

Introduction to a modeling program for MRTI: Unity3D/Vuforia Virtual buttons, and Augmented Reality

J. Vezien, Nov 2022

In this tutored class, we will create a game simulation using the Unity game engine. We will be using Unity 2020.3.11f1 (LTS) or higher. Unity can be downloaded freely from www.unity3d.com. There is also a lot of documentation there, as well as a vast amount of video tutorials on Youtube.

Unity works by managing projects. A project will be created in a directory with the same name, containing all the relevant files necessary to run it. When starting Unity, you can either create a new project or load existing ones.

1) Introduction to the Unity layout

Start Unity with a new project.

Inside a project you can create different scenes, which are as many "stages" for your project. Scenes are stored within the project hierarchy, inside an Asset folder, which normally contains everything your project will use in terms of building bricks.

You can visualize the project hierarchy in the project tab. Next to the project is the Console tab where the program outputs (and other messages such as compile errors) will be displayed.

Next to these tabs is the Hierarchy window where all the objects of the scene (camera, geometric primitives, lights, etc.) will appear. Some objects can be "empty", being just containers for other objects (parent/child hierarchy)

Above are the game (or rendering) window, and the edit window where most of the creation interaction will take place.

Finally, the inspector window will serve to display, and possibly edit, the objects' properties.

2) Create a few objects

The 2 main ways to create objects are with the GameObject menu and in the "project tab", sub-item "all models".

First, create a 1m x 1m plane object in the XZ plane (you will scale it down by 0.1 factor in X and Z direction). Rename it "Ground". Note that Unity uses an *indirect* reference frame, with Y in the "up" direction (objects fall in the Y down direction).

We will need a ball to play with: create a sphere of radius 10 cm and position it at the center of the arena. Name it Ball. Place this ball slightly over its "zero" initial position (Y=0.25).

Now we need to dress up these objects a bit. Materials can be found in the project tab, in All Materials. Select one, and drag and drop it on the elements in the scene window.

3) Camera and lighting

To see the objects properly, we need to position the camera accordingly. The camera is an object like all the other; it can be placed where needed. The game window shows the camera view by default.

Also at least one light is needed. By default, a directional light is there. Position it properly so that that the arena you just created is fully visible.

You can add some lights to make the scene more or less visible, highlight some objects, etc.

4) Make the ball a physical object

Next we need to make the ball a real physical body that can fall and roll on the board. To do that, we need to add a Rigidbody property to the ball. Select the ball, and select Component-->Physics-->Rigidbody. Change the Interpolate property to "Interpolate", and the Collision detection to "Continuous dynamic".

Then we need to attach a physical material to the ball: first, let's create one with Assets-->Create-->Physic Material. In the inspector, make bounciness maximum (=1) and Bounce Combine to Maximum. This will make the ball rebound infinitely without ever losing speed. Now attach this physic material to the ball by drag and drop the material in the Material field of the sphere collider property of the ball object.

Now hit the "Play" button. Normally, the ball should fall and bounce on the plane forever. If it doesn't, check the Time Setting in Edit-->Project Settings, and change it to a lower or higher value. How do you explain the result?

In the Rigidbody property of the Ball, deactivate gravity by default. We will reactivate it when we start the game with a script (see below).

Next we can input an initial force to the ball. Download the script BallBehaviour, in C#, from the download archive.

In the script, one should declare a public Vector3 InitialImpulse variable (this variable will be editable in the properties of the object the script will be attached to). Then in the Start routine, one can use AddForce to input this impulse to the Rigidbody.

This is what the script should look like:

```
using UnityEngine;

public class BallBehaviour : MonoBehaviour
{
    public Vector3 initialImpulse = new Vector3(0.03f, 0, 0);
    public GameObject position_initiale;

    // Use this for initialization
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        // Lorsqu'on appuie sur la barre espace, le jeu commence,
        // la balle est lancée depuis le milieu
        if (Input.GetKey(KeyCode.Space))
        {
            transform.position =
            position_initiale.transform.position;
            GetComponent<Rigidbody>().velocity = initialImpulse;
        }
    }
}
```

Now drag and drop the script on the ball to attach it to this object. You can see that the public fields declared in the script automatically appear in the interface. Convenient!

You should now create an Empty GameObject (a 3D object that does not appear on the screen, corresponding to a reference frame). Rename it “InitialPosition” Put it where you want the starting point of the ball to be located. Drag and drop the object from the hierarchy tab into the `position_initiale` field of the BallBehaviour script of the ball.

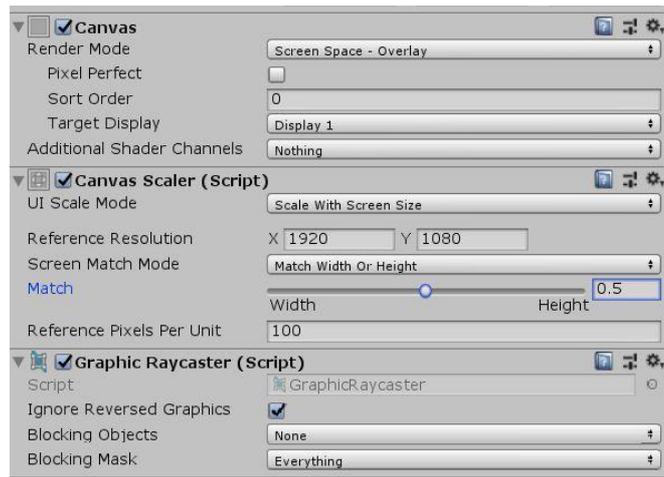
Run the program, and hit the space bar to see the result!

5) Make the scene a bit more interesting

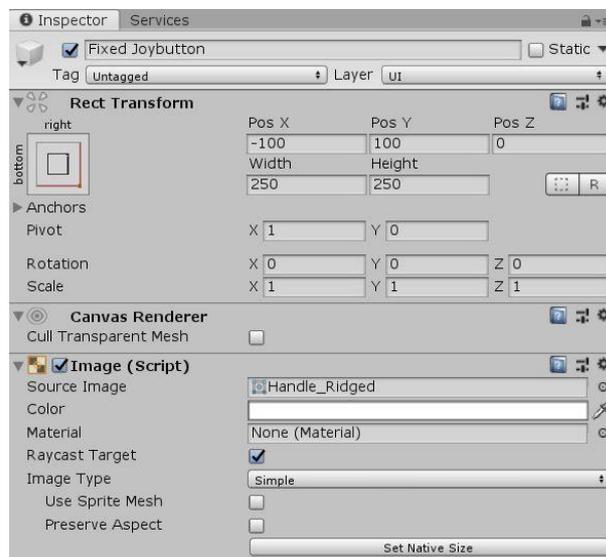
- Add walls, some extra objects the ball can hit, etc. These objects do not have to be physical (i.e. do not assign a `Rigidbody` component to them) otherwise they will be subject to gravity. A physical object will interact with a non-physical object. You can also turn the gravity off on the `Rigidbody` component of an object.
- Visit the Asset Store (Ctrl-9) to download materials to make your objects more beautiful. Type “Materials” in the search window of the store, and sort by price. Select a free package, download it, then Import it to your project. Now, in your “Assets” subdirectory, you should find the proper material (materials are listed in the Unity “Folder” explorer as spheres covered with a specific coating). Drag and drop the material on an object in the Scene window to assign a material to the object. Alternatively, you can drag and drop it in the inspector window.

6) Add 2d controls

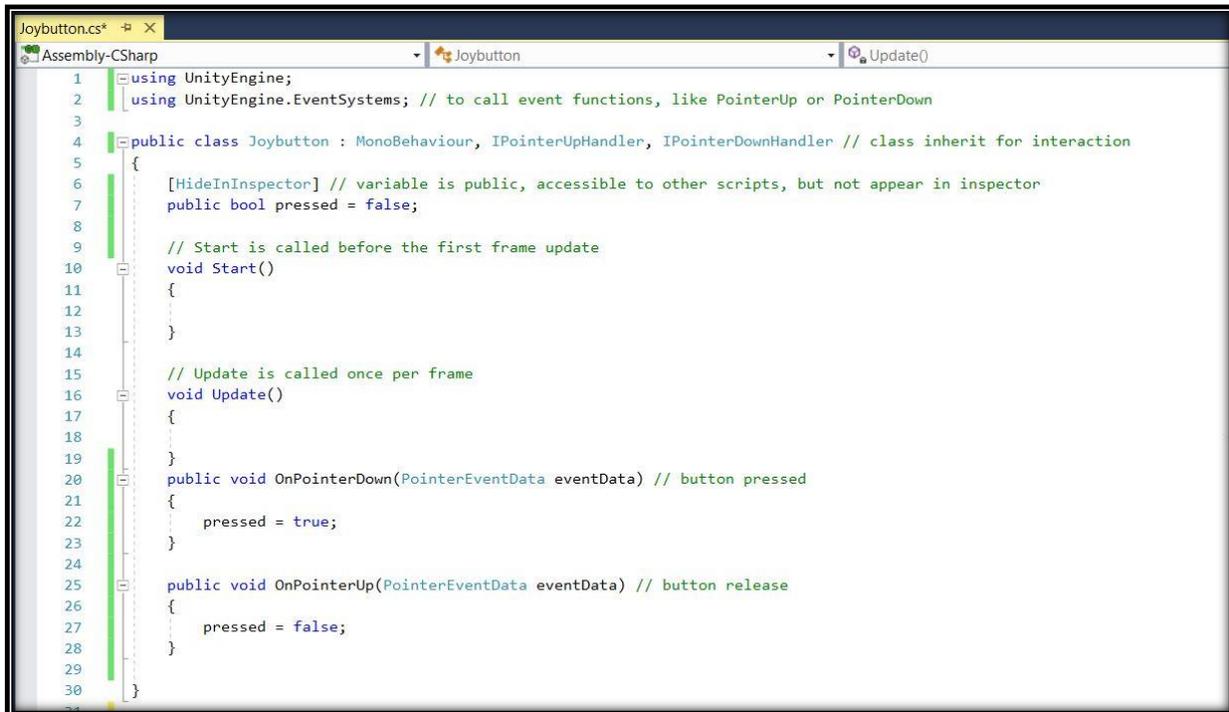
- Got to the asset store and retrieve the “Joystick pack” (free package).
- Create a “Canvas” object (GameObject->UI->Canvas). This will be a 2D object on which you will put the interactors: a joystick and a button. Adjust settings to the following:



- In the Joystick pack folder, open the Prefabs dir. Prefabs are ready-made objects. Drag and drop a Fixed Joystick object into your scene. Make it the child of the Canvas one. Adjust width and height to 256, and change the Source Image to “Handle Ridged Arrows” for a better look.
- We will also need a button in our interface. This does not exist, so let us create it by adding first an Image object to the scene (GameObject->UI->Image). Rename it Fixed Joybutton. Make this image a child of the canvas as well. In the Image script variables (created as a property of the image object), change the field “Source Image” to “Handle Ridged” for a better look. In the Rect transform field, set the anchors to Bottom Right, and adjust the field to the following:



- e) Let us attach a script to the button, namely the `Joybutton` script from the archive. The script looks like this:



```
1 using UnityEngine;
2 using UnityEngine.EventSystems; // to call event functions, like PointerUp or PointerDown
3
4 public class Joybutton : MonoBehaviour, IPointerUpHandler, IPointerDownHandler // class inherit for interaction
5 {
6     [HideInInspector] // variable is public, accessible to other scripts, but not appear in inspector
7     public bool pressed = false;
8
9     // Start is called before the first frame update
10    void Start()
11    {
12    }
13
14    // Update is called once per frame
15    void Update()
16    {
17    }
18
19    public void OnPointerDown(PointerEventData eventData) // button pressed
20    {
21        pressed = true;
22    }
23
24    public void OnPointerUp(PointerEventData eventData) // button release
25    {
26        pressed = false;
27    }
28
29
30 }
```

- f) Finally, we can make the ball react to the inputs we just defined. Use the `BallBehaviour_enhanced` script which looks like this (you will need to rename it in order to replace the previous `BallBehaviour` script):

```

BallBehaviour_enhanced.cs
Miscellaneous Files | BallBehaviour
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class BallBehaviour : MonoBehaviour
6  {
7      protected Joybutton joybutton;
8      protected Joystick joystick;
9      protected bool jump = false;
10     public float scale_inter = 2f;
11     public GameObject position_initiale;
12
13     // Start is called before the first frame update
14     void Start()
15     {
16         joybutton = FindObjectOfType<Joybutton>();
17         joystick = FindObjectOfType<Joystick>();
18     }
19
20     // Update is called once per frame
21     void Update()
22     {
23         Rigidbody rb = GetComponent<Rigidbody>();
24         rb.velocity = new Vector3(joystick.Horizontal * scale_inter,
25                                 rb.velocity.y,
26                                 joystick.Vertical * scale_inter);
27
28         if (Input.GetKey(KeyCode.Space))
29         {
30             transform.position = position_initiale.transform.position;
31             GetComponent<Rigidbody>().velocity = new Vector3(0f,0f,0f);
32         }
33
34         if (!jump && joybutton.pressed)
35         {
36             jump = true;
37             rb.velocity += Vector3.up * scale_inter;
38         }
39         if (jump && !joybutton.pressed)
40             jump = false;
41     }
42 }
43

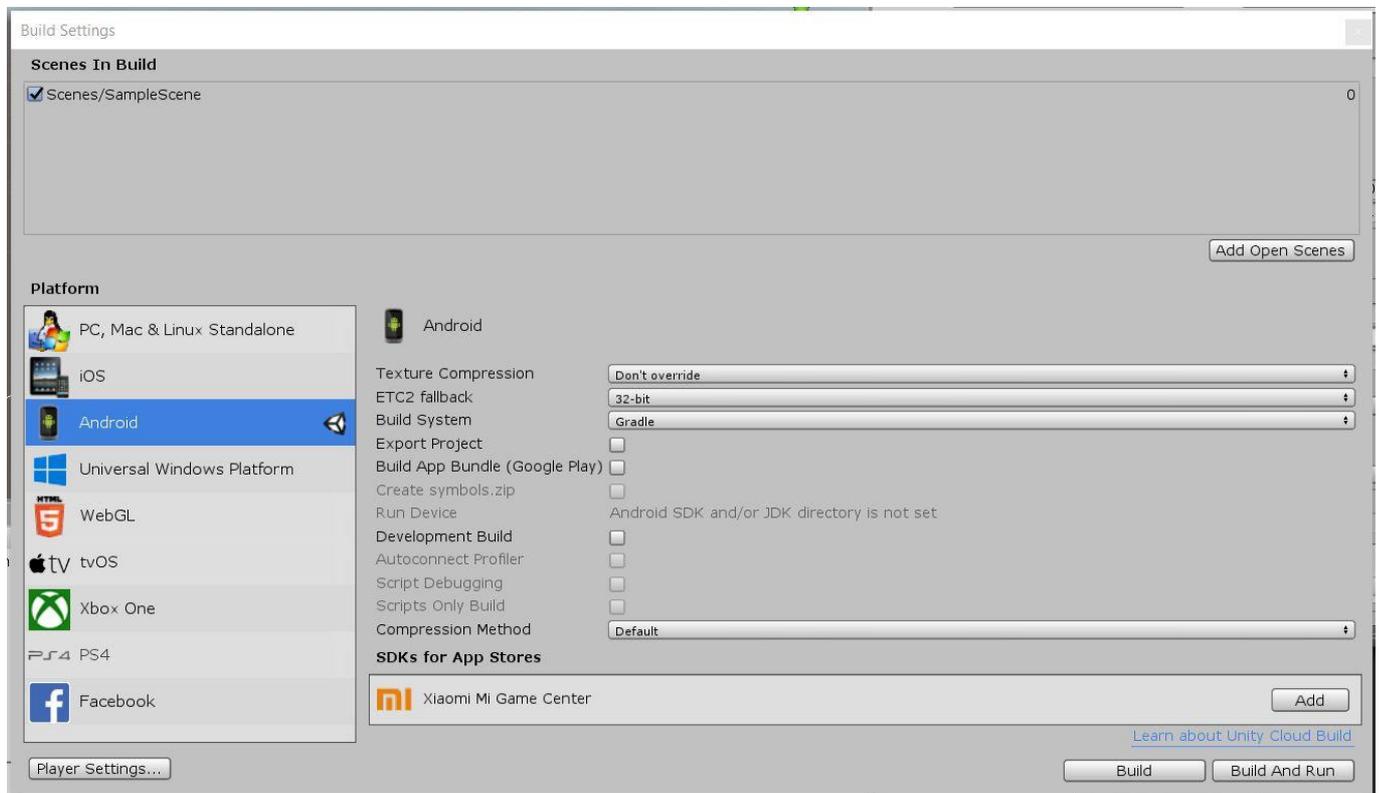
```

Try and interpret the content of this script by yourself.

7) Run the new program and download it to Android

- To stop the ball from going in all places, reset the Material field of the Collider to None, and set the InitialPosition to slightly above the plane.
- Run the program in unity by pressing "Play" button. Use the mouse on the pad and button to manipulate the ball.

- c) Optionally, add a “Init” button to the touch interface and link it to your custom Init.cs script callback.
- d) Go to the File->Build menu, and set the platform to Android. Do not forget to add the open scene to the list of build:



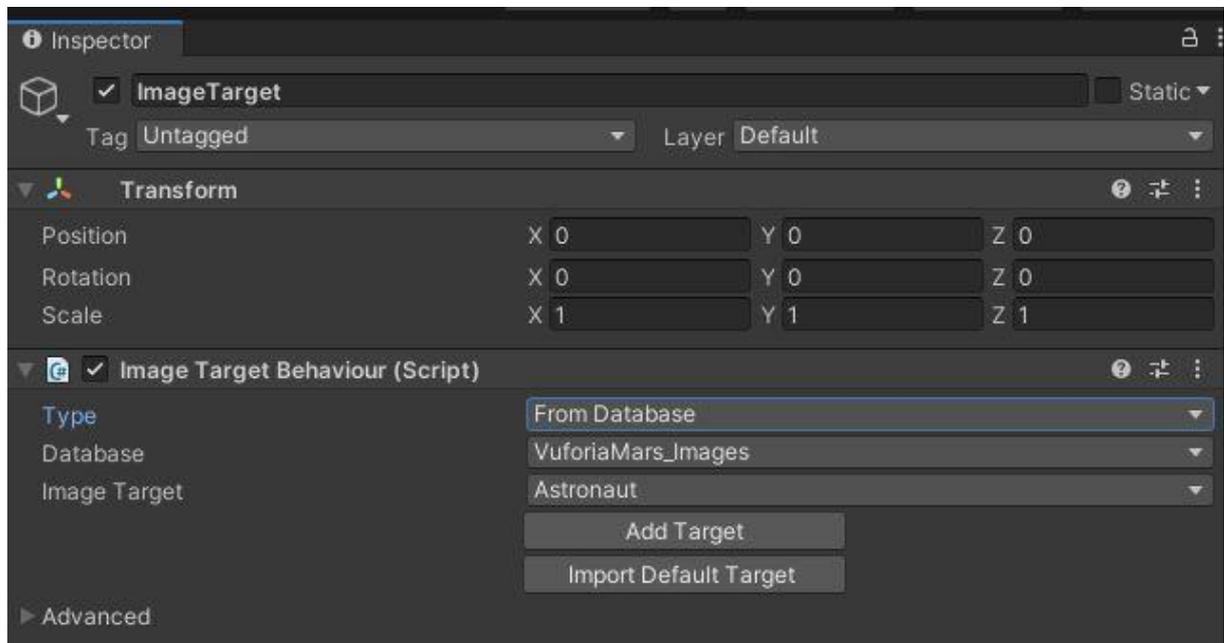
- e) Build the application. Select a location (you should create a Build/Android subdir in your project), and build to create the apk file. Plug your smartphone or tablet with USB cable, and transfer the file in your download dir. Launch the app to install it, and voilà!

8) Adding AR functionalities

We will be using Vuforia to provide Augmented Reality (AR) to our Unity simulation. **Please follow the dedicated tutorial “Augmented reality for Unity: the Vuforia package” for a complete introduction.** Then follow the steps below:

- a) The first thing we need to consider when adapting a program to AR is **scale**. Most mainstream cameras available today are set for closer range pictures. Furthermore, it is difficult to setup large markers. In our case, we want to use Vuforia markers, either the default ones (Images from the “Mars” database), or a newly created marker (see Vuforia tutorial and documentation at <https://library.vuforia.com/>). These are printed on a regular A4 page, hence with a dimension around 20cm. Therefore, you should scale your ground plane and ball accordingly, so that you can see both when you image your physical target.

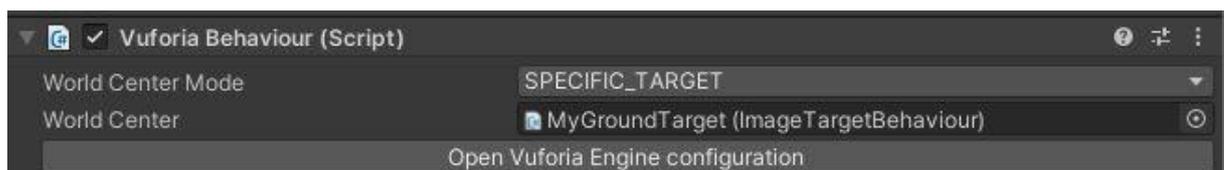
- b) In our simple example we will need only one AR tangible control: placing the playing ground in the real world. To do that, we will create a simple ImageTarget (GameObject → Vuforia Engine → Image Target), and make the ground plane the child of this target. Be sure to place the virtual ground object “just on top” (slightly above) of the ImageTarget (say 5mm). Select the correct properties for the ImageTarget, especially the real image it corresponds to. For instance:



(but you can create your own database and targets, see the dedicated tutorial).

Once you have set the target and properly configured the Vuforia engine (again, see the specific tutorial for details), you can try it in real life! Of course you will need the physical target to look at, either as a real piece of printed paper, or on an auxiliary display (it may not be the correct size, in this case the simulation will not be the expected scale, but will still work fine).

- c) When starting the program, you may notice that the ball seems to bounce all over the place, and fall off the ground to nether land. This is because, by default, the world origin is placed with your image capturing device (webcam, smartphone) ... and so is the vertical direction, determining gravity! Fortunately, we can change this by going in the ARCamera inspector, and change the property “World Center Mode” (just above “Vuforia Configuration”) to SPECIFIC_TARGET, and selecting the proper ImageTarget to define the “World Center”.



(Alternatively, use the FIRST_TARGET option to use the first target encountered).

This way, when your ground plane `ImageTarget` is encountered and tracked, it will determine the origin of the world, hence the gravity direction. Try it! Now everything should be fine, the ball should keep bouncing on the floor, whatever your webcam position is. And of course, the virtual buttons should still work fine.