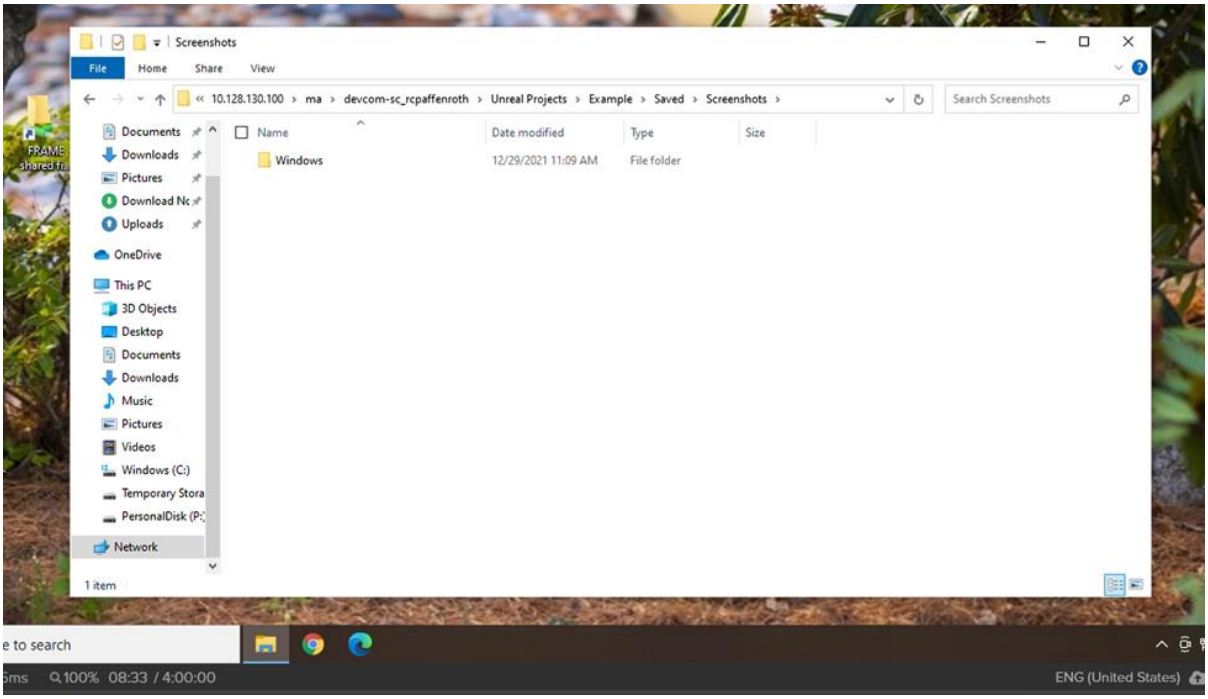


Click **Save** and **Compile** in the top right corner.

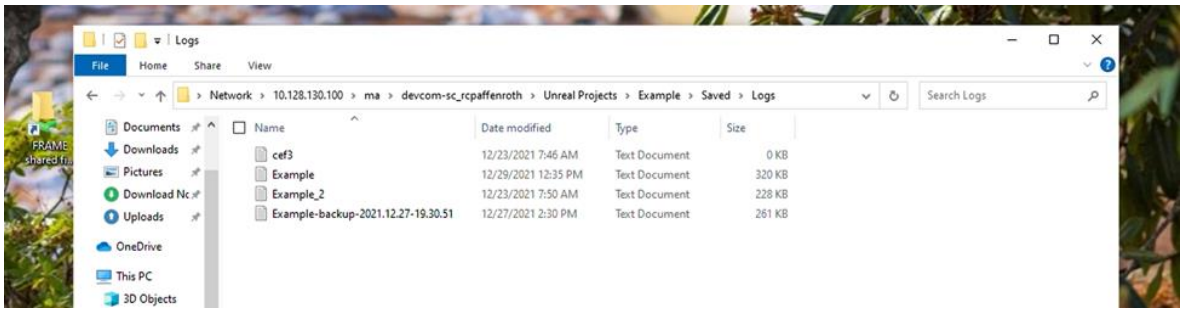
Data Organization

Organizing the data from the drops requires two pieces of information from the project file.

First you need to locate the folder where the screenshots were saved. This will be found at <yourProject>/Saved/Screenshots/Windows. This entire Windows folder can be moved or copied, since we will run a program that sorts through the whole folder.

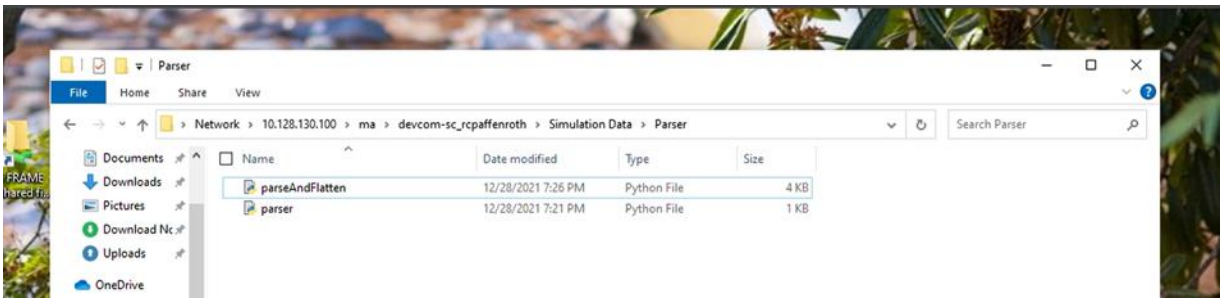


Next you will need to find the log file, which is where the location data for the images is saved. This should be saved in the folder <yourProject>/Saved/Logs. Sometimes multiple logs will be produced for a single run of the project, so to quickly check you have the correct one, within a log file you can search for ‘x=’ (by using the ctrl-f keyboard shortcut) and check that there are instances of that written into the log.



We advise you to make a new folder somewhere with ample space and move both the log file and the Windows folder containing the screenshots there.

Now, to parse the locations from the log, use the parser code we made, located on the desktop folder WPI Research Storage, in the Simulation Data/Parser folder.



The folder contains two important python scripts, the first of which is a basic parser, named parser.py, that compiles the locations of each picture in order and writes it to a csv file. These match up with each image in the Windows folder, in order from earliest to latest screenshot.

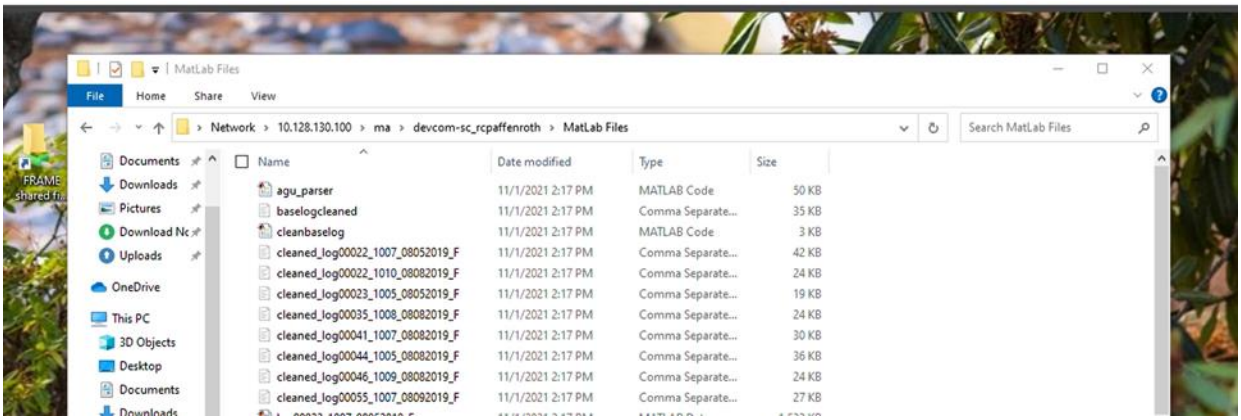
The other file, parseAndFlatten.py, also parses the log file and organizes locations, but also matches it to a flattened version of the associated image, along with the flatten first image from the series. It writes this data out as a pickle file, which is like a more efficient CSV file that is unviewable to users.

Before running either of these files, review the file paths written into the program to check they match with your new folder that contains the log and screenshots you want to parse, and the written files are named the way you'd like. Once this is confirmed, run the file.

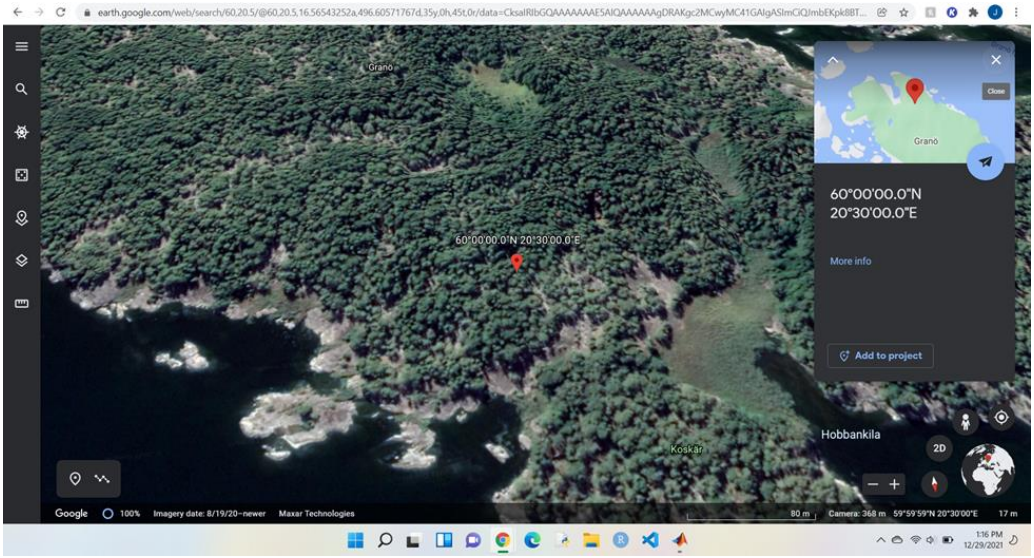
Alternate Drops

We have written code to easily build new CSV files for drops anywhere in the world. The code will follow the path of a real drop, which you specify, and the parachute will land where you choose.

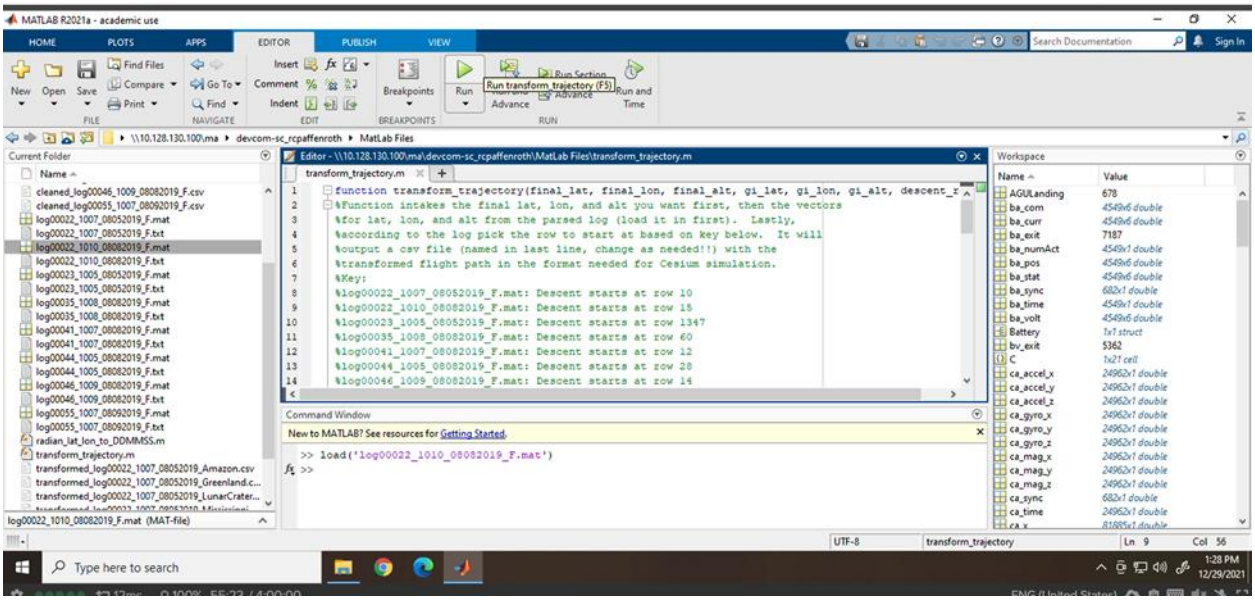
The files you will need to do this are found in the MATLAB Files folder located in the WPI Research Storage folder.



The file transform_trajectory.m contains code for creating data for a drop file based on a real drop path. The output is a CSV file compatible with our Unreal Engine simulator. The functions first three inputs are the final latitude, final longitude, and final altitude you want. These values can be found easily using Google Earth; your cursor's location is given in the bottom right-hand corner of the window. For example, if I'd like my drop to land in Sweden at (60, 20.5), I can type that into Google Earth, hover over the pin, and find the height of the ground there, 17 meters. You can also just scan the map for a location and similarly use your cursor to find a landing point.



Next, you'll need to know which drop path you'd like to follow, since the next three inputs are then the vectors for latitude, longitude, and altitude from the parsed drop log. These are easily available to the function by simply loading the desired file first, then using the given names of gi_lat, gi_lon, and gi_alt when you run the function. Eight pre-parsed logs are provided in the MATLAB Files Folder already, identified by their ".mat" ending.



So, if you wanted to follow the trajectory of the drop log00022_1010_08082019_F, you would load that file either by double clicking the .mat file in the left pane or by running the line “load('log00022_1010_08082019_F.mat')”, as shown above.

The last input for the function is the row to start at based on the key below. This simply discard values that were recorded while the parachute package was still in the plane:

Key:

log00022_1007_08052019_F.mat: Descent starts at row 10

log00022_1010_08082019_F.mat: Descent starts at row 15

log00023_1005_08052019_F.mat: Descent starts at row 1347

log00035_1008_08082019_F.mat: Descent starts at row 60

log00041_1007_08082019_F.mat: Descent starts at row 12

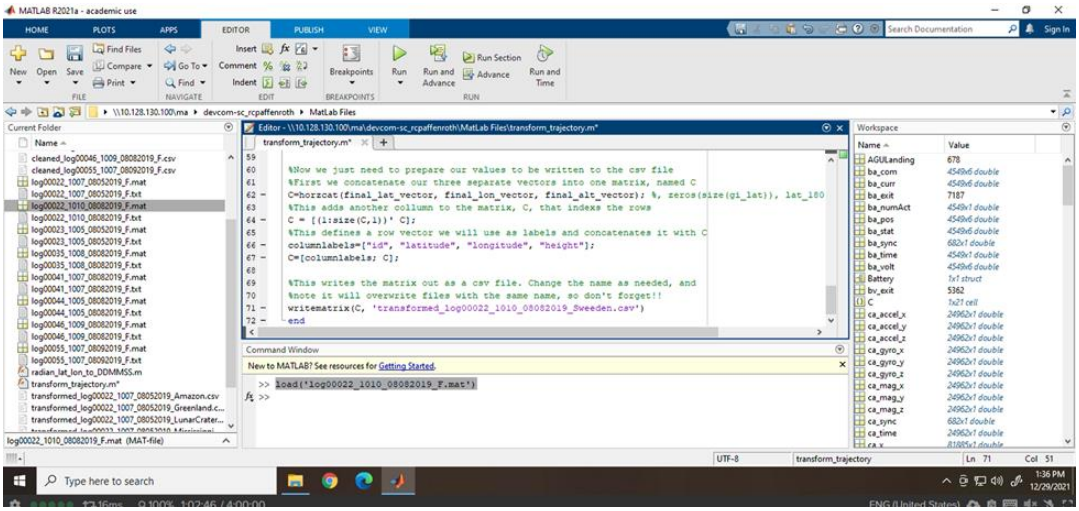
log00044_1005_08082019_F.mat: Descent starts at row 28

log00046_1009_08082019_F.mat: Descent starts at row 14

log00055_1007_08092019_F.mat: Descent starts at row 19

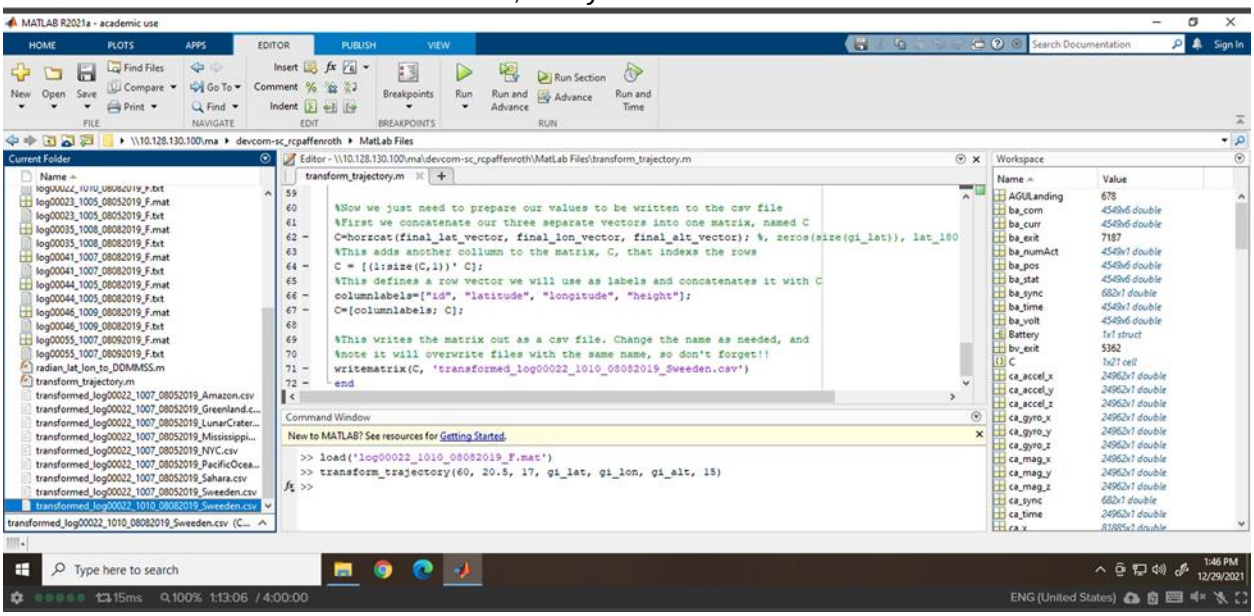
In the case of this example using the path from log00022_1010_08082019_F, you would input 15.

The last thing you need to do before running the file is setting the name of the output csv file you’d like. You can do this in line 71 of the file. You can change the name to anything you’d like, but make sure to maintain the .csv ending.



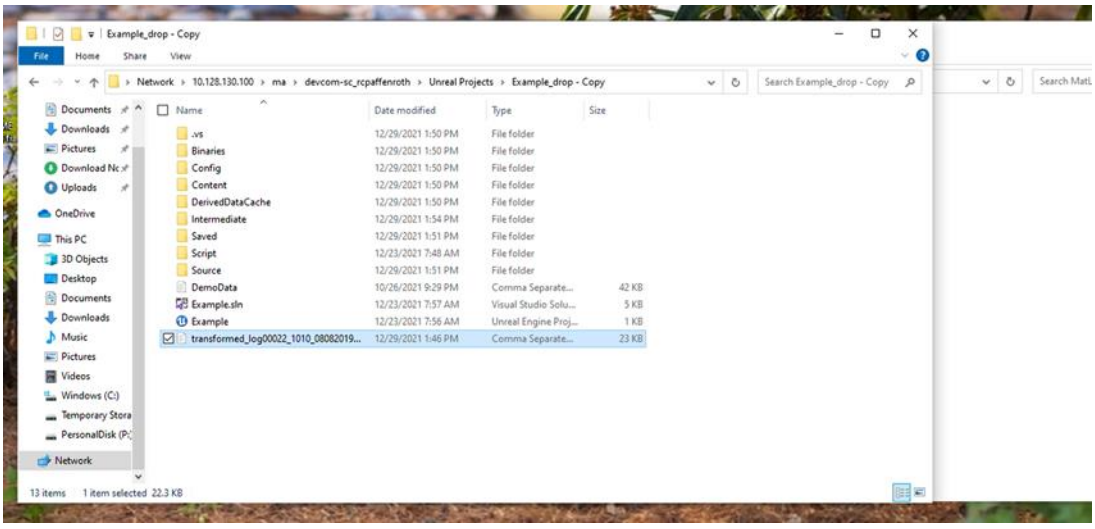
After making this change, make sure you click the **Save** button in the top left of the application.

After making this change, you could run a transform of the drop to the specified location in Sweden by typing the line: transform_trajectory(60, 20.5, 17, gi_lat, gi_lon, gi_alt, 15) The desired csv file will be written into the MATLAB Files folder, and you can use it in a simulation.

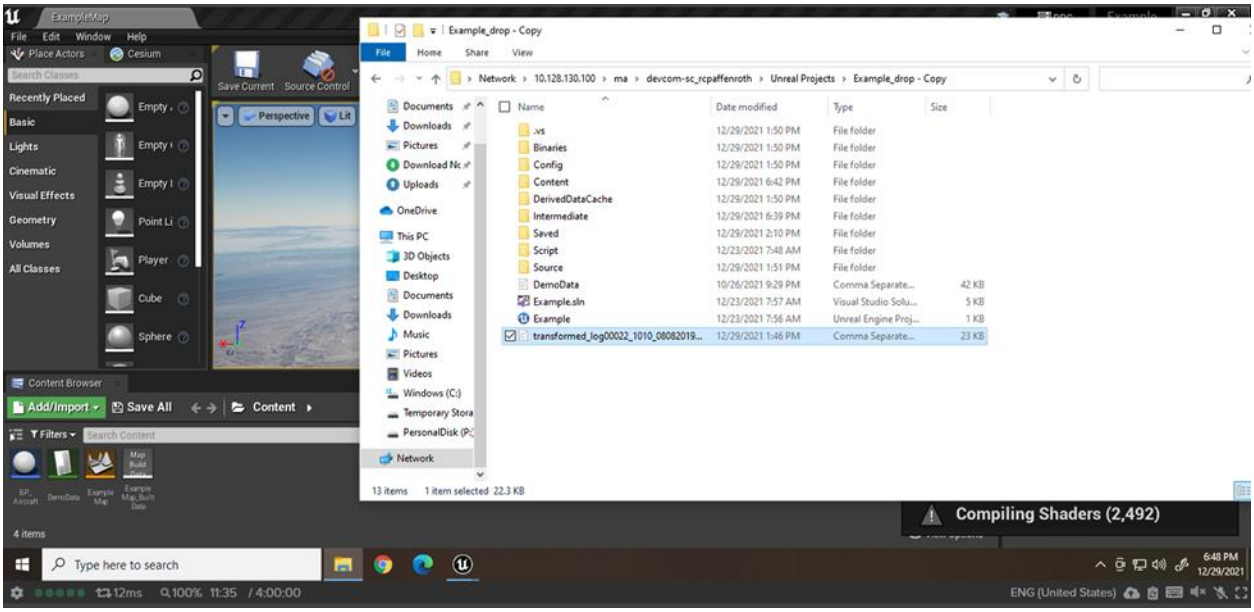


Note: the file cleanbaselog.m works very similarly, but instead of transforming the location, it simply cleans the path or a drop and writes it to a csv in the appropriate structure to be used in the simulator.

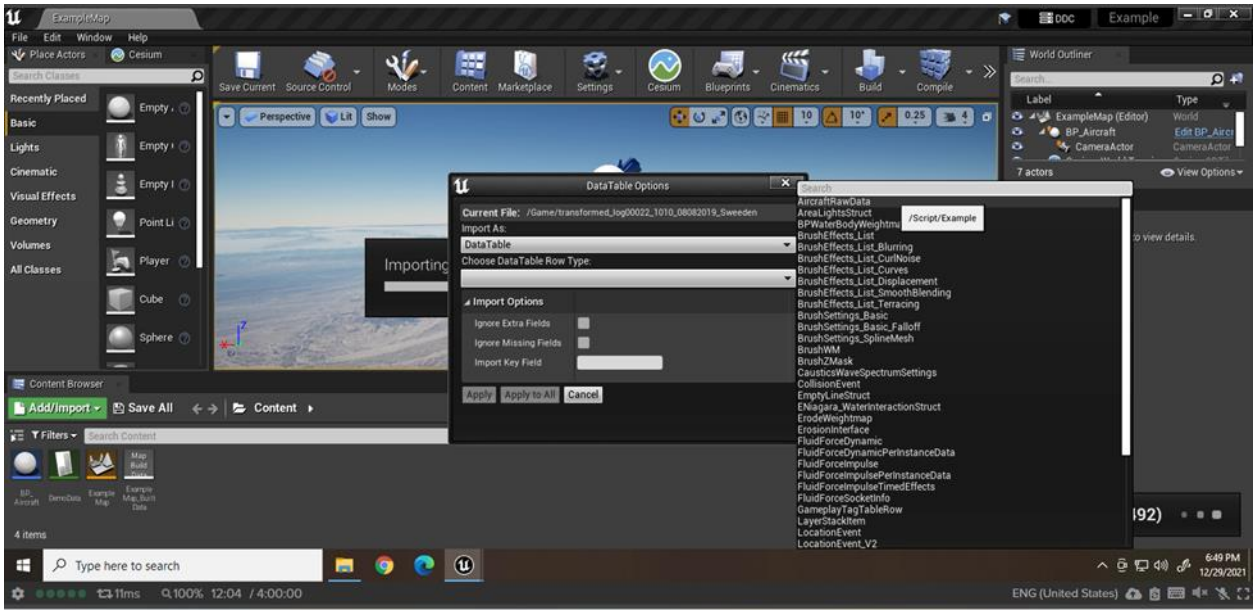
Now we will load the new CSV file into our simulation. First, move the new CSV file to the project folder you're using to make this simulation.



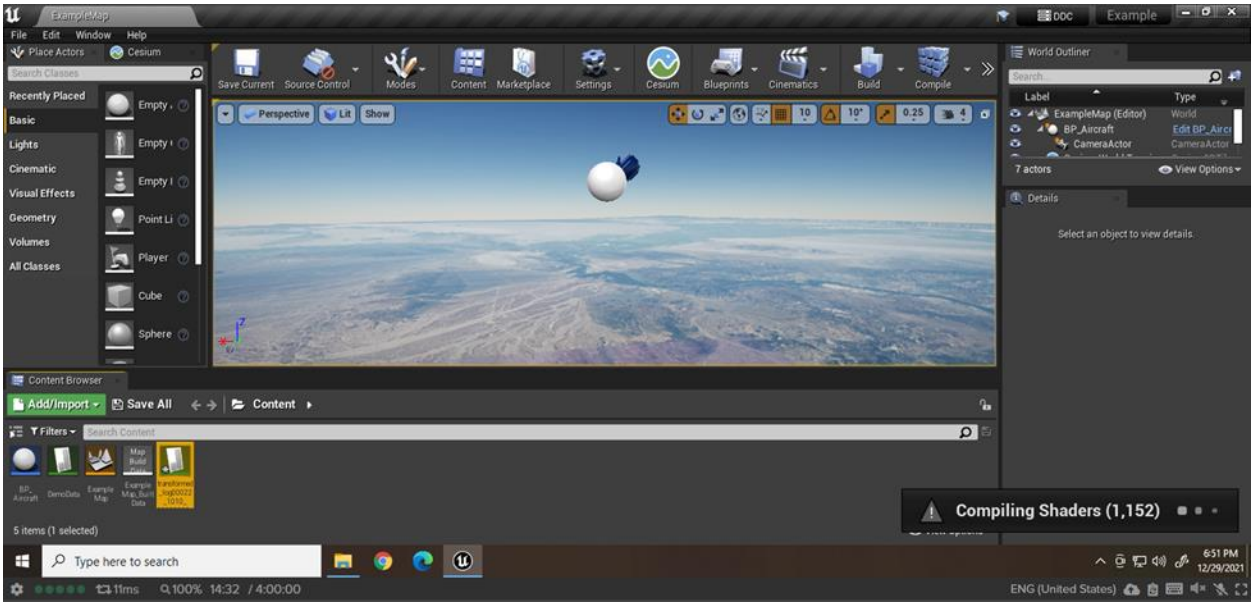
Next you will drag-and-drop the .csv data file into the Unreal Engine Content Browser. This is the panel at the bottom of your editor.



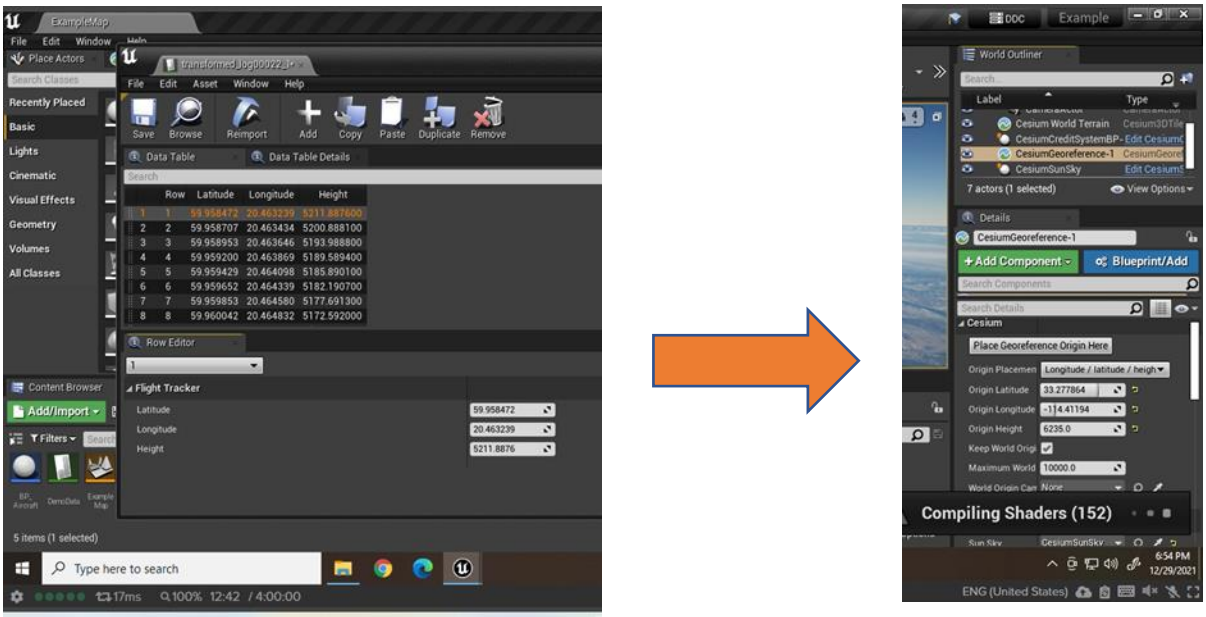
After you drop in the file, a pop-up window will appear. In the **Choose DataTable Row Type** dropdown select **AircraftRawData** (typically the first option):



Click **Apply**. Then to check the data uploaded correctly, double-click on the file in the Content Browser to open the data table:



Note the first line of the new data table, next you will input these values to position your view to the drop location. In the World Outliner on the left, select the CesiumGeoreference object.

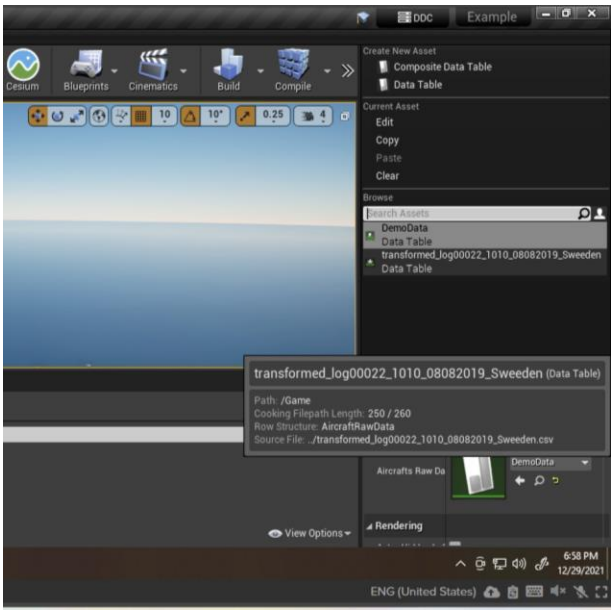


Here you want to change your location in this tab to be the first coordinated in the data file, in this case:

Origin Latitude = 59.958472; Origin Longitude = 20.463239; Origin Height = 5211.887600

If the scene appears dark, adjust the time of day by selecting the CesiumSunSky actor in the World outliner Panel and adjusting the **Time Zone** slider.

Select the PlaneTrack actor. In the Details panel, then set the Aircrafts Raw Data Table variable to the data table you added.



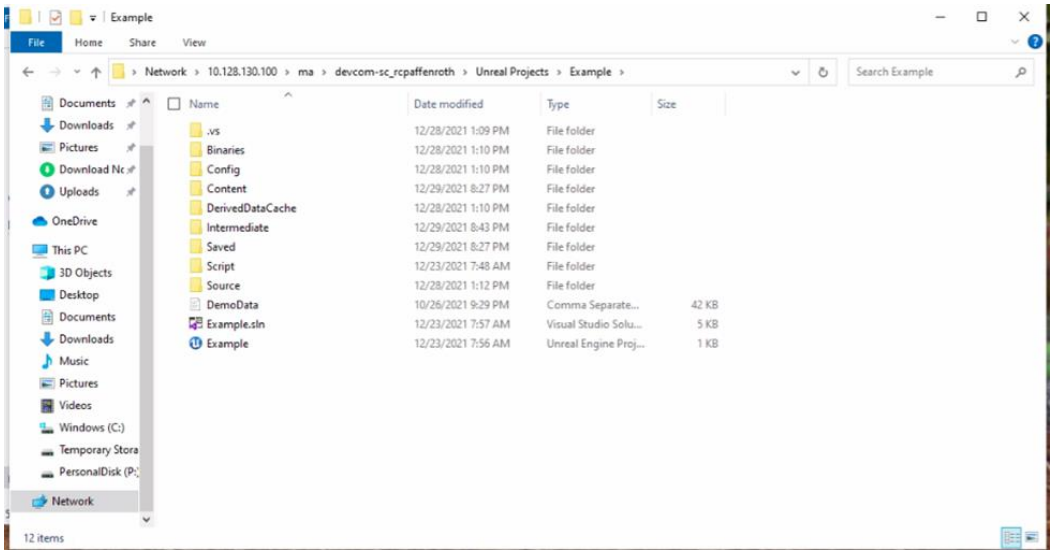
Finally, **Save** the project. Now you will have a simulated drop in the new location according to your CSV file.

User Guide

These are instructions on how to open and run a project, load in flight files, and collect data. We will be using our project named ‘Example and load in a new location, fly the drone, and collect data.

Open and Load Files

Each Unreal Engine project is saved with its own folder. Open the project folder and double click on the <project_name>, in this case named Example. The file type will be Unreal Engine Project. It will take a couple minutes to load the project.

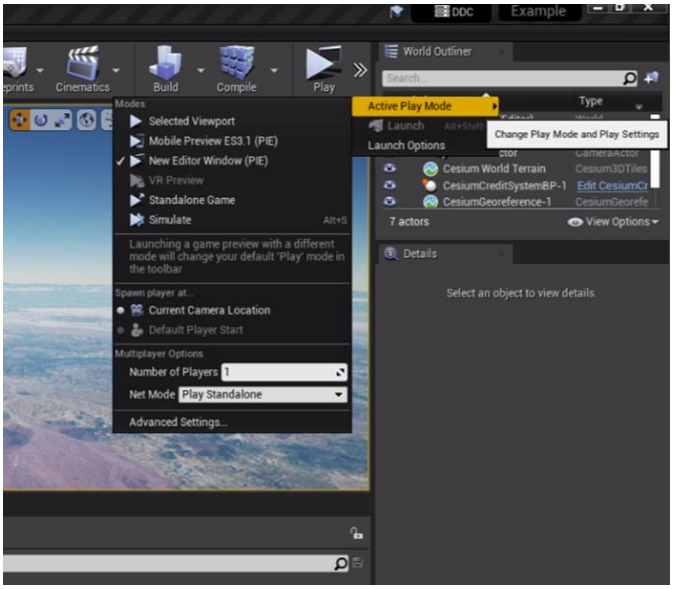


Once the project opens it will look like this:

Note: The 'Compiling Shaders' message in the bottom right is letting the user know that the editor is loading in the scenery and lighting. For best performance we recommend waiting for all the shaders to compile before playing the project. This can take several minutes.

Now we'll play our project. Our premade projects are preloaded with different drop patterns to follow. If you would like to change the drop file referend XYZ in the manual. In the top bar there are several icons. Locate the **Play** button. If you cannot see it, you may need to increase the size of the window.

There are a variety of ways to play and display the simulation which can be found under **Active Play Mode.** We recommend playing in the **New Editor Window (PIE)**, click on it to start playing the project.

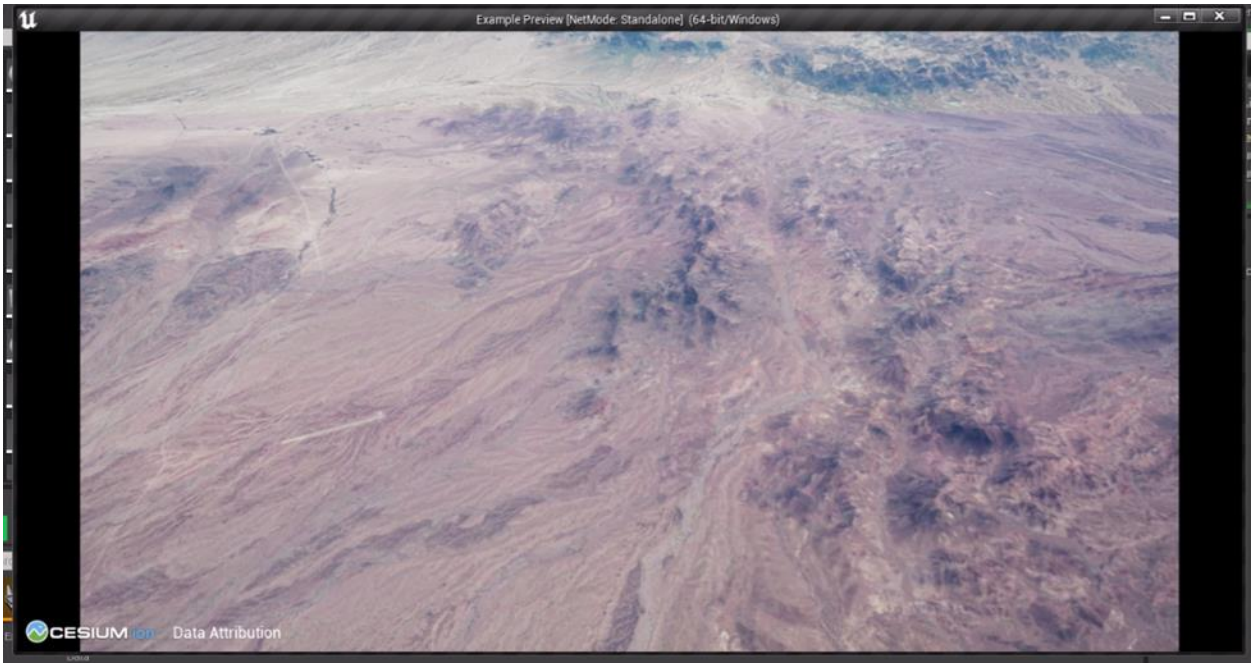


A new window will appear running the project. You can click anywhere in the window to begin mouse control. You can move your mouse around to look around in the simulated world. The view is a general viewport view.

Flying and Collecting Data

There are several built in key triggers to control the drone in the simulation:

L - Mounts the camera to the drone and changes the view to the drone perspective, once in this perspective you can no longer control the view with your mouse. The view will face downwards as seen below.



M - Begins the flight

C - Manually capture a screenshot and location, can be clicked multiple times

F - Begins an autonomous collect of screenshots and locations that will run during the whole flight

To start the flight first click L to change perspective then M to begin moving. Then you can click C or F to collect data depending on how you would like to collect it. Drops typically last about 20 minutes.

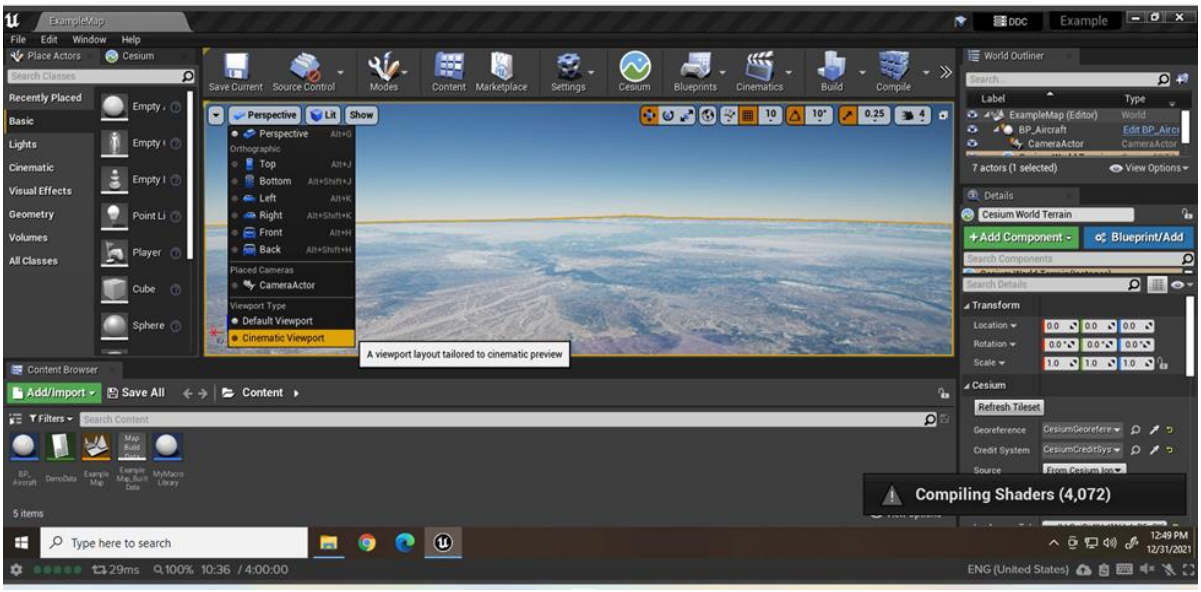
Accessing Data

Once the drop has completed you can close the window and all the data collected will be saved. To access the screenshots collected navigate back to Project Folder -> Saved -> Screenshots -> Windows. To access the log files containing the locations navigate to Project Folder -> Saved -> Logs.

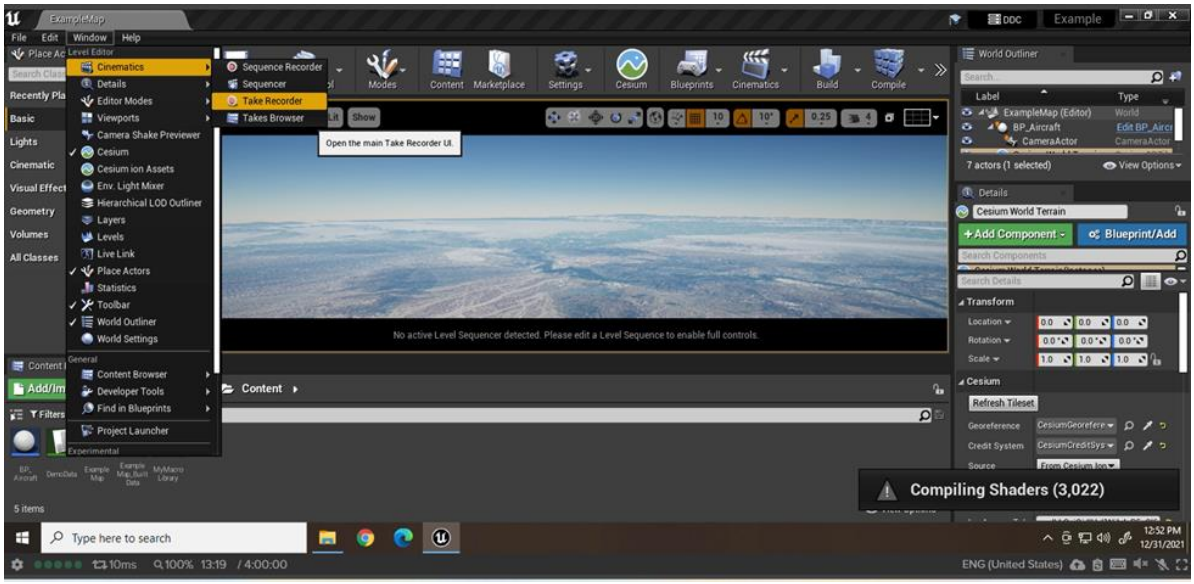
Recording a Drop Video

To record a video of a drop, the first step is to run the drop once and record how long the simulation takes. This is a key step, since in the recording step you will not be able to observe the movement while you record. In this example, I will be working with a project where I know I want to record thirty seconds of flight.

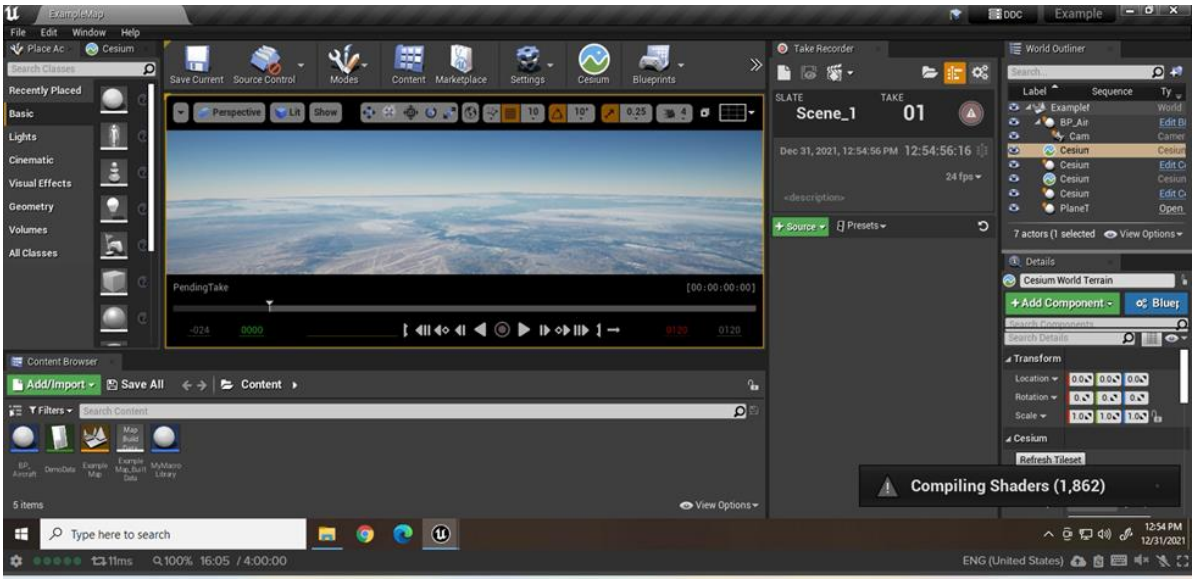
First, in the top left corner of the viewport select Perspective → Cinematic Viewport.



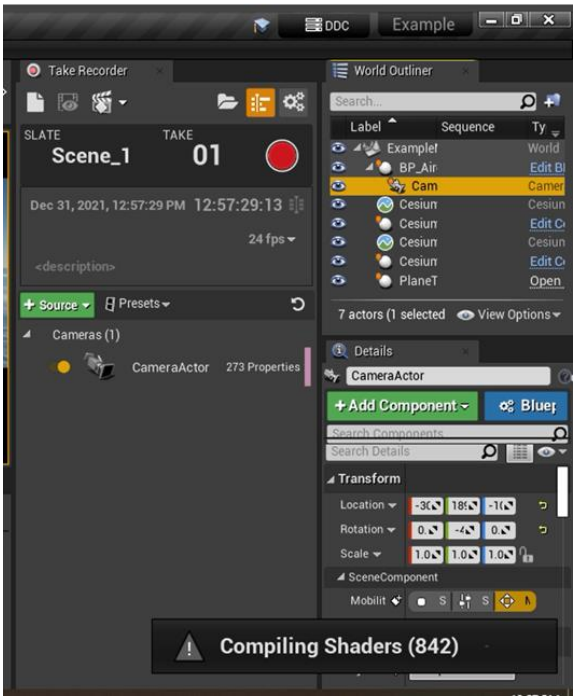
Next, bring up the recording panel by going to the top left corner of the Unreal Editor and selecting Window → Cinematics → Take Recorder.



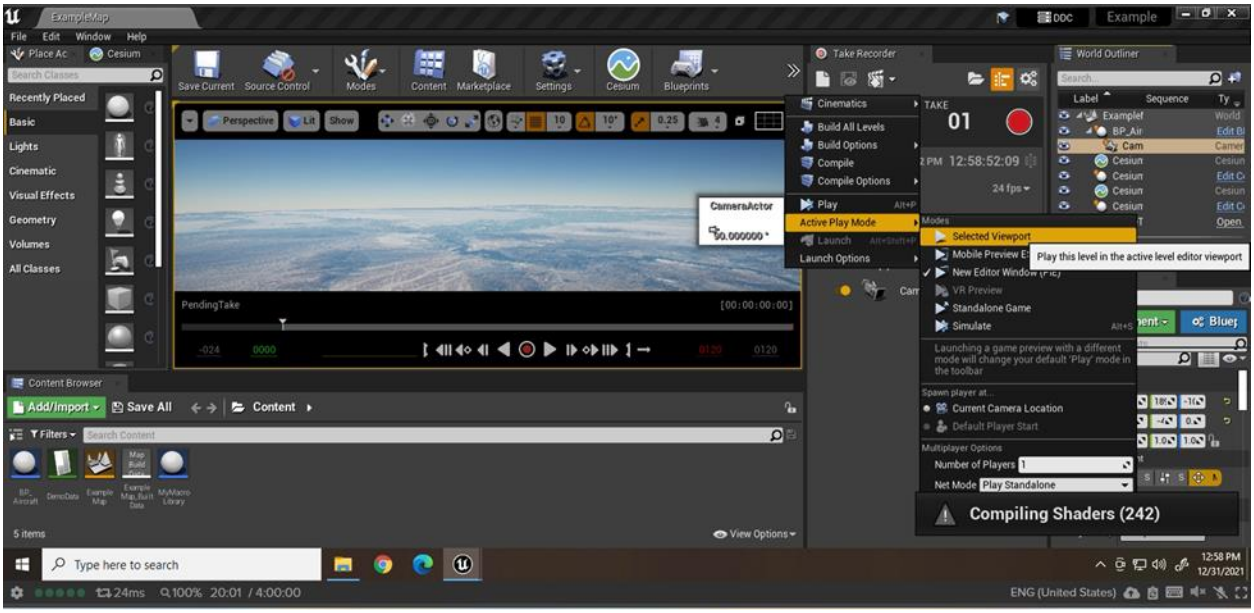
This will open a popup window, which you can minimize, and underneath there will be panel at the left, titled take recorder.



Click and drag your CameraActor from the World Outliner into the bottom section of the take recorder panel. This will tell the program which perspective to record from.

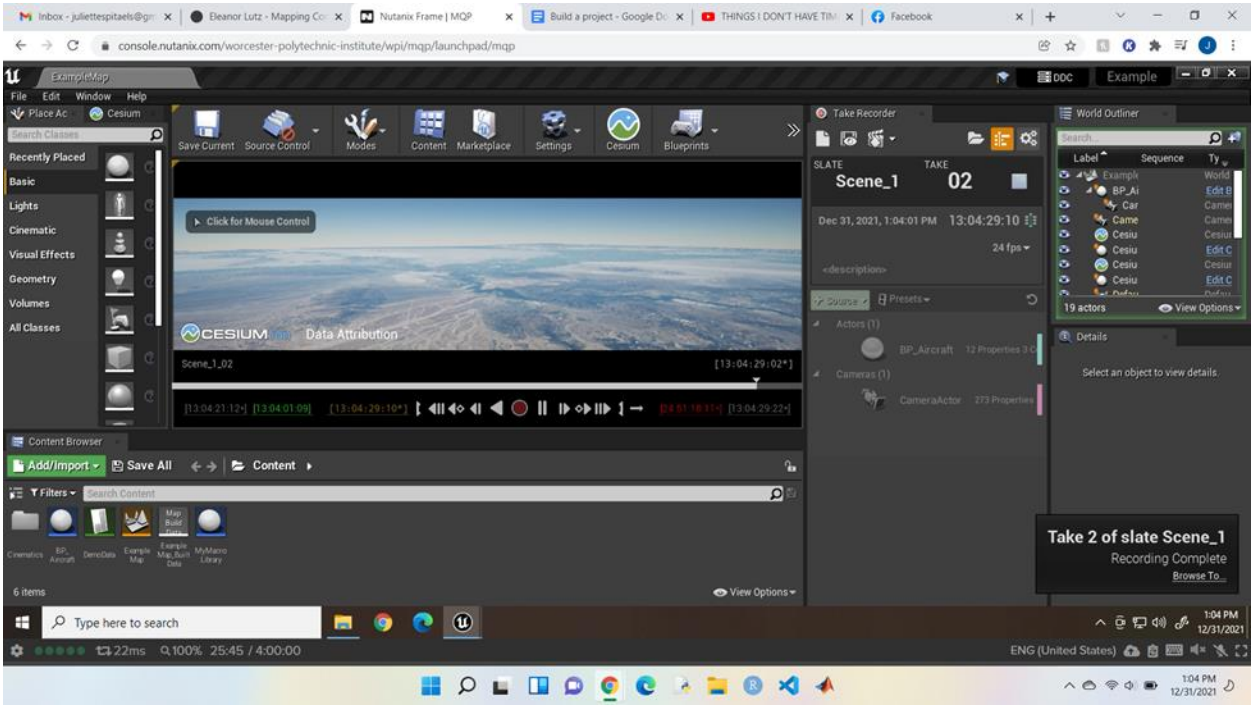


Now play the project, making sure you're using "selected viewport" Active Play Mode.



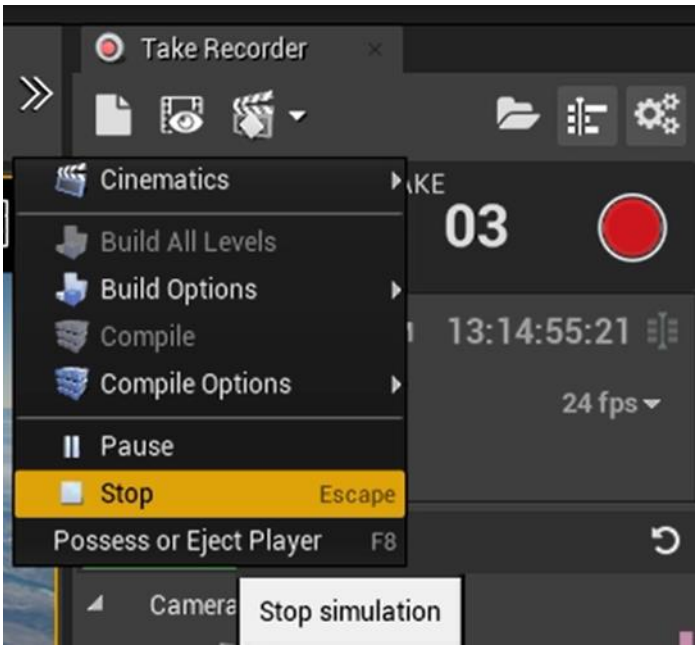
Now you can press the large red record button in the Take Recorder panel. This bring back up the popup, which again you can minimize, and below will be a three second countdown.

To start the movement of the parachute, click into the viewport and press the M trigger key. **Note: Make sure you do not press L to mount the camera, or else it will not record properly.**



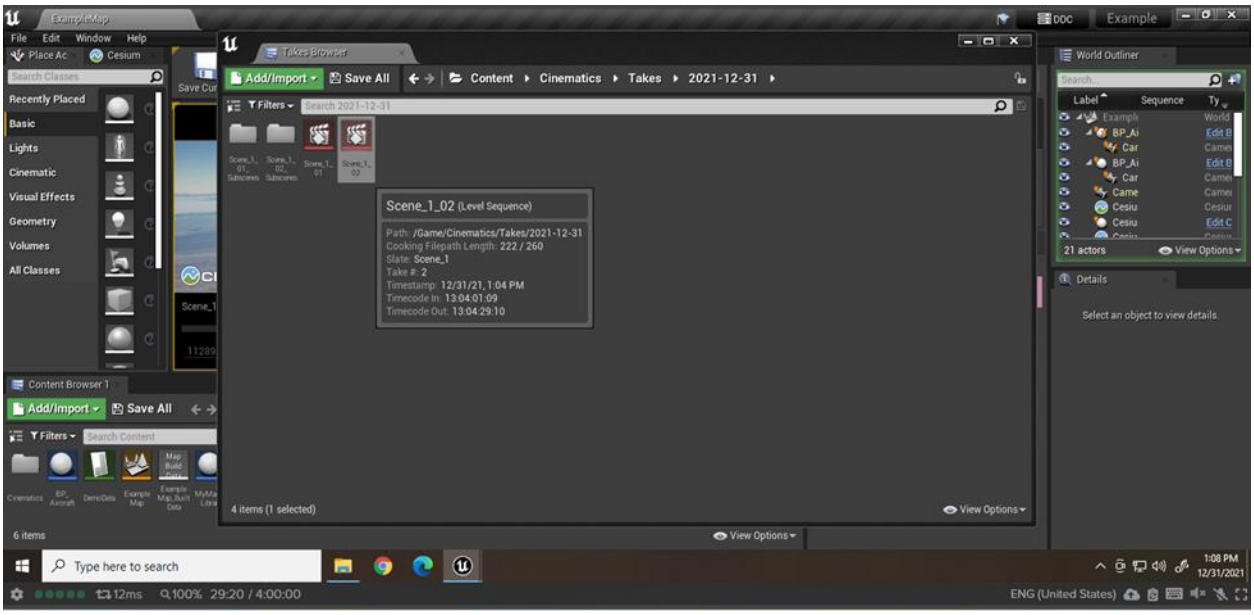
Record for your desired time, then press the gray stop button in the Take Recorder panel.

Again, this will cause the popup window to open, which you can again minimize, and instead stop the game. (This stop button is found in the tool bar at the top of the editor.)

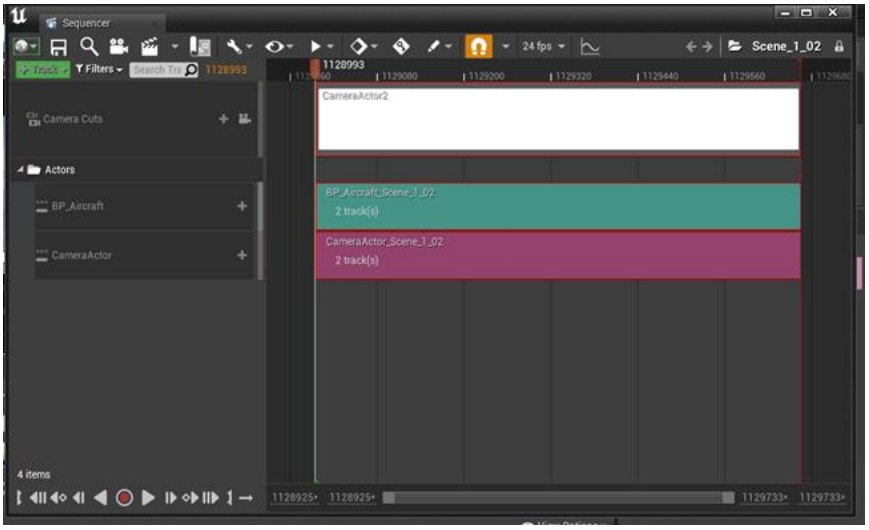


Now click the takes browser, represented by the open folder button.

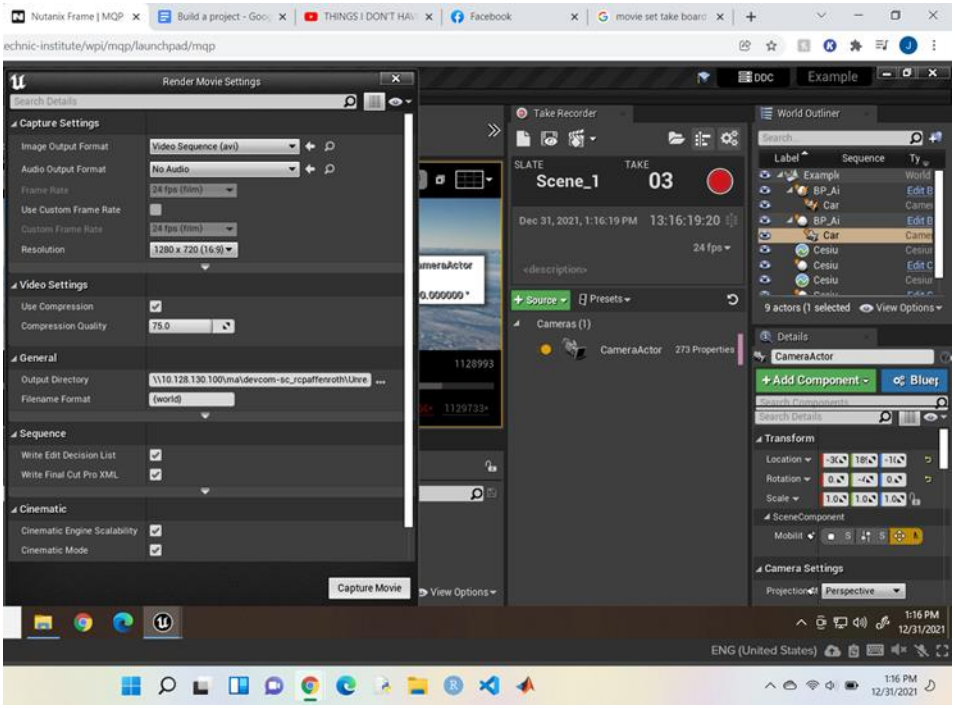
Select the take you recorded, from the popup. Note: I ran my recording twice, which is why I have two different takes to choose from. The takes are numbered with the most recent recording having the highest number.



Selecting your take will open another popup window, titled Sequencer. In this window you will select the render to video button on the top left, which is the button that looks like the clapperboard.



This will open yet another window, where you can review the settings used for the recording. **If your shaders are not finished compiling for the project, WAIT. Continuing past this step without compiled shaders often leads to the engine crashing and not saving your takes.** If your shaders are compiled, at the bottom left of this window click the gray Capture Movie button. If you're prompted to save your project at this step, do so.



Completing the previous step will cause a preview window to open somewhere on your screen.

Once your preview disappears, it means your video has finished rendering. To view it, go into <your project folder → Saved → VideoCaptures. There you will see your video saved according to your level name. If you render multiple videos, numbers will append to the end to differentiate the files.

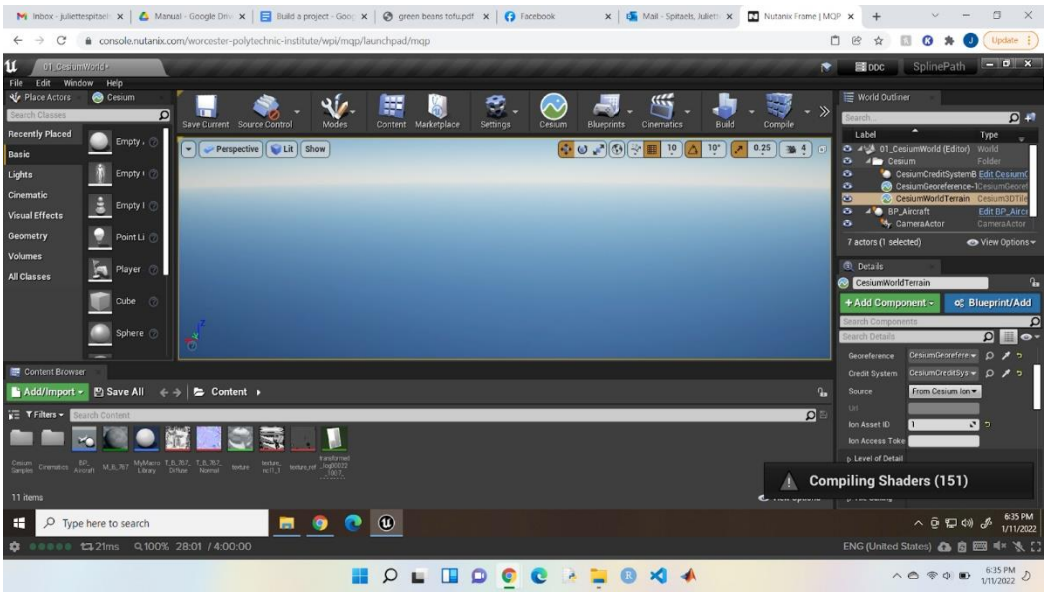
To check your video saved properly, click it to view.

It's a good idea to **Save** your project after this step. Now you can record more videos or close the Unreal Editor if you're finished.

Troubleshooting

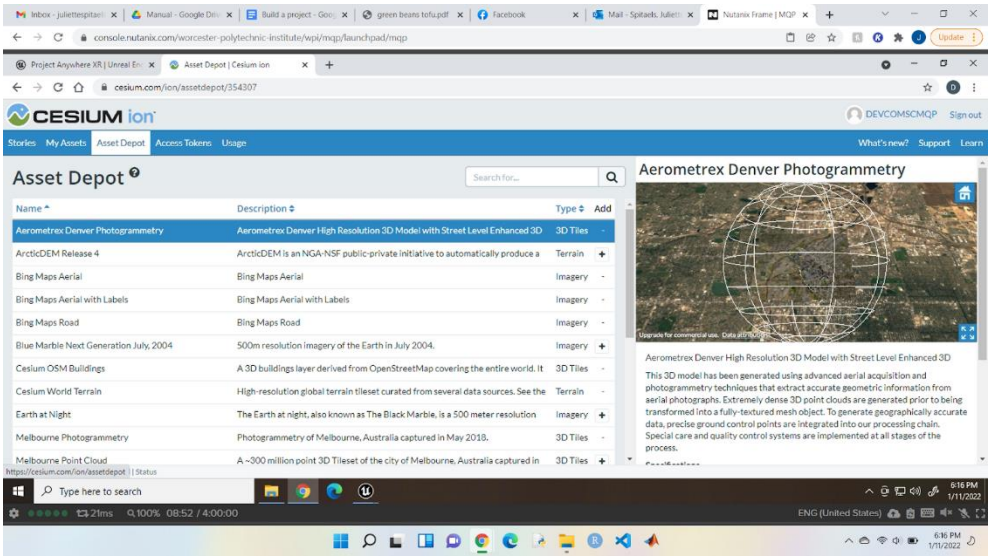
Cesium World Disappears

At times you may encounter an issue with tokens, which are unique identifiers for access to streaming Cesium imaging. If your globe ever appears as a blank blue ball, these steps can be followed to fix the issue in affected projects.



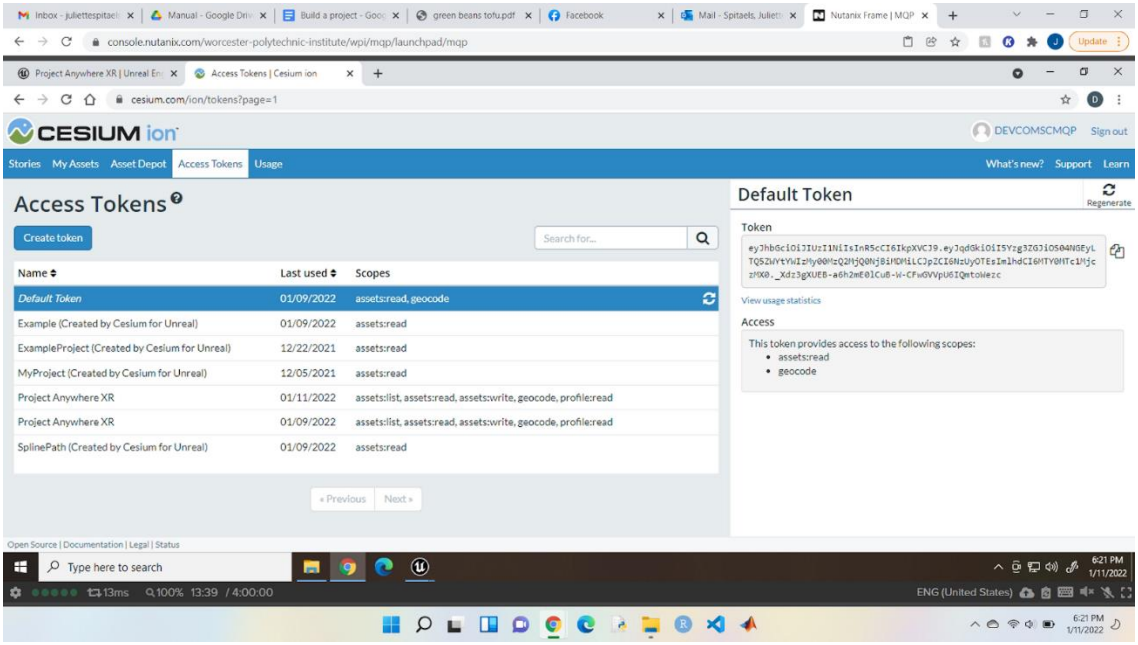
For further detail, see tutorial: docs.unrealengine.com/4.27/en-US/Resources/Showcases/ProjectAnywhereXR/#projectsetup

Log in to your Cesium Ion account and go to the Asset Depot tab, in the top left of the window.

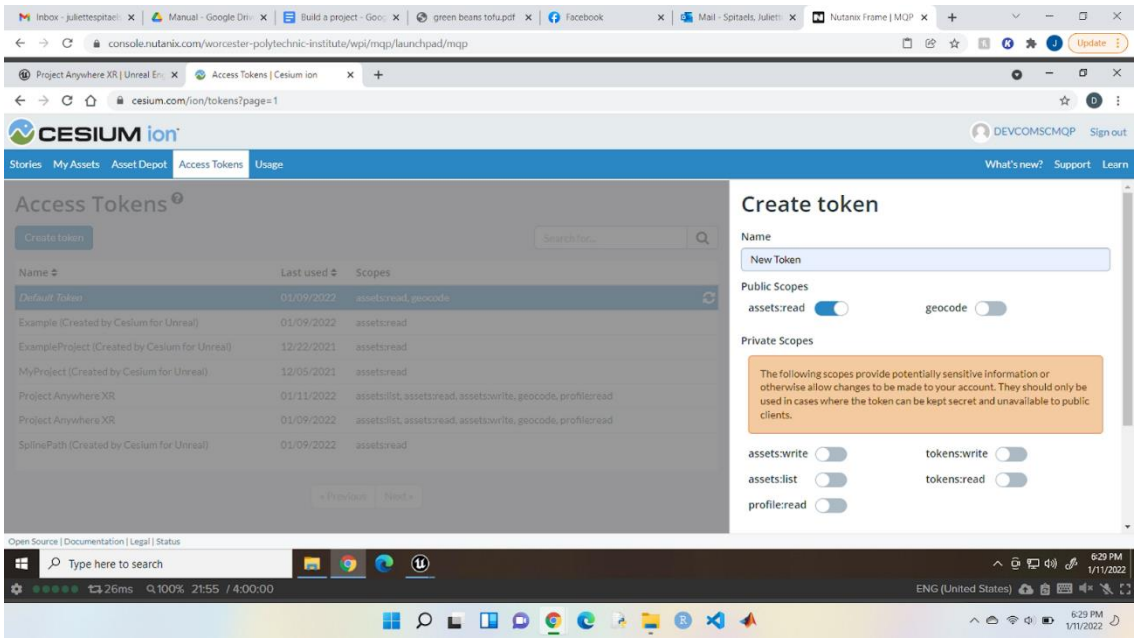


Make sure you have the following 3D Tiles assets Added, by selecting the plus button on the left:
Bing Maps Aerial; Cesium world terrain

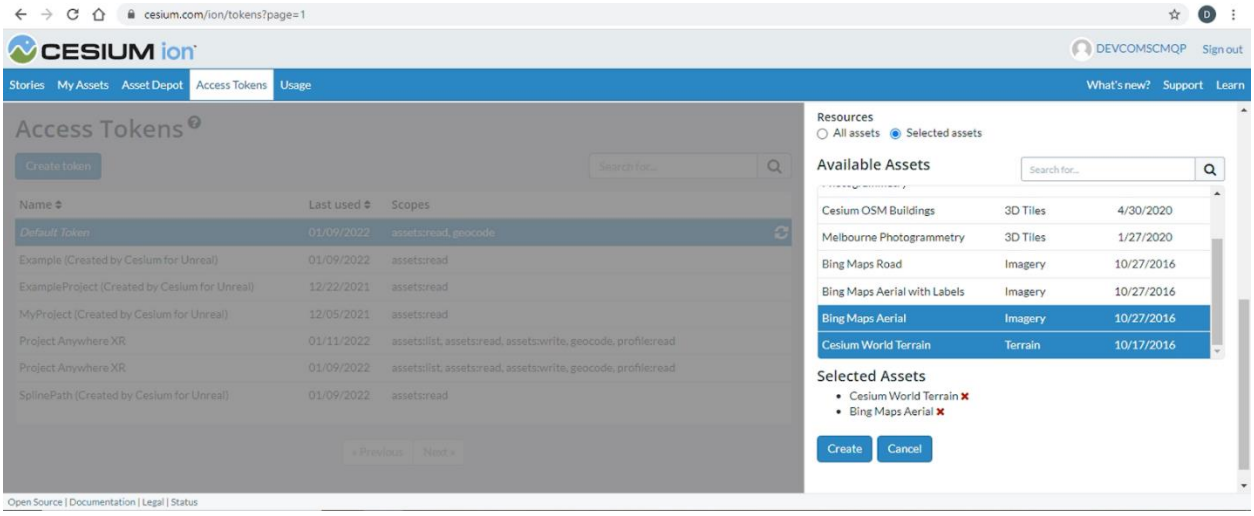
Go to the Access Tokens tab, also located in the top left of the window.



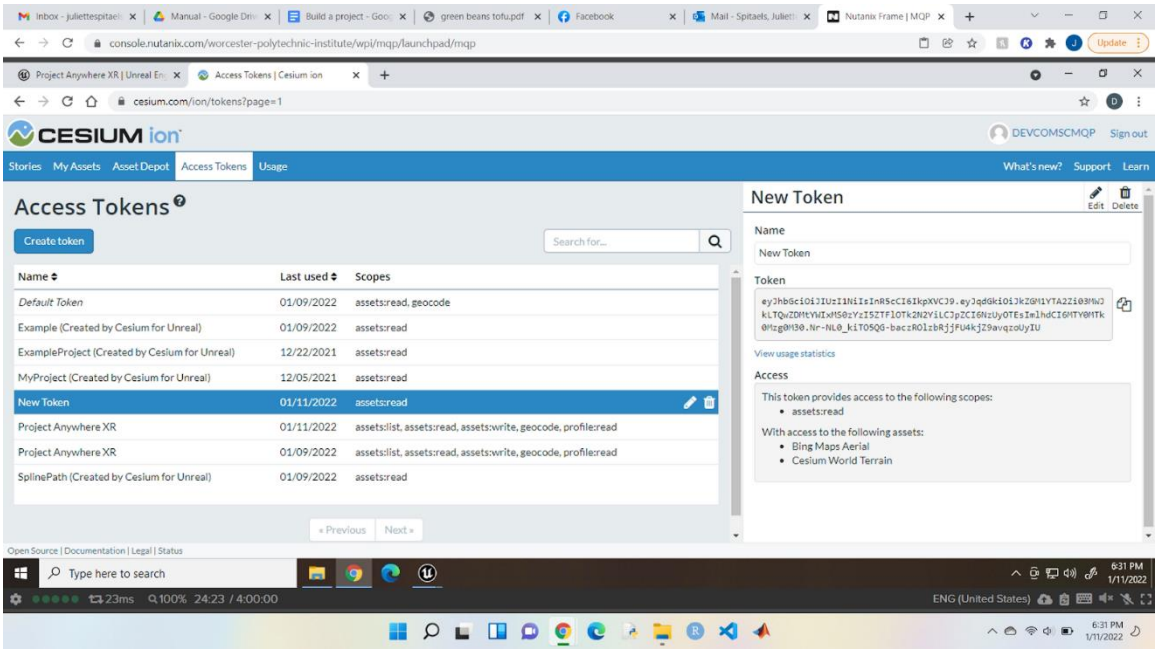
Click the blue **Create Token** to open the Create token panel. In the Create token panel, first name the Token. In this case it's named 'New Token'.



Under Resources, select the radio button for Selected assets to show the list of Available Assets. In the Available Assets list, select the following assets: Bing Maps Aerial; Cesium world terrain

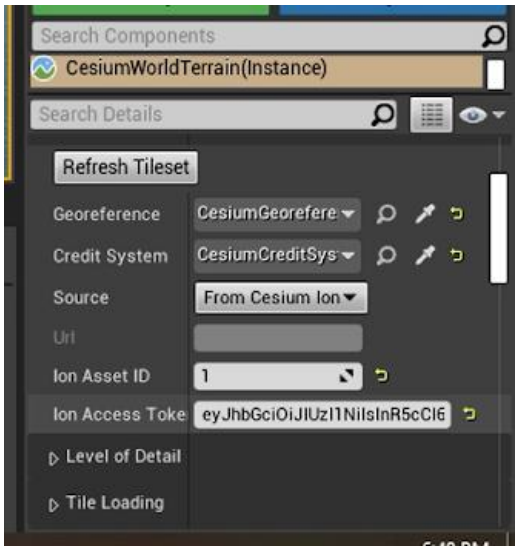


Click the blue **Create** button to create the token. Now select your new token from the list of tokens, it should look like:

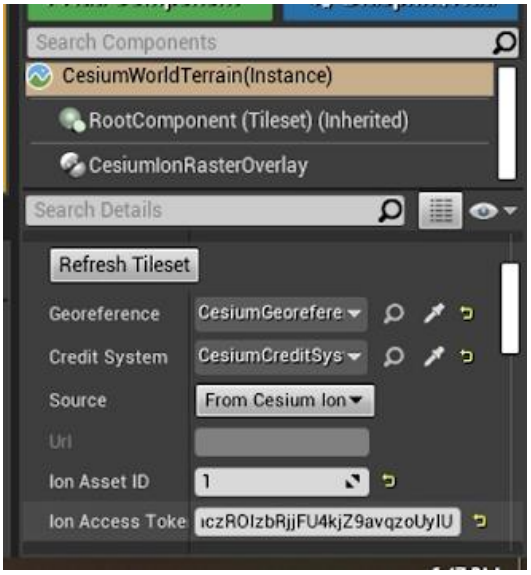


Copy the text under Token by clicking the copy icon at the left of the code. Now open your desired project. In the Unreal Editor's World Outliner, select the following Cesium World Terrain Actor.

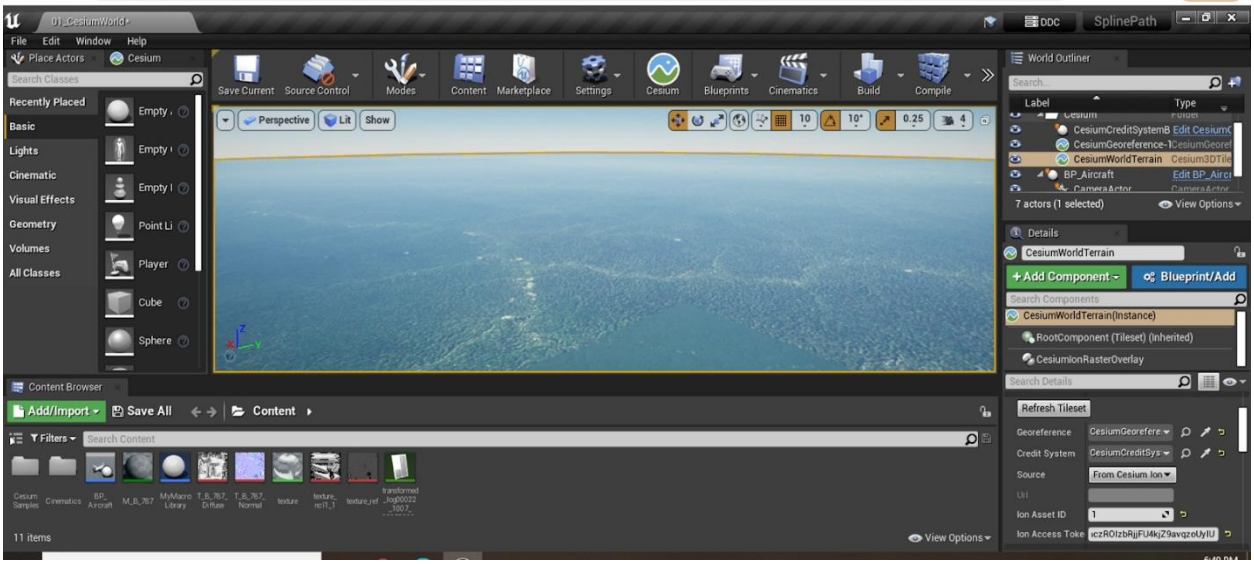
In the Details panel find the Ion Access Token property. Delete the existing token and replace it with the new token you generated.



Next, click and drag to expand the components panel, which is just above the details search bar. Select the CesiumRasterOverlay from the list. In the panel, find the Ion Access Token property. Again, delete the existing token and replace it with the new token you generated.



Now your globe should show land masses and other features again.



To finish, press the Save button in the Unreal Engine toolbar.

Frequently Asked Questions

-