Unity and Vuforia – Easily Developing an AR application

Jean-Marie Normand – jean-marie.normand@ec-nantes.fr
Ecole Centrale de Nantes

Introduction

The goal of this practical is to give you a very brief introduction to Unity and the Vuforia AR framework. At the end of this session, you will know how to setup Unity and Vuforia and how to create **a very simple AR application using images as targets**.

The final result will be an AR application that can recognize an image and to display and animate a 3D object related to this image. Thanks to Unity, the application we will build will run on a PC but it could also run on a mobile device without needing a lot of modifications. WE chose this configuration (Unity + Vuforia) since both are intensively used to develop real world AR applications.

This practical will present two slightly different cases to display a 3D object:

- Using a Vuforia predefined image;
- Using our own custom image.

The second use case will require the creation of an online Vuforia developer account, more details are given in the following.

Please note that we do not assume any priori knowledge of Unity not Vuforia but do not aim at turning you into Unity experts. We will detail and guide you through the struct minimum to build our AR application which will be very simple. The final goal is not to obtain a final AR application but rather to give you the basis to be able to build an AR application. Turning it into a “real” AR application will be left as future work for you.

Outline

- A very brief introduction to Unity
- A very brief introduction to Vuforia
- Installing and integrating Vuforia within Unity
- Unity's GameObjects and Vuforia
- A first AR application using Vuforia's predefined images
- Using your own images as markers
- Conclusion

Unity

Unity is one of the two (the other being Unreal Engine) most popular and used 3D game engine currently available. It presents many advantages among which:

- Multiplatform: Unity allows you to develop for PC, Linux, Mac or even mobile platforms (Android and iOS)
- State of the art rendering engine
- Support for physics engine
- Easy integration of 3D models
- Powerful scripting mechanisms
- Simple and user friendly UI
- Active community with lads of tutorials online
- Allows for easy and rapid prototyping of games, 3D, VR and AR applications

More on the web: www.unity3D.com

Vuforia

Vuforia is a framework that offers the possibility to easily develop AR applications. It is very easy to use and allow for the rapid development of simple AR applications. Another advantage of using Vuforia is that is allows for the augmentation of different types of "**markers**": images (as will be done in this practical), but also 3D models. Vuforia also supports both ARCore (AR library from Google designed for Android phones) and ARKit (Apple's AR library) frameworks whenever available on the targeted mobile device. When the target platform does not support either ARCore not ARKit, then Vuforia offers for its own SLAM/3D model tracking solutions.

In the following, we will refer to "markers" as AR targets that need to be recognized and that are "augmented" that is to say used to display 3D objects in the scene.

Installing and integrating Vuforia within Unity

Since 2017, Vuforia is directly integrated within Unity, which further simplifies the development of AR applications within Unity.
The only thing required is to check the "Vuforia Augmented Reality Support" checkbox when installing Unity.

Unity's GameObjects and Vuforia

Before going into the configuration of Vuforia we will briefly present the different "objects" we will use in Unity. In Unity, the objects are called "GameObjects".

The different GameObjects we will use are the following:

- **ARCamera**: a Vuforia object that will manage the camera used to detect the objects (images, 3D models, etc.). It also serves as the "main" Vuforia object and thus allows us to configure the AR application.
- **Image (or ImageTarget)**: represents the image that Vuforia will try to detect and augment (i.e. display a 3D object on top). It will be configured to represent an image and the 3D model that will be displayed is defined as a "child" of this object in the Unity hierarchy.

To summarize, when the application will start, if the **ARCamera** object detects one of the **ImageTarget** objects, then the 3D model attached to the detected **ImageTarget** will be displayed automatically.

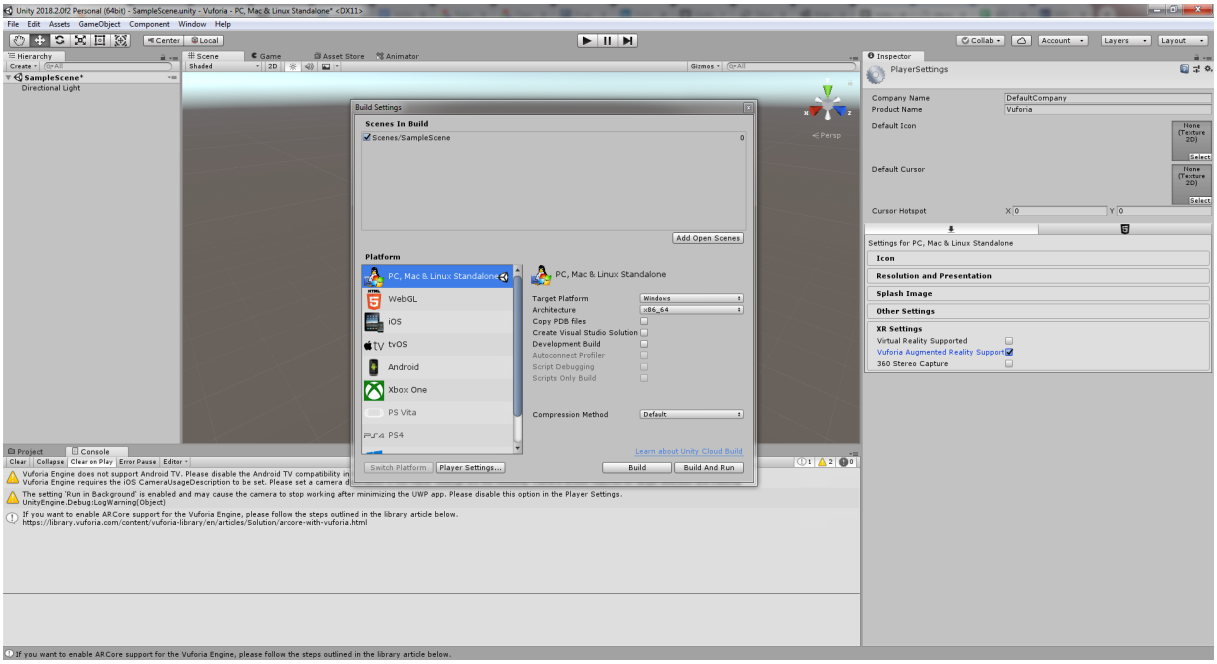
NB: Make sure that, when the **ARCamera** is selected in the Inspector does not ask you to install a new Vuforia/Unity version (you need at least v8.0.10). If Vuforia asks you to install a new version you just have to click the link in the Inspector, download and install the plugin and it should work.

A first AR application using Vuforia's predefined images

First, we need to create a new Unity project:

- Launch Unity
- Create new project
- Chose a name and location on the hard drive and chose 3D
- Select OK

We now need to configure Unity/Vuforia. To do so, we first have to make sure the target platform is correct. Go to "**Files → Build Settings**" and make sure "PC, Mac and Linux" is selected. Note that if you want to build an application for Android or iOS you should make the modifications here.
You should also make sure that "**Vuforia Augmented Reality Support**" is checked in the "XR Settings" menu on the right.



Setting up Vuforia's Objects

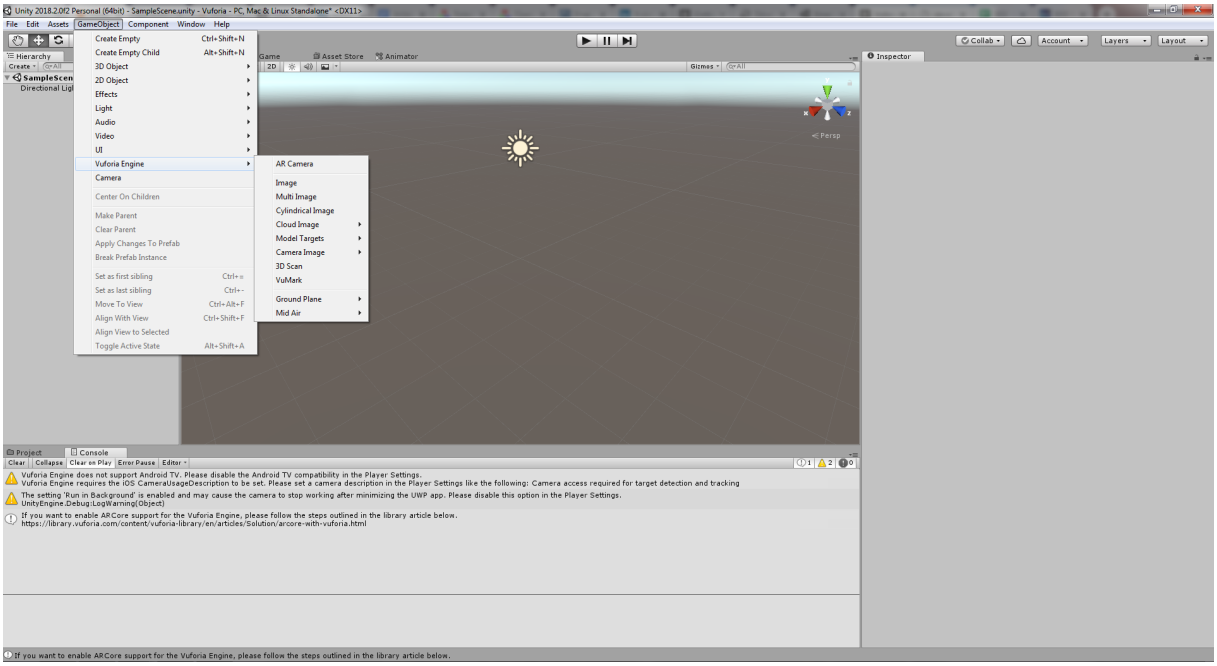
Now we need to prepare our scene for the Vuforia AR application.

Removing useless objects

Delete the "Main Camera" GameObject from Unity's hierarchy panel (in theory on the left of your screen). In a normal Unity project this object represents the main camera of the game. Here we will not be needing it since we will replace it by a special Vuforia AR camera GameObject.

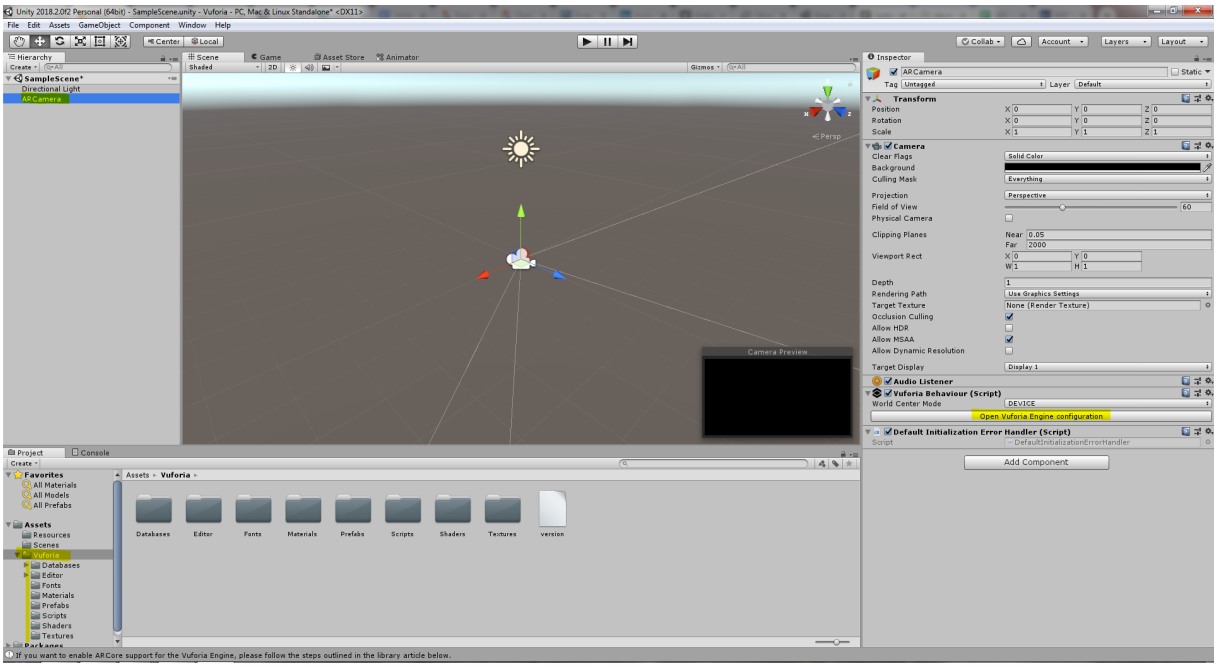
Adding Vuforia's objects

In theory, from the « **GameObject** » top menu, you should be able to see a "Vuforia" sub-menu, enter it and create a new "**ARCamera**" object.



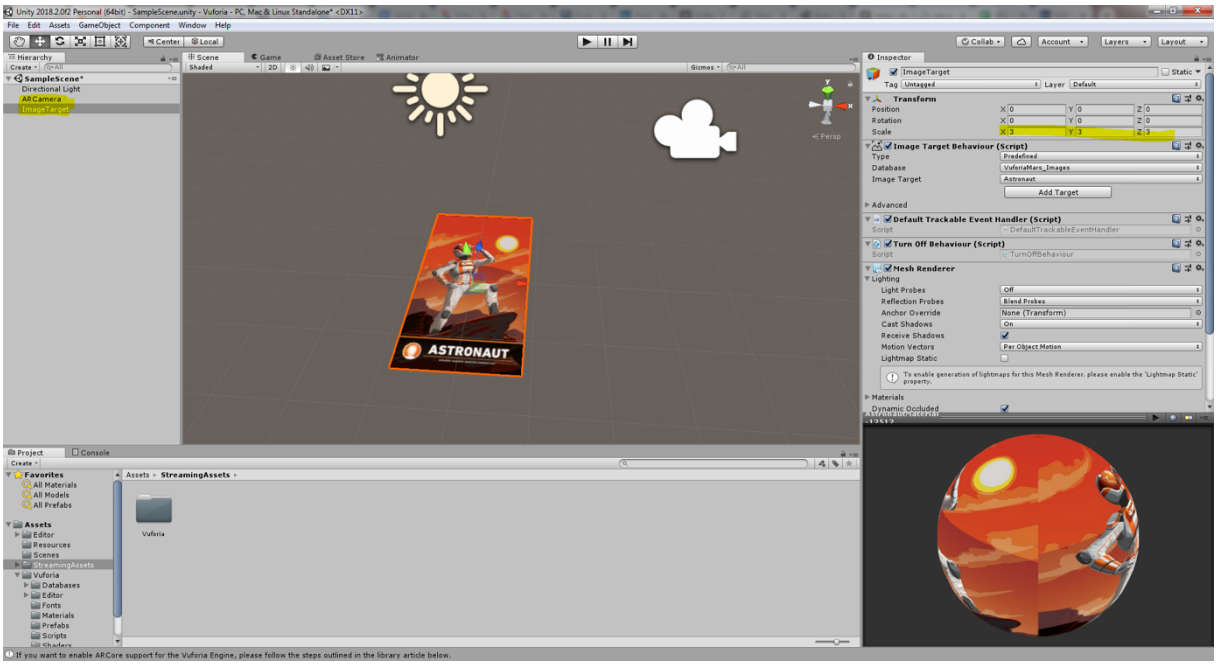
As mentioned before, the ARCamera object will store all the necessary information regarding the AR application and the cameras used to detect the AR markers.

A pop-up menu will appear telling you that some assets need to be imported. **Click Import to proceed**. Unity will download automatically for you the needed asset and put them in the project.



Now, we need to setup an **AR marker**. As mentioned; we will use images in this practical. To do so, go to the Vuforia menu and add an "Image" GameObject. A pop-up menu will appear telling you that you do not have a database and asking you whether you want to import the default database. **Click Import to proceed**. A new folder called "**StreamingAssets**" has appeared in the project view. Moreover, the "**ImageTarget**" will be added to Unity's hierarchy.

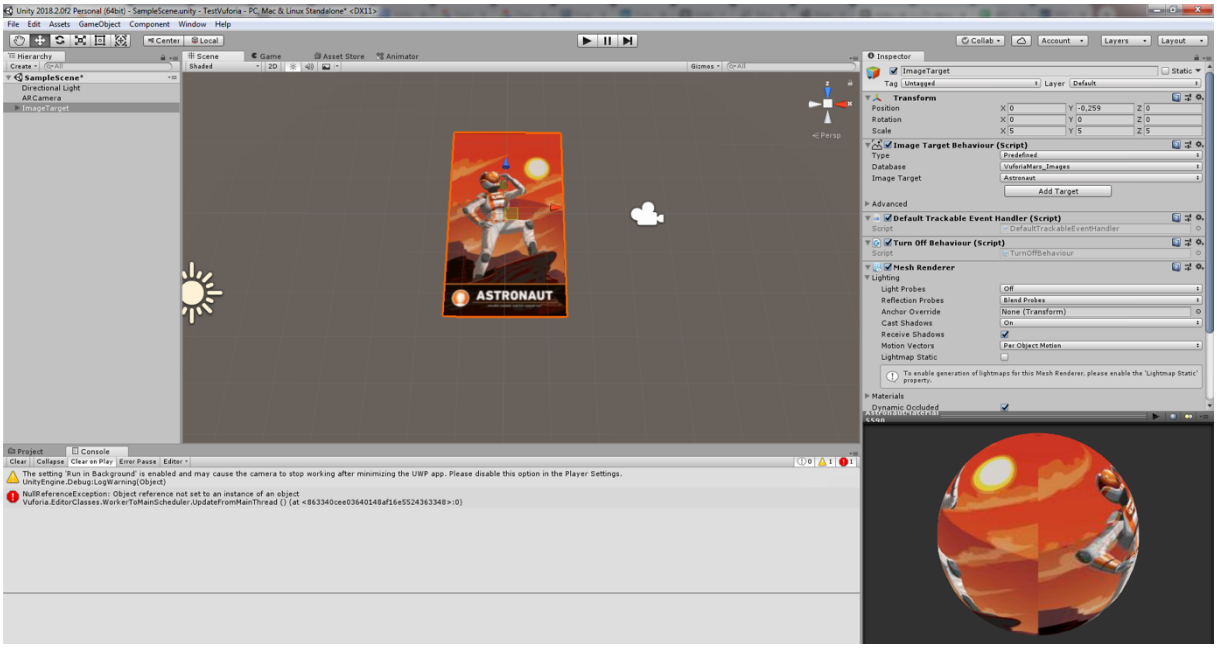
First: you need to change its size (since it is usually very very small). Modify the "Scale" field from the **ImageTarget's** "Transform" Component to **X 3 Y 3 Z 3**. The object should now be visible in the editor.



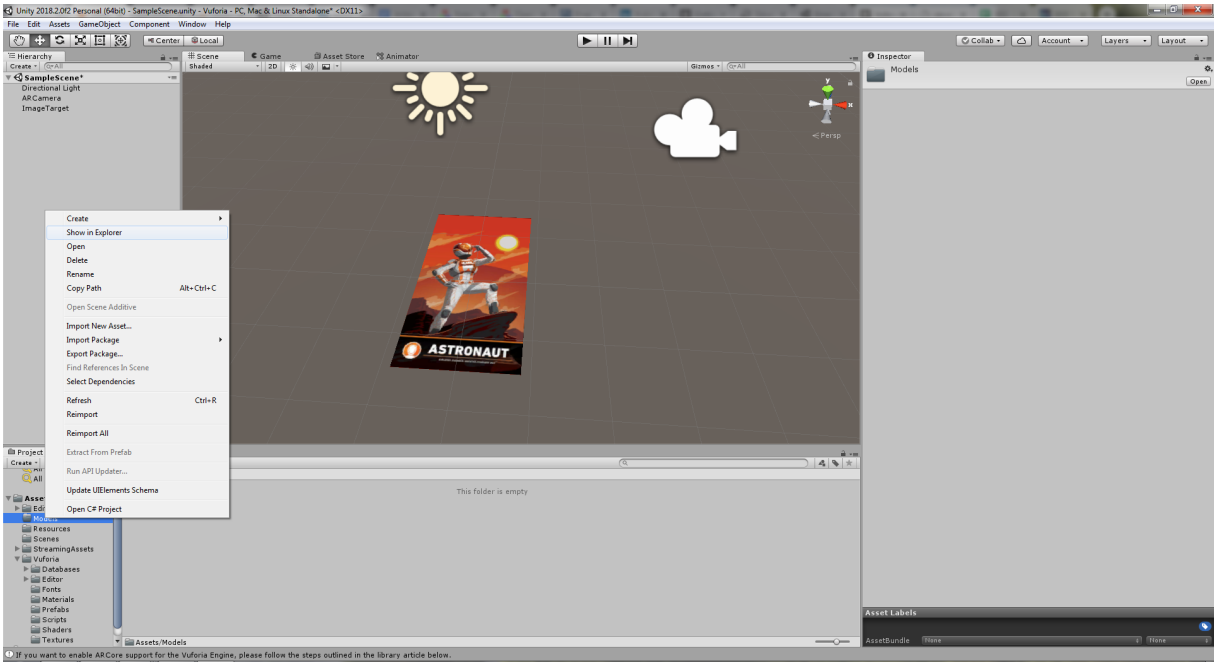
By default, Vuforia has configured the ImageTarget to use on of its "default AR marker". It is possible to modify it by using the menus of the ImageTarget object in the Inspector. For the time being, leave it to:

Type: Predefined
Database: VuforiaMars_Images
Image Target: Astronaut
There is no need to click on "Add Target"!!

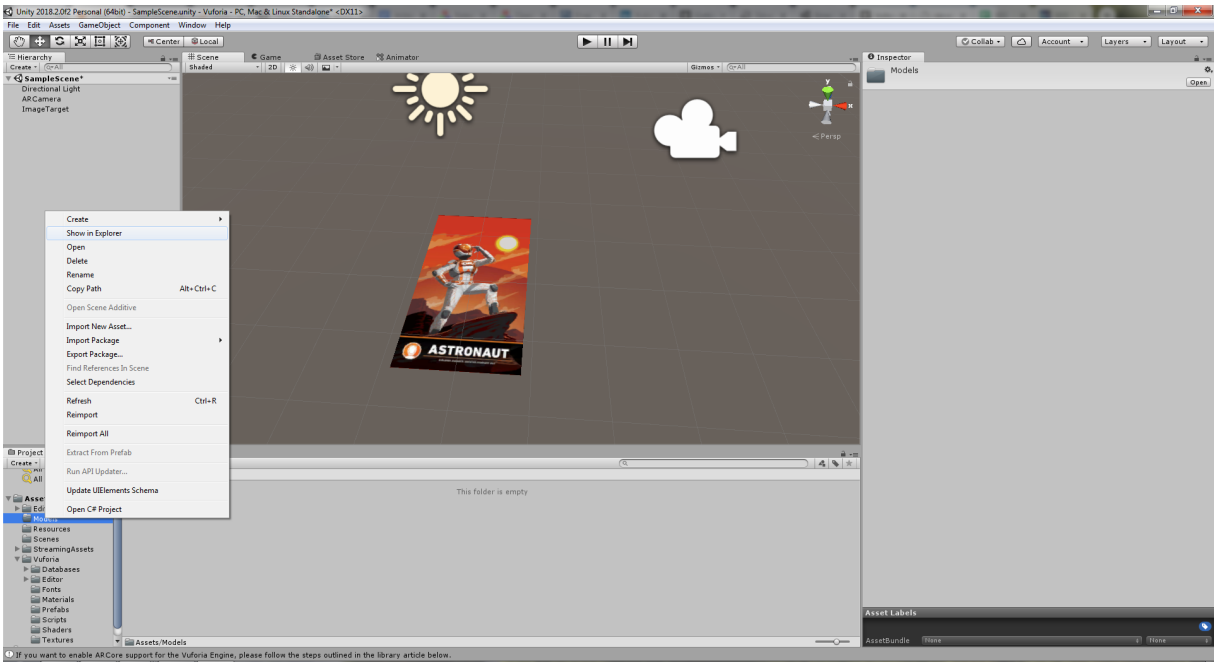
Pay attention that both "**ARCamera**" and "**ImageTarget**" objects should be at the root of the hierarchy. Note that you can add multiple **ImageTarget** objects in the scene (we will actually do that later on with our own image).

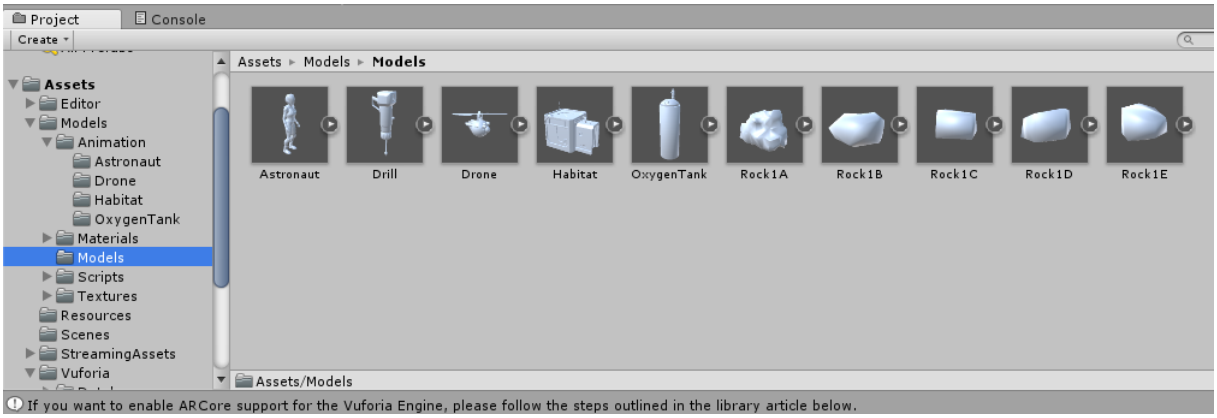


Before going on, we will now add some 3D models to our scene. We have given you a file called "**AugmentationAssets.zip**". This archive contains some 3D models that we are going to use in our application. We need to add them to our project. In the project view, right click on the "Assets" folder, and select "**Create → Folder**" to create a new folder. Call it "**Models**".



Select the newly created “Models” folder and right click on it to select “Show in Explorer”. Then go into this folder and unzip the “**AugmentationAssets.zip**” into it. When going back to Unity, the models, textures and animation should be automatically imported. You should have the following result in the project view. This contains the models we will use for the augmentation.

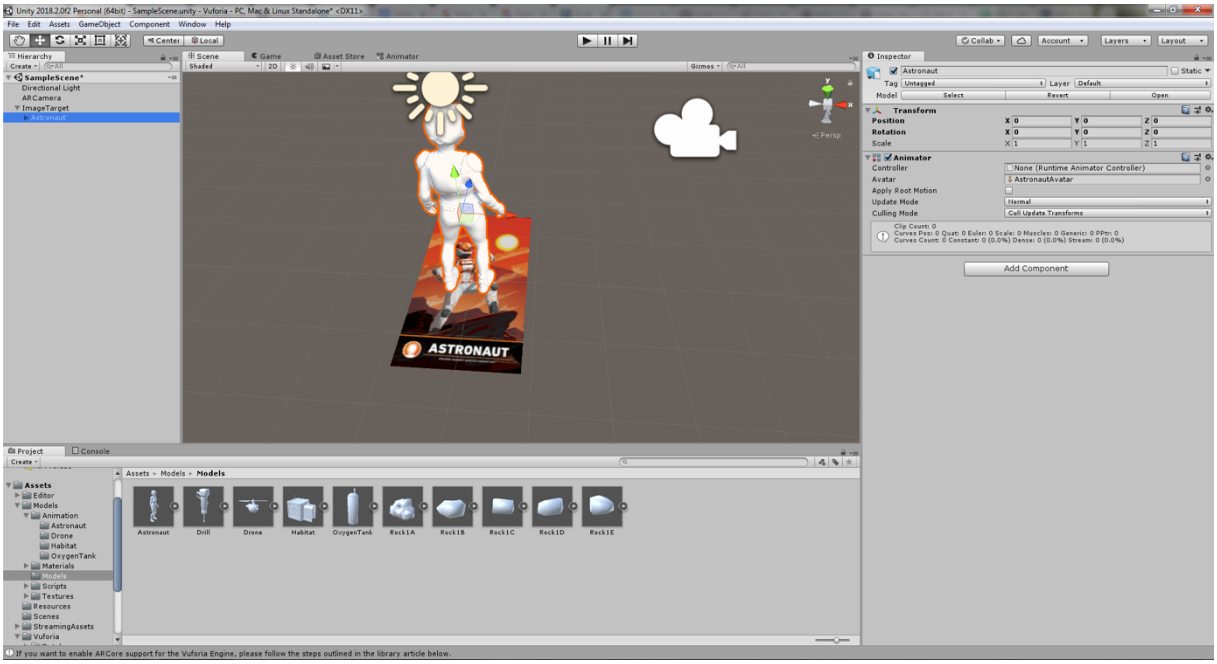




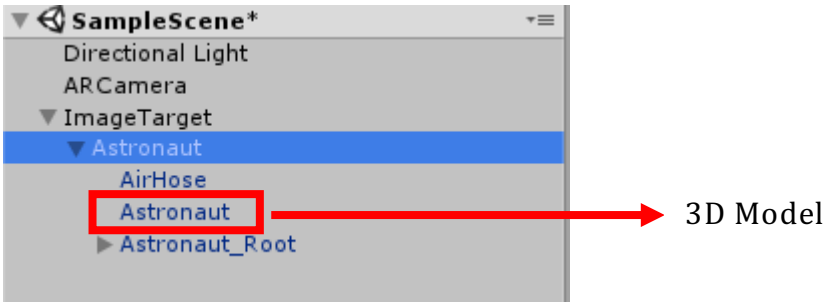
Choosing the 3D augmentation

Linking the ImageTarget and the 3D augmentation is very simple! You just need to drag and drop the 3D model to the ImageTarget. This will add the 3D model as a child of the ImageTarget's object in the hierarchy and both will thus be connected.

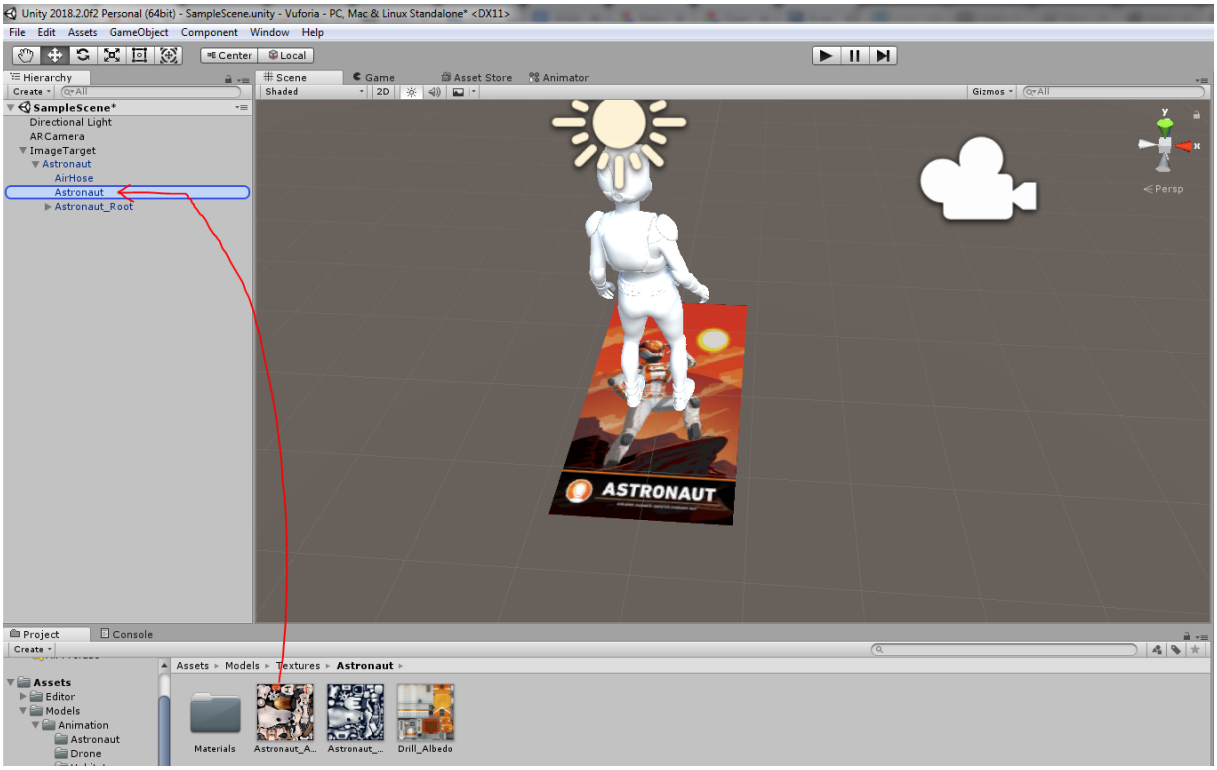
To proceed, go to the **"Models → Models"** folder and drag and drop the **Astronaut** model to the ImageTarget in the hierarchy's view. The astronaut should automatically be put under the ImageTarget. Pay attention to the fact that the Astronaut we see directly under the ImageTarget object is NOT ONLY the 3D model. It is a Unity GameObject containing: the 3D model, a "transform" (i.e. position, orientation and scale of the node) as well as an animator. This is very much Unity specific but is important for the next step.



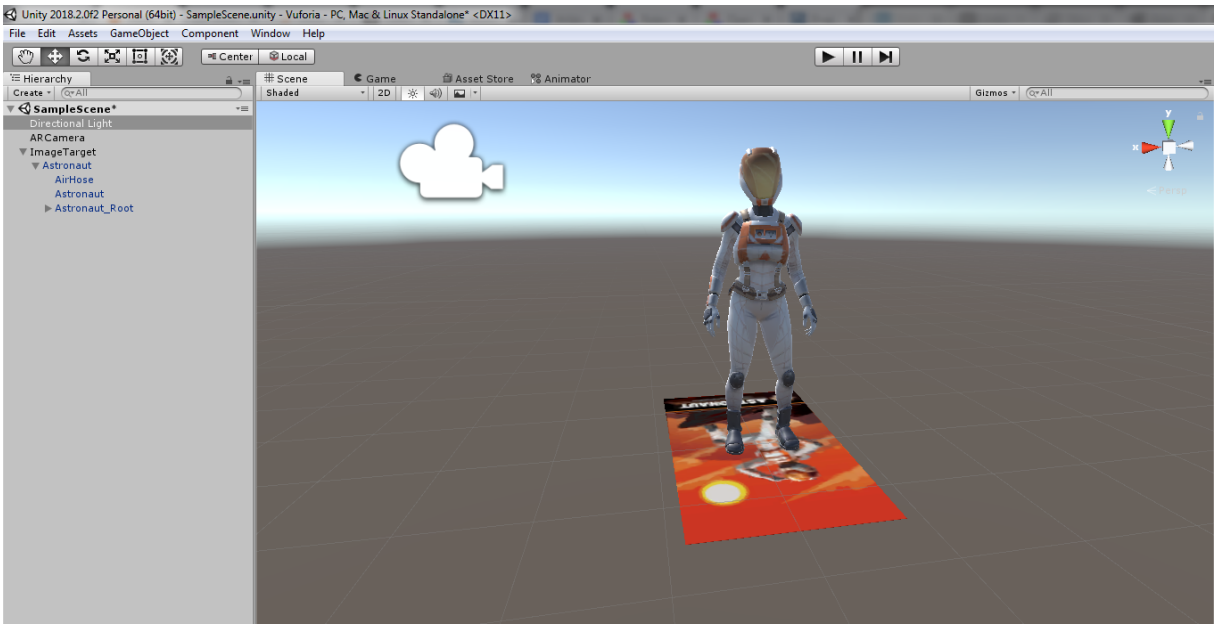
We can make it more beautiful by adding the correct texture file onto the Astronaut. To do so, we first need to select the "Astronaut 3D model". This is easily achieved by clicking on the arrow next to the Astronaut in the hierarchy. This will display the sub nodes of the "Astronaut" GameObject and you should now see 3 new sub nodes (the last one being called "Astronaut_Root" and also having sub nodes). The result should be this in the hierarchy:



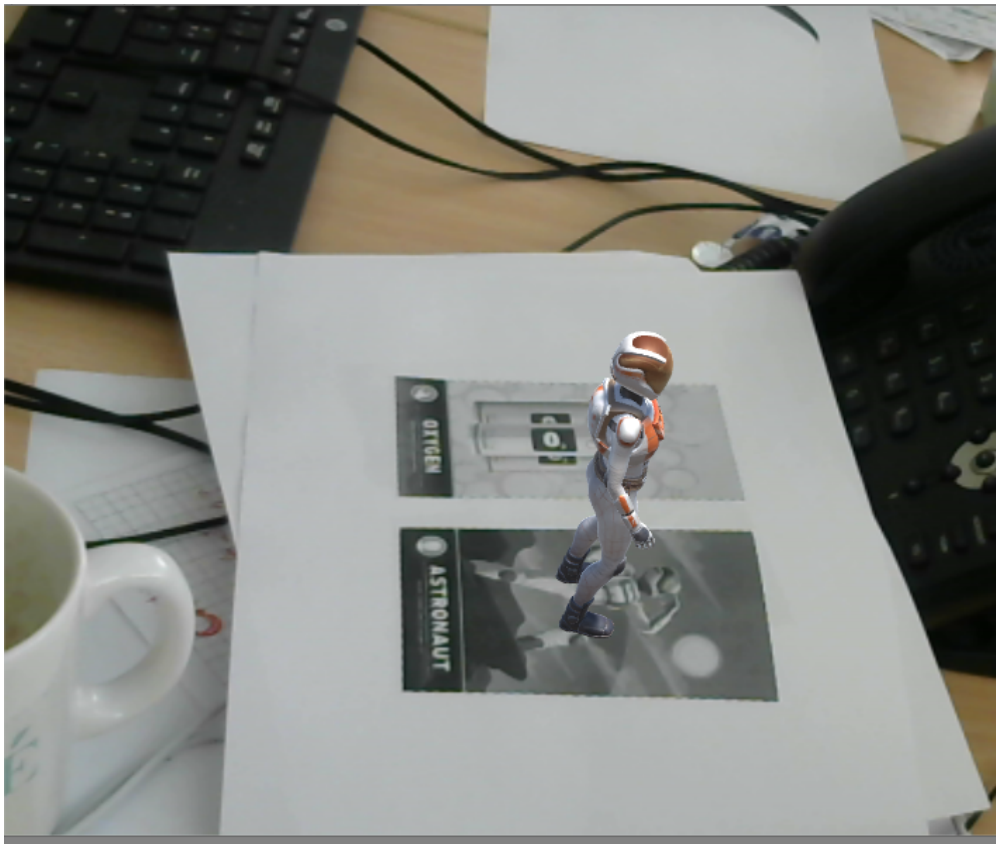
To add the texture to the Astronaut, we need to add it to the "Astronaut 3D model" which is in fact the sub node called "Astronaut". To add the texture, go to the "**Models → Textures→ Astronaut**" folder. Drag and drop the file called "Astronaut_Albedo.png" onto the Astronaut 3D sub node.



You should now have a nicer result.



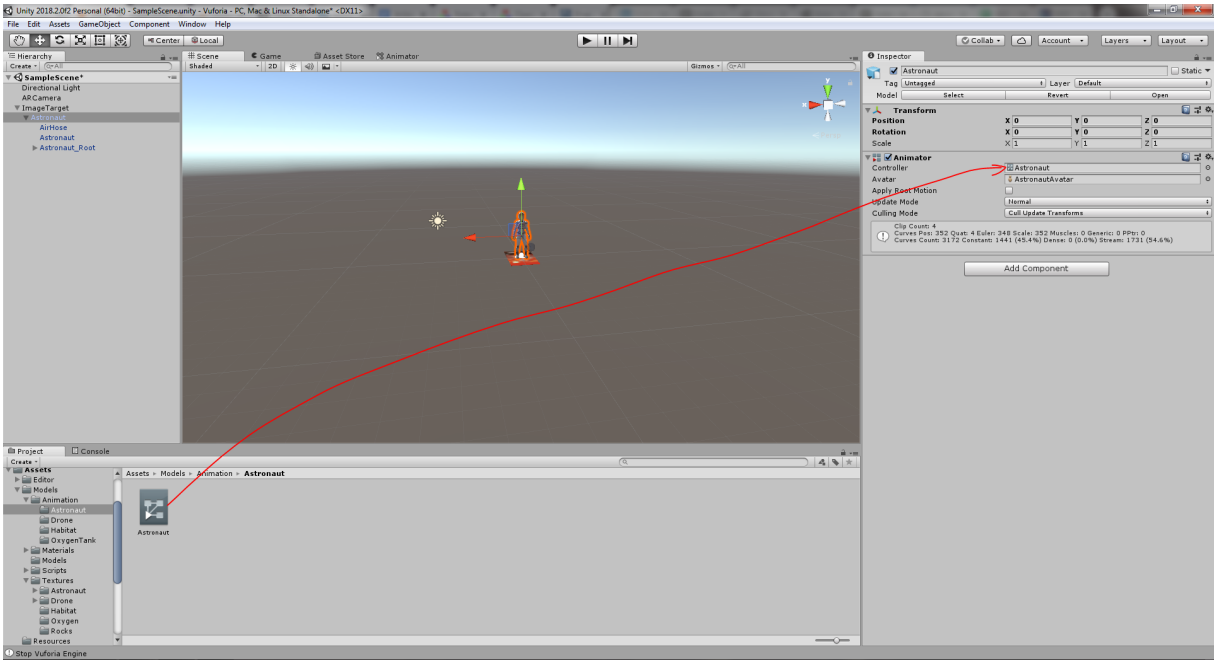
We can now try our application. Make sure you have a webcam connected to the computer. You can verify this by clicking the "**Open Vuforia Engine Configuration**" button when selecting the ARCamera. When it is the case, just press the Play Button in the Unity interface. You should obtain the following result!



Adding a nice animation

To do something a bit more interesting, we will add an animation to the Astronaut and control it when some keys will be pressed on the keyboard.

To do so, first select the "Astronaut" upper node in the hierarchy view. Then go to the "**Assets → Models → Animation → Astronaut**" folder in the project view. Finally, drag and drop the "Astronaut" animation file to the Astronaut "Animation Controller" field.



If you double click on the "Astronaut" animation controller (illustrated by the point of the red arrow in the previous figure), then Unity will open the animation editor view. You can thus visualize the different animations available for the object.

Animator View

In this view, you will be able to see the graph of animations that are currently attached to your **GameObject**. The different animations are represented by rectangles, their colors also has a meaning.

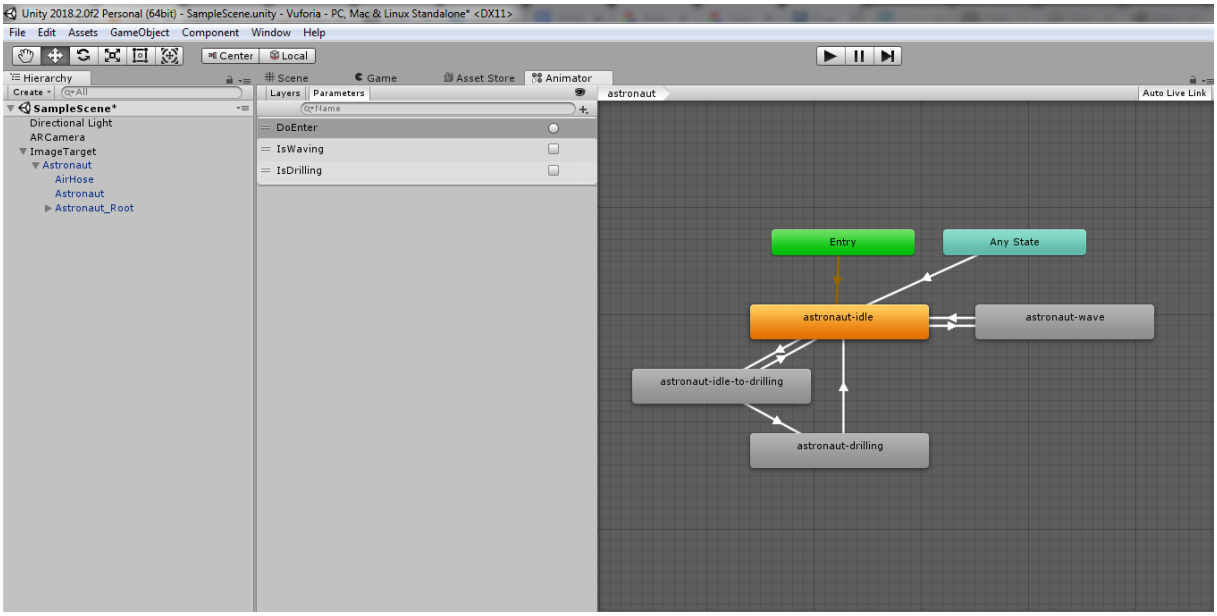
In the Figure below, you can see for example that the "default" animation for the ***Astronaut*** is called "astronaut-idle" (represented as the orange rectangle in the view) and that from this animation you can transition to two other animations:

- "astronaut-idle-to drilling"
- "astronaut-Wave"

We know this because there are some arrows coming from the “astronaut-idle” animation in the direction of “astronaut-idle-to drilling” and “astronaut-wave”.

There are some things to notice in this view:

- There are some parameters (on the left hand side), three of them are defined: “**DoEnter**”, “**IsWaiting**” and “**IsDrilling**”.
- There is a “**+**” button: it allows you to create some new parameters that you will be able to address (modify, get values, etc.) in some scripts!
- You can create some new parameters if you want to, and pay attention to the type you chose (there are a few of them, boolean, int, etc.)



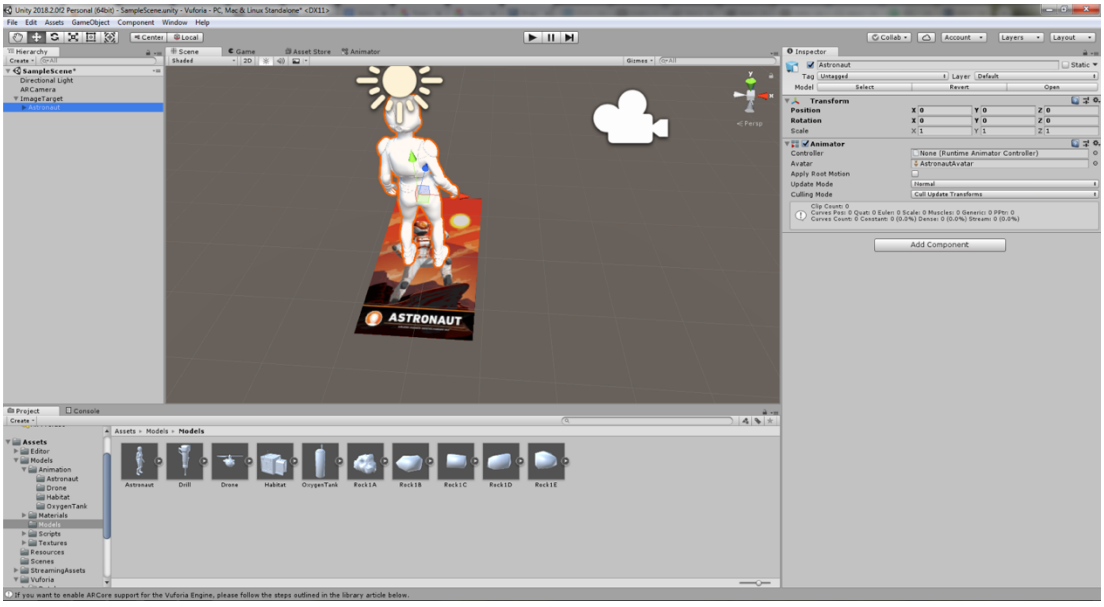
If you click on the arrows coming from and to the different animations you can see what drives the transition between one animation and the next one. Usually, some values of the parameters trigger the transitions from one animation to the next one.

Obviously, these transitions can be triggered by scripts! That’s what we’ll do now.

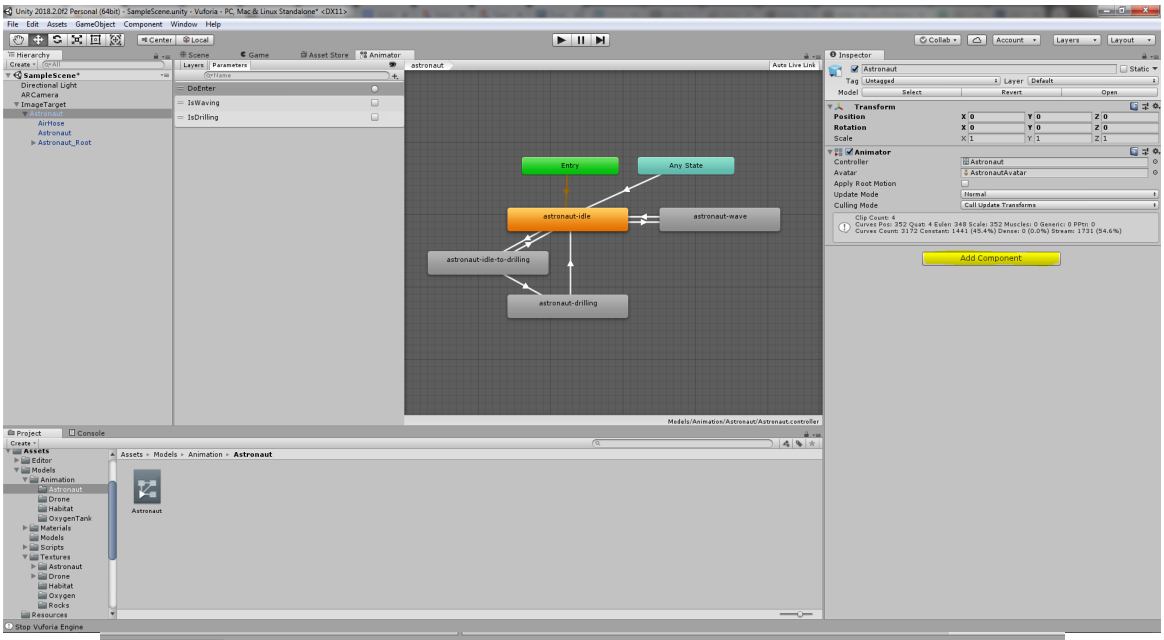
One final (obvious) comment: when you try to trigger animations by scripts, pay attention to the existing transitions!! You will not be able to go from one animation to another one if there is no transition!

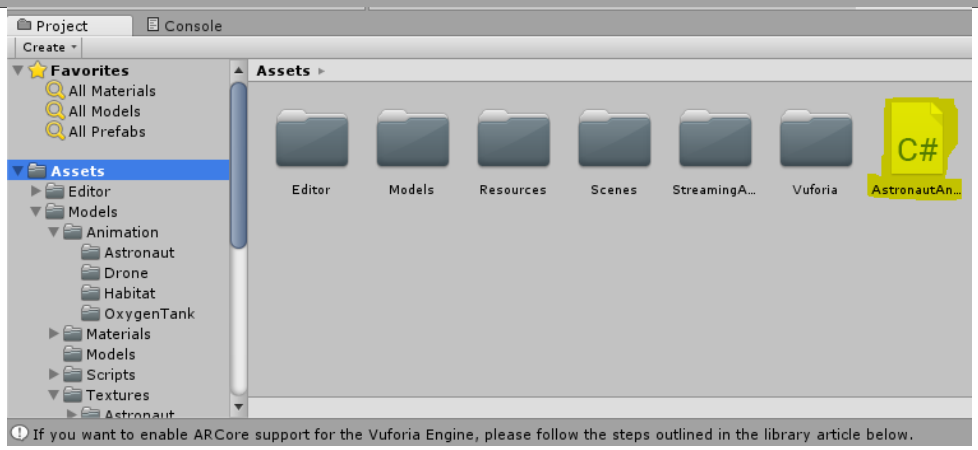
Controlling the animation

We will now add a script to control these animations!! To do so, select the Astronaut in the hierarchy view.



Click on the "**Add Component**" button and select "New Script", you can now call the script "**AstronautAnimation**" and click on "Create and Add". If you click on the "Assets" folder in the project view, you will see that a new C# script has been added.





Double click the script and Visual Studio should open automatically (or MonoBehaviour, or even another editor, it does not matter).

Please replace the script with the following one:

DroneAnimation.cs | AstronautAnimation.cs

Miscellaneous Files

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class AstronautAnimation : MonoBehaviour {

    private Animator m_Animator;
    private bool drill = false;
    private bool wave = false;
    private bool idle = true;


    // Use this for initialization
    void Start () {
        m_Animator = gameObject.GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update () {

        if (Input.GetKeyDown(KeyCode.D))
        {
            Debug.Log("Toggle Drilling");
            wave = false;
            m_Animator.SetBool("IsWaving", wave);
            drill = !drill;
            m_Animator.SetBool("IsDrilling", drill);
        }

        if (Input.GetKeyDown(KeyCode.W))
        {
            Debug.Log("Waving");
            drill = false;
            m_Animator.SetBool("IsDrilling", drill);
            wave = !wave;
            m_Animator.SetBool("IsWaving", wave);
        }

        if(Input.GetKeyDown(KeyCode.Space))
        {
            Debug.Log("Resetting");
            drill = false;
            wave = false;
            m_Animator.SetBool("IsDrilling", drill);
            m_Animator.SetBool("IsWaving", wave);
        }

    }
}
```

As you see, the idea here is to launch some of the animations (presented in the animator view) when some keys are pressed on the keyboard. More details will be given orally during the practical.
Now launch the application and try pressing 'D' or 'W' or the Space bar!! Enjoy!

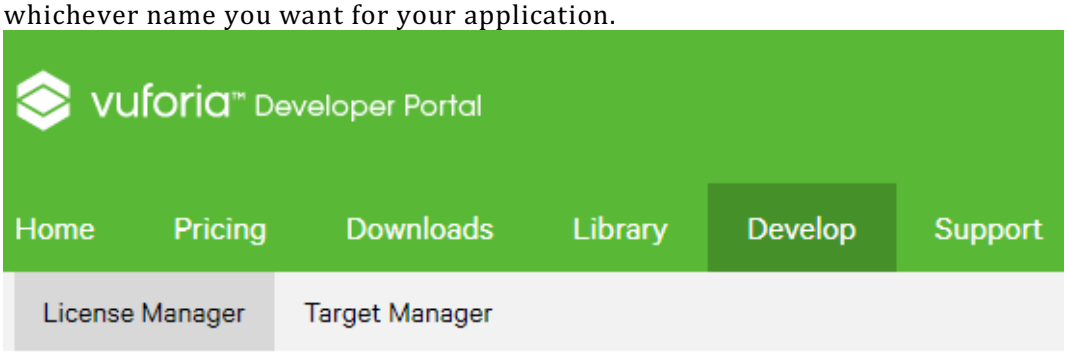
Using our own images as AR markers

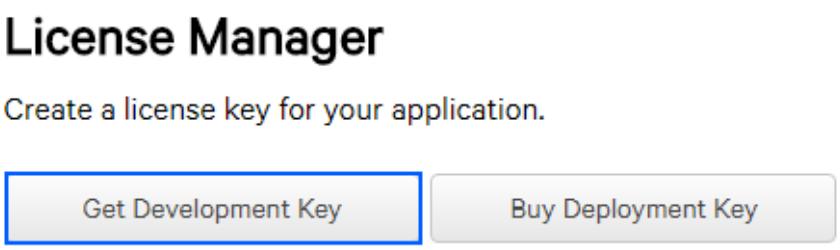
In this second part, we will see how you can use your own images. To illustrate this we're going to use the school's logo.

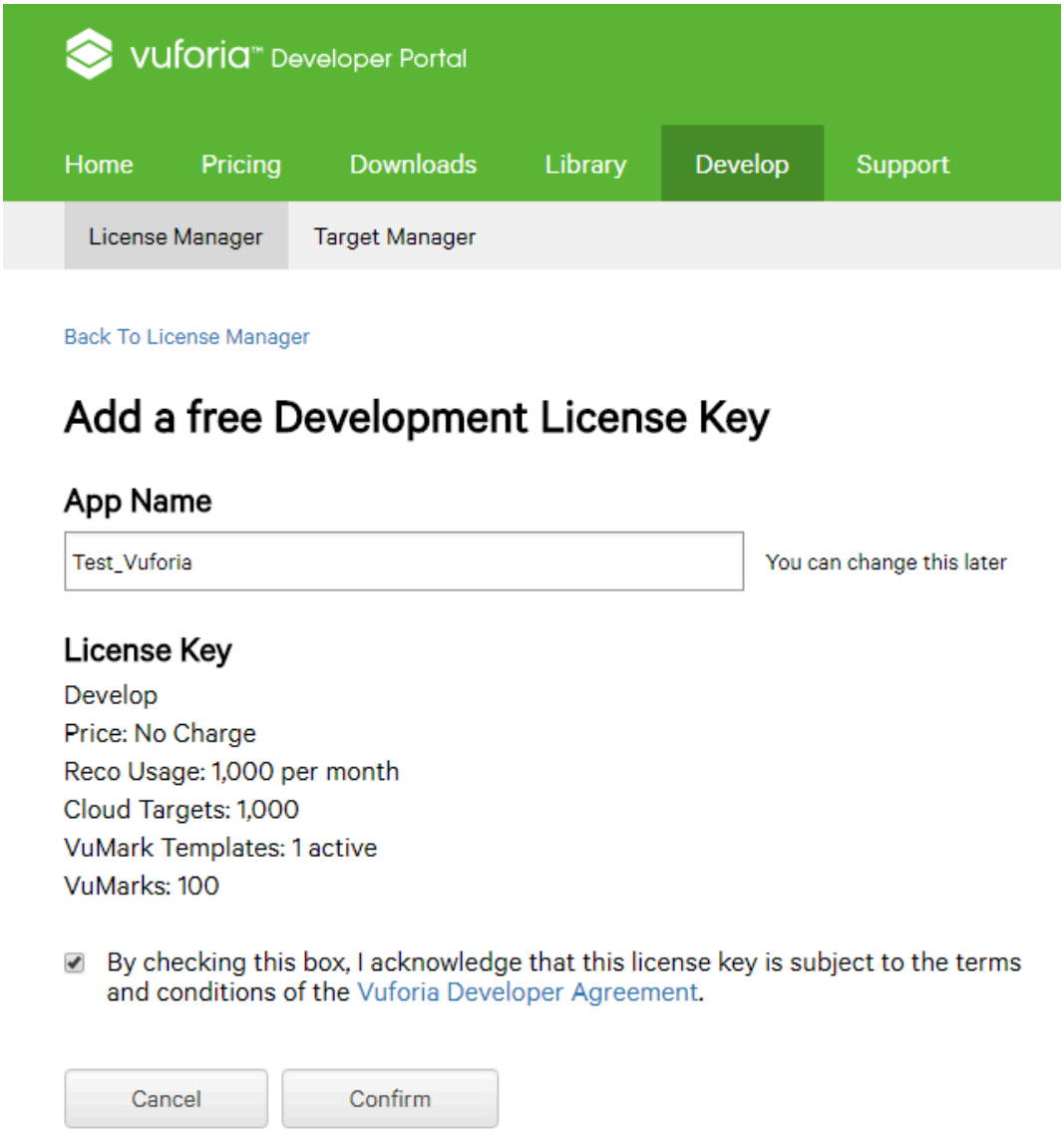
Create a Vuforia Developer Account

The first step is to create a Vuforia developer account, which is required to use your own images. Go to:
https://developer.vuforia.com/
and select create an account, follow the steps and log in to your own page.

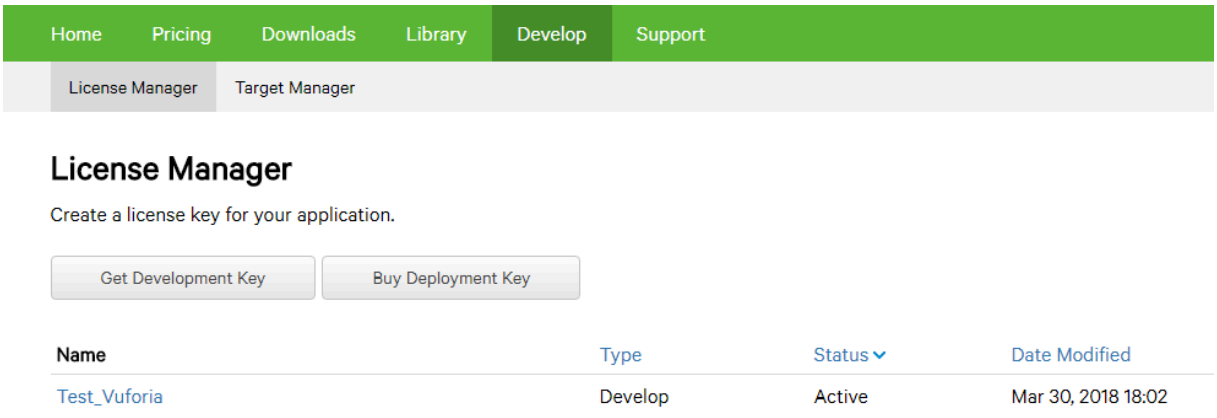
Click on the "**Develop**" tab and then on "**License Manager**". Select "Get Development Key", put whichever name you want for your application.







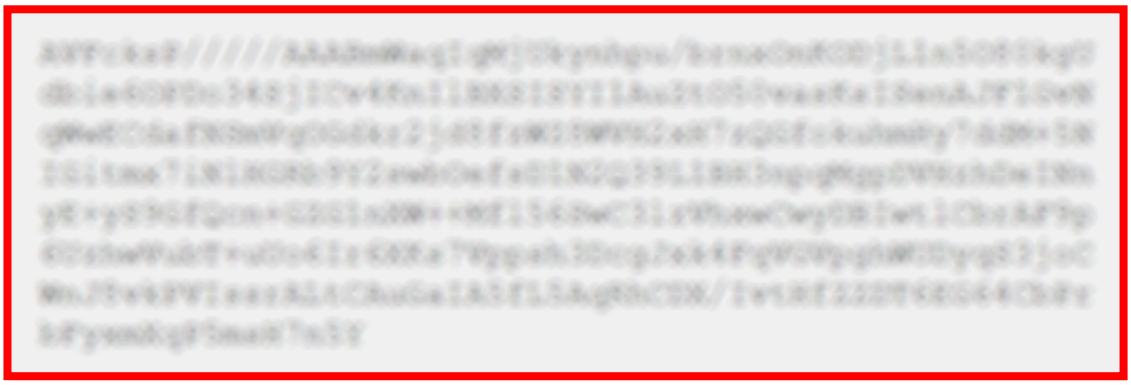
You application is now listed in the "License Manager"' menu, click on it to retrieve the "license key".



Test_Vuforia Edit Name Delete License Key

License Key | Usage

Please copy the license key below into your app



Type: Develop

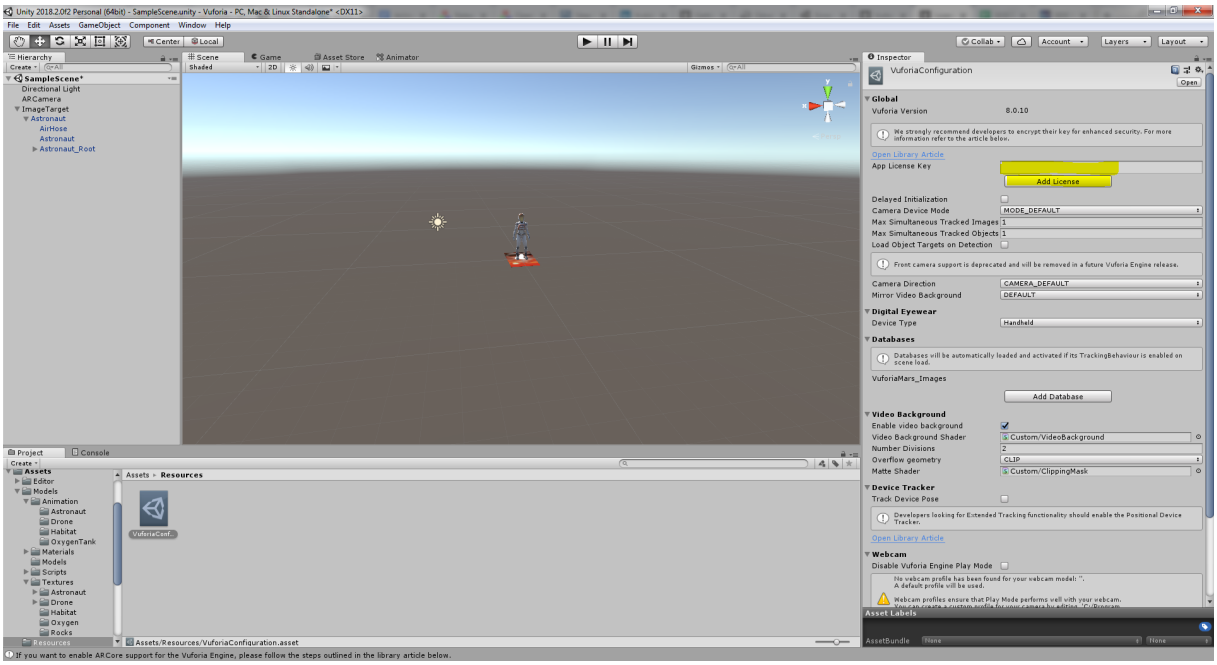
Status: Active

Created: Mar 30, 2018 18:02

History:
License Created - Today 18:02

Back to Unity

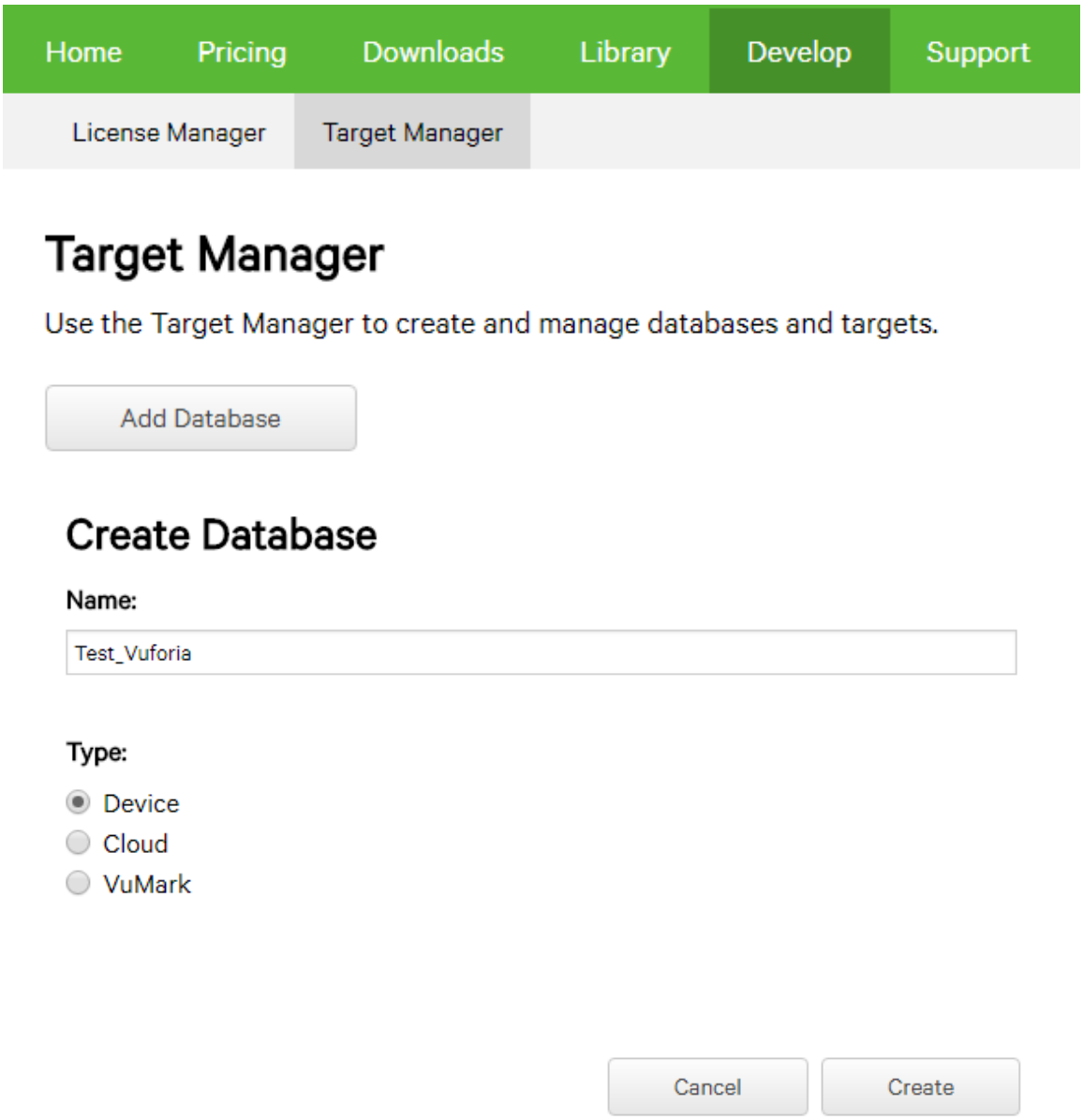
You now have to copy paste this key into the Unity application. To do so, select the “ARCamera” object in the hierarchy view and select “Open Vuforia Configuration”. You can copy paste the key into the eponym field and click on “Add License”.



We now have a license key! We still need to create our own marker using our own images. This step will require the creation of a custom Marker Database.

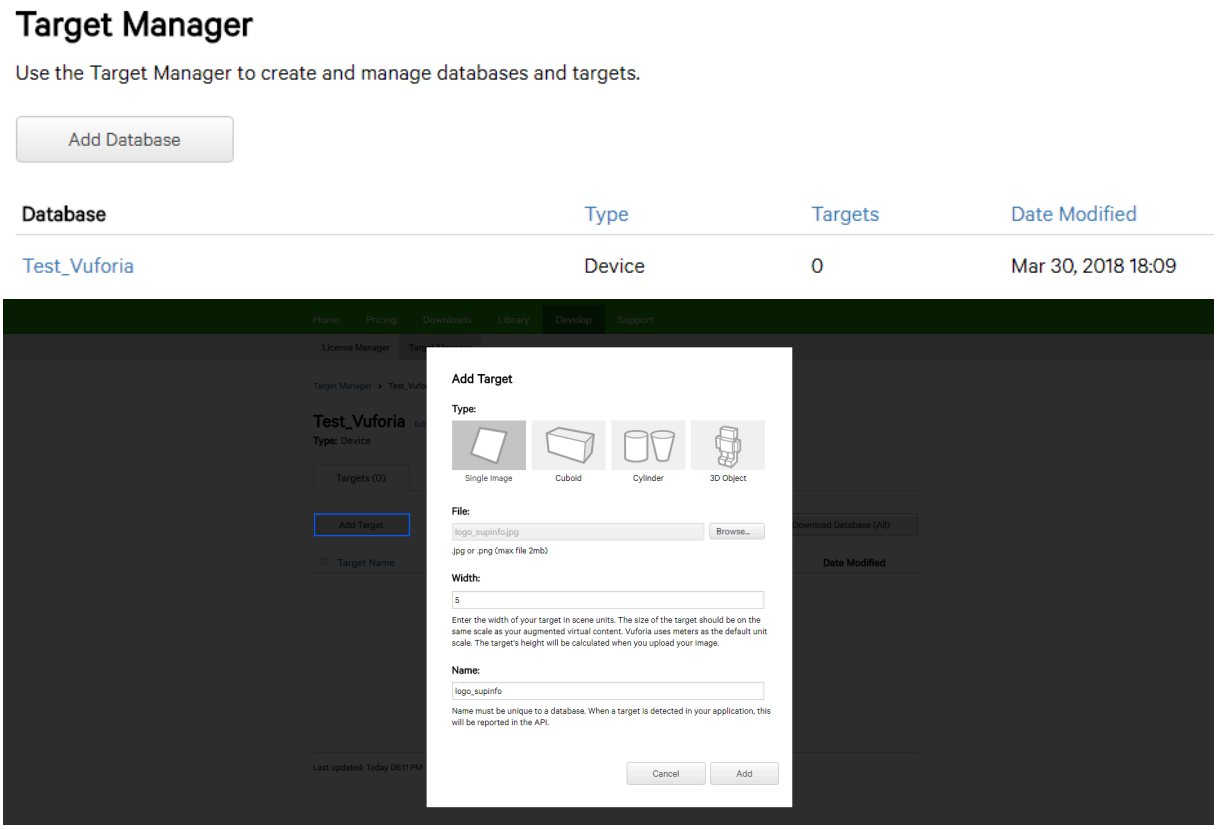
This is also achieved on the Vuforia Developer website. You should now go back to the web page and click on the “**Target Manager**” tab. Click on the “Add Database” button.

Give a name to your database and select the "**Device**" type.

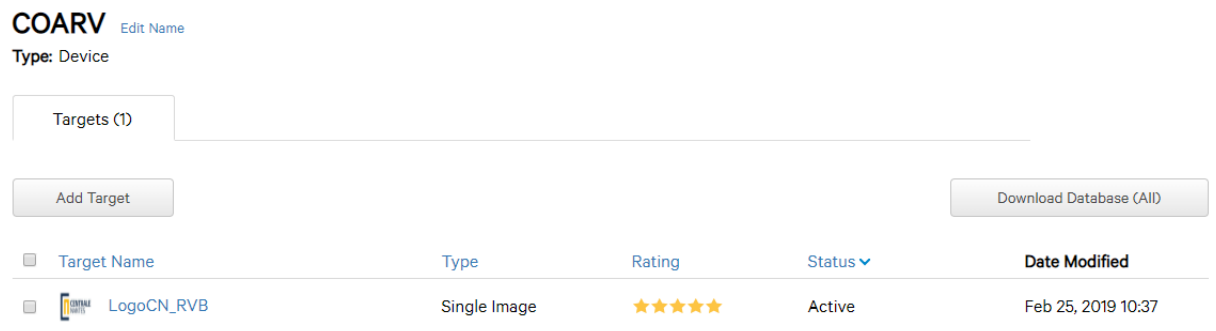


Your database is now listed in the list of databases associated to your Vuforia Developer account. Click on the name of your database and then on the "**Add Target**" button.

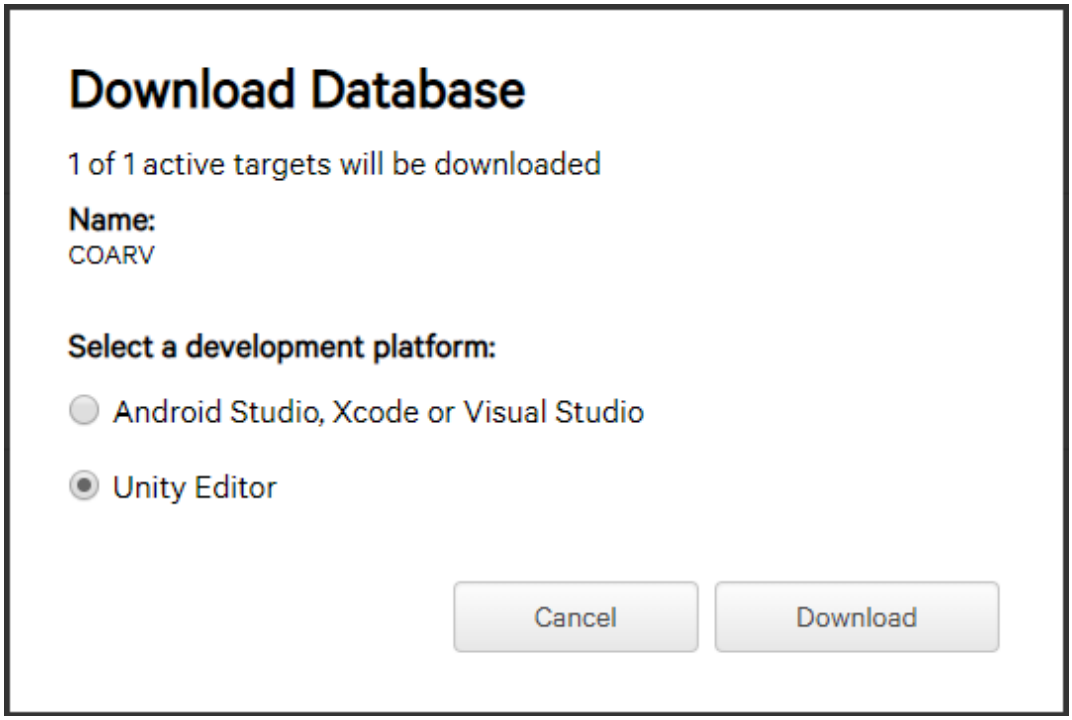
To create your own marker, you should now search your hard drive to look for the image you want (we provide you with the school's logo), and you should put a default size, for example 5. Of course, you can create multiple targets if you want to, just repeat the above steps.



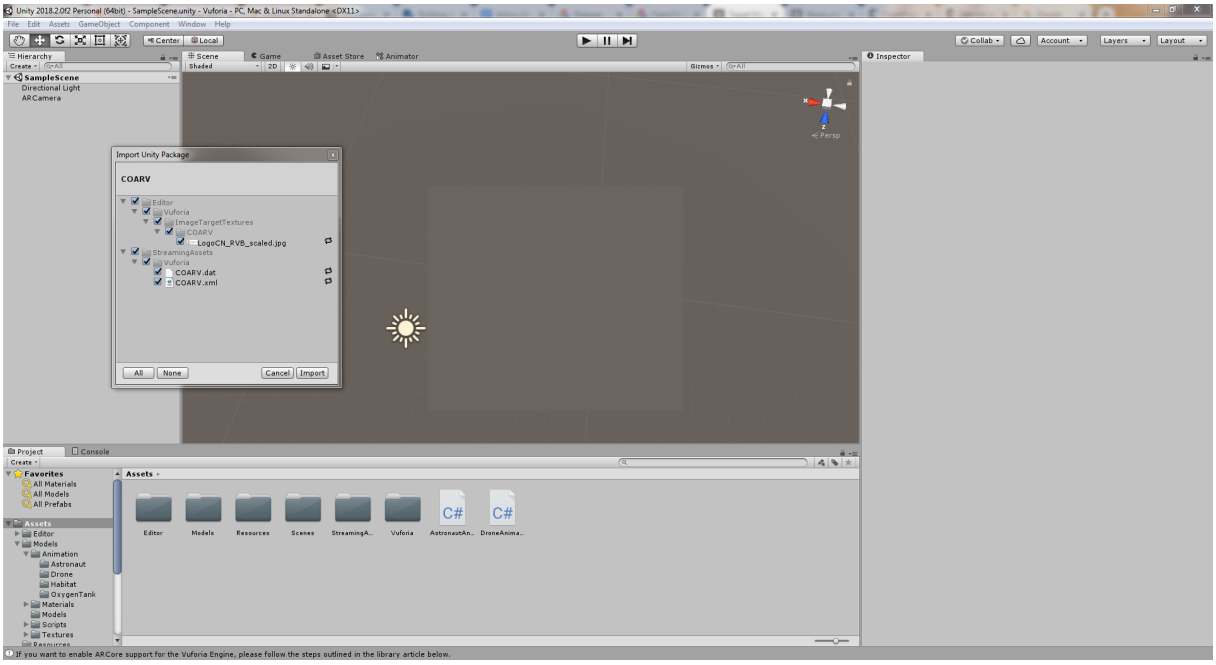
You should now see the marker listed in your database. Please note that Vuforia "rates" your image so that you know whether it is likely to work well as a marker or not. Here, we obtained the highest score meaning that our image is a good choice for an AR marker. This means that Vuforia succeeded in identifying sufficient feature points etc. on the image.



We now have to export this database to open it in Unity. Click on the "**Download Database (All)**" button and choose the "**Unity Editor**" option. This will create a file with the extension ".unitypackage" that Unity knows how to open.



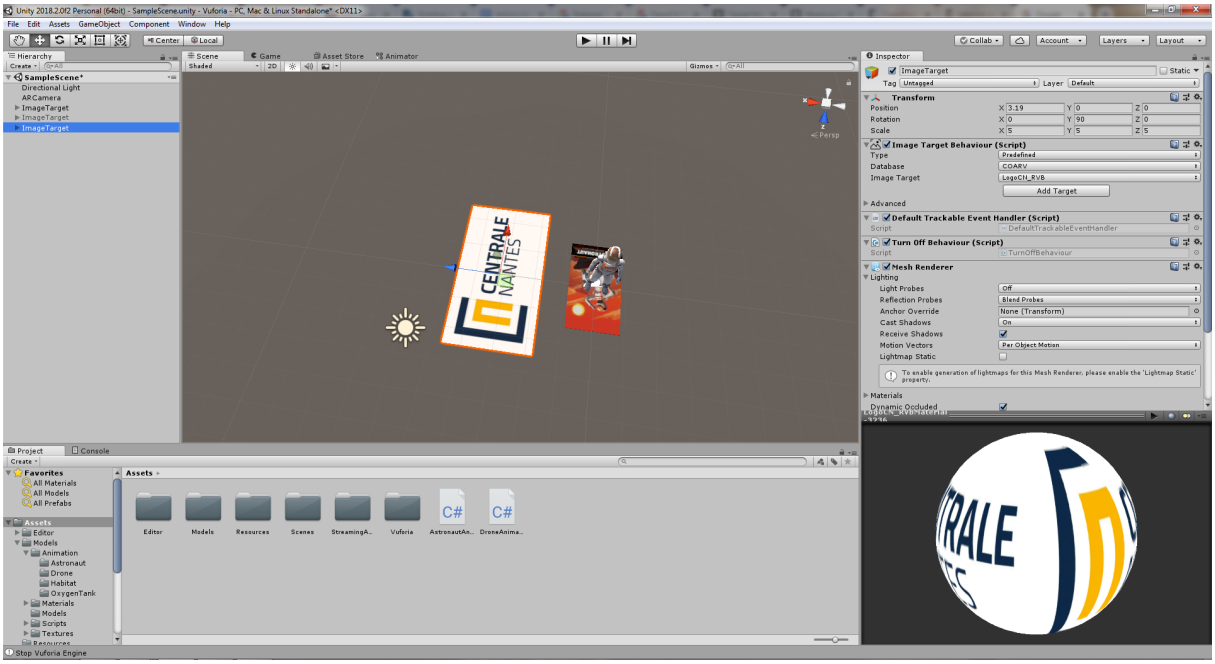
With the Unity project open, go to the downloaded file and double click it. It should open a window asking whether you want to import the package in your project. Click on "**Import**" to proceed.



Go back to the Vuforia configuration page (ARCAmera object and click on Open Vuforia Configuration). In the databases section, you should now see the name of your database being listed along with the previous one (VuforiaMars_Images). If it does not appear, click on the "**Add Database**" button. This should take you back to the Vuforia Developer web page, select your database and click again on the Download Database button. Select "Unity Editor" and it should work this time.

Now we can try adding a new **ImageTarget**. Proceed as before, and add a new ImageTarget object in your scene.

In theory, it should have automatically used the target we just created! You can resize it so that it's more or less the same size that the Astronaut one (i.e. put a scale of 3).

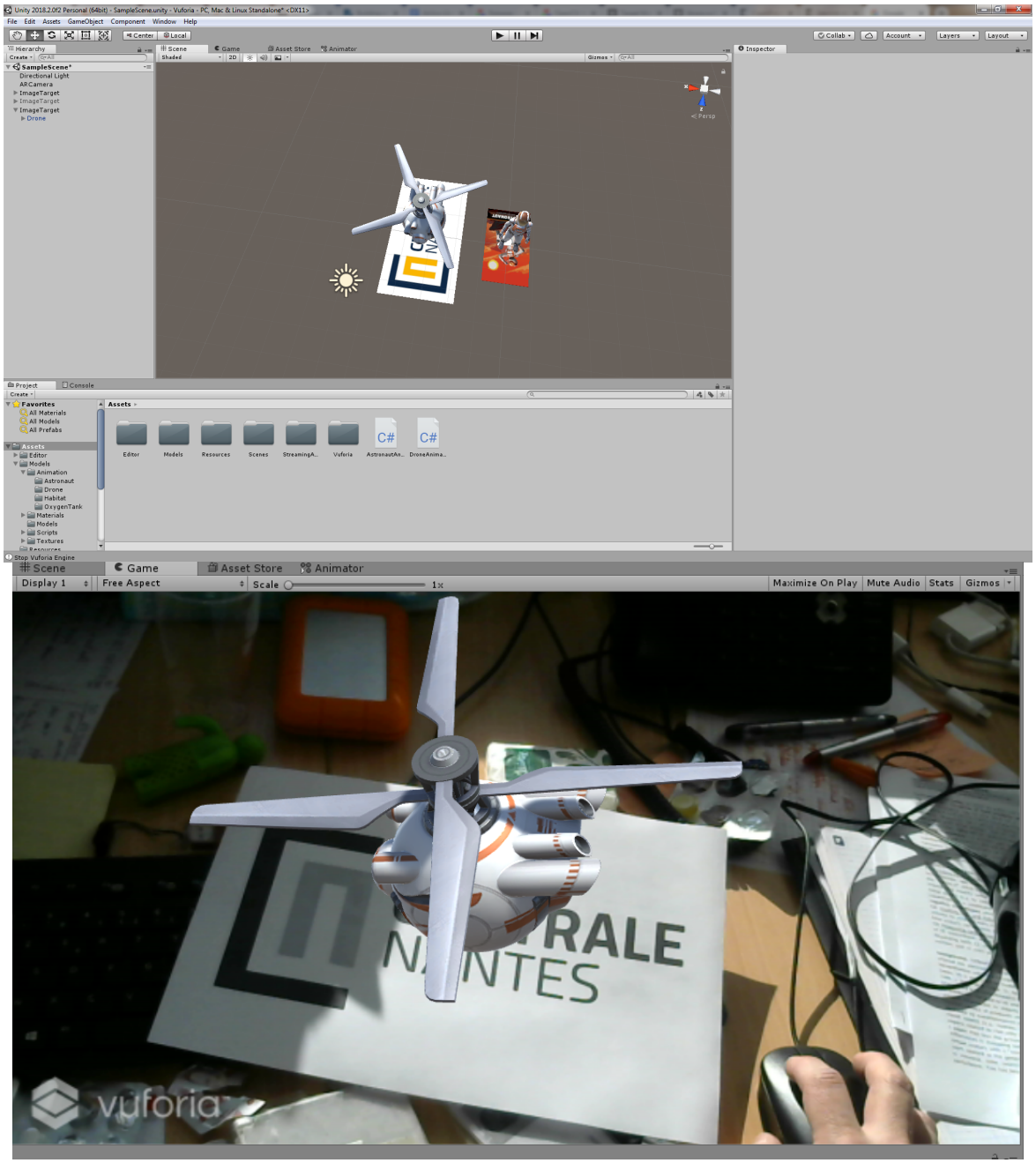


If it did not work, select your new ImageTarget object and select the correct marker!

A bit of work!

Now, to see if you understood what we've done so far, I want you to:

- Add a new 3D model to the new **ImageTarget**
- This 3D model should be the **Drone** model (see the "**Models →Models"** folder)
- Add its texture to the Drone
- Position it correctly with respect to the image target if needed
- Drag and drop the drone animation onto the object
- Add a new C# script to the Drone to control its animation
- Trigger the drone's "scanning" animation when the 'S' key is pressed on the keyboard.
- Try detecting both markers at the same time to display both the Astronaut and the Drone → **If it does not work directly Try to solve it yourself!!**



Conclusion

In this practical we presented how to use Unity and Vuforia to create a small AR application that detects images and displays and animate 3D objects on top of them.

You can of course go a bit further, for example you could:

- Try to deploy the application to a mobile device
- Try to use Vuforia's ability to use 3D objects as markers instead of only 3D images
- Add some more interaction with the objects
- Add some interactions between both objects
- Etc.

The purpose of tis practical was only to give you a short introduction to the creation of AR applications with Vuforia and Unity, since it is practical, quick and allows for the production of quality results, especially when comparing this to other libraries (such as ArUco for example).