

LP2B – Multimedia: Digital Representation

- LP2B Tutorial Class n°3 -

Sound, Video, and Component Interaction

Using more component types and understanding their C# logic further

This class will give an overview of :

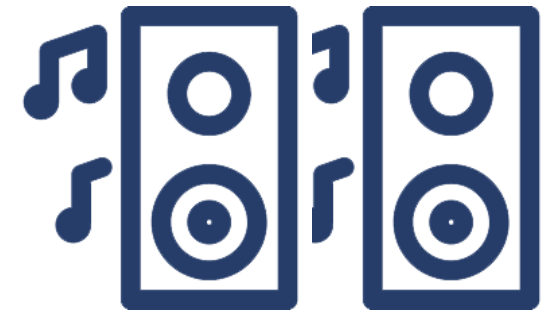
- *Importing Sound and Video assets -*
- *Creating and Using AudioSource Components without code -*
- *Accessing and controlling a component in the Code -*
- *Creating a new component entirely through Code -*
- *Using the Video Player component without code-*
- *Basic interactions with a video through code-*

Lets go back to our Basket and Apples...

- We are going to take our project from last time, and add sound to it!
- Open Unity Hub and use it to re-open the project from last time
The usual. Hub being a light app, it doesn't make much of a delay
- We are going to add music and sound effects to this project
Once we do that, our project will finally have two media, and be true multi-media!
- In Unity, sounds have more in common with Images than you might expect
Even though they are completely different media, they have similarities. Because...
- The core logic "GameObject + Component + Asset" also applies to sounds!
That's why the process is surprisingly similar to the one we used for images
- Let's start by importing some sound assets to the project...



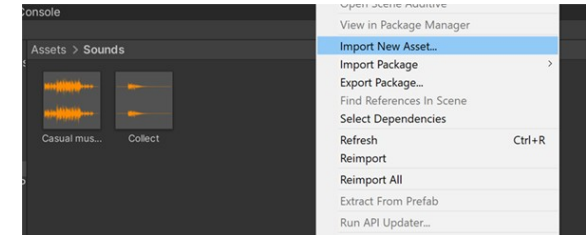
*The project in the state we left it in.
(with a variable number of "extra objectives" done)*



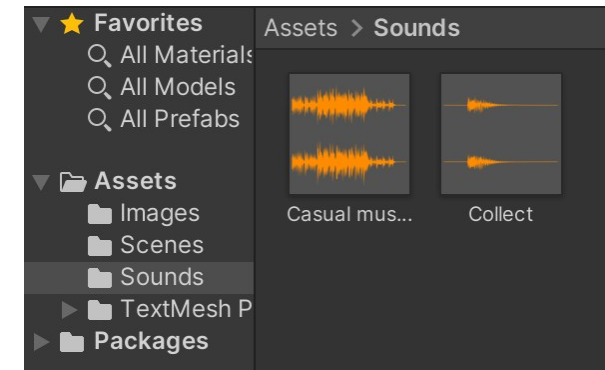
*After today, it will look the same,
But will have music and sound effects!*

Preparing Audioclip assets for Use

- We will now begin the journey to add sound to our project!
And by doing so, it will finally have 2 types of media, making it multimedia!
- First step : we need to have sound assets available in our project
It's exactly like we need image files to give to the SpriteRenderer components
- Take the provided Sound Effects and add them to the project
It works just like adding images, so you should be able to do it on your own
- Imported Sound Effects become instances of the **AudioClip** class
That's why Unity users call sounds "AudioClips". C# code also uses that name
- Once they're imported, organize those sound assets in a separate folder
Please take the habit of organizing your projects as soon as possible
- You can either import directly in that folder, or move the assets later
You can move assets easily by drag and dropping them on another folder



You can import either through this contextual menu, or by directly copying the files into your Assets folder



The Imported Assets, properly sorted into a separate "Sounds" folder

Adding a looping Music through the GUI

- To use AudioClip assets, we need the matching component : **AudioSource**
Just like our Sprite Assets need the SpriteRenderer component to be used
- The familiar “GameObject + Component + Asset” logic appears once again!
And this should be no surprise. It is the core logic of Unity after all
- Select the basket GameObject, and add an “AudioSource” component to it
You should be able to do it : it’s just like adding any other component!

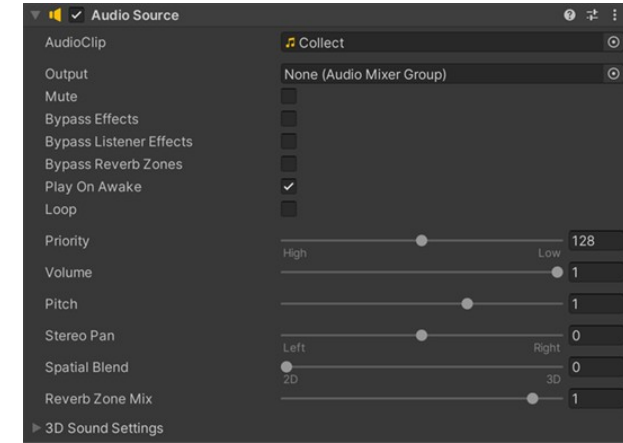
In case you forgot :

- In the first tutorial class, we added “Rigidbody2D” components with the “add component” button. You can add an AudioSource in the same way!

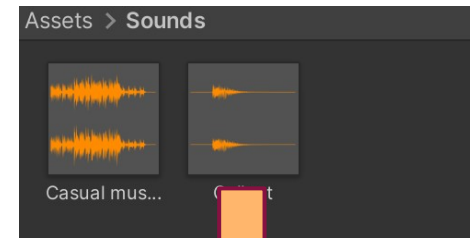
- Drag and Drop the “collect” sound into the AudioClip field of the AudioSource
Then, launch your game. **You should hear the sound play right as it starts!**

Now, it's your turn!

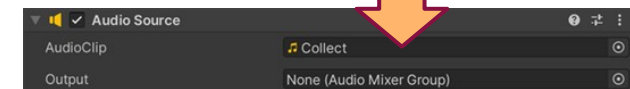
- Use this solution to add the “casual music” as a loop in both scenes



The AudioSource Component as it appears in the Inspector Tab



DRAG AND DROP!



Accessing a Component through the Code

- We already saw a way : public attributes. But there is another one.
- In a C# Script, we can access components connected to the same GameObject with : `GetComponent<TypeToGet>()`
- The method returns a **reference** to the component if found. It returns **null** otherwise.
null is the name given to an empty reference
- The reference will be lost if it is not saved in a variable or attribute. This is why the method is usually used this way :

```
TypeToGet ma_variable = GetComponent<TypeToGet>()
```

Now, it's your turn!

- Create an AudioSource and add it to the Basket. Disable the option "Play on Awake"
- Write a code in the Start() method to get its reference. We will use it later.
- Make sure the reference is saved in an Attribute, so we can use it whenever we want

What are References, anyways?

- References are a key concept in object-oriented programming. And therefore in Unity as well.
- In practical terms, a reference is a copy of the memory address of an original object.
- The reason they are interesting, is that changes made through a reference will affect the original object :

Here, we save the reference returned by "Instantiate". This is a reference to the apple that was just created by that very same "Instantiate"



```
GameObject newApple = Instantiate(apple_prefab);  
newApple.transform.position = new Vector3(randomX, 6.0f, 0);
```



*On the next line, we change the position of the reference.
By doing so, we actually change the position of the original object*

- Similarly, if you call a method of the reference, it is actually called on the original object :

*Remember this?
We had a reference to our TextMeshPro saved in "displayed_text"*



```
displayed_text.SetText("Score : " + score);
```



When we call the "SetText()" method on the reference, it is called on the original object

- In case you wondered, this is why the reference need to be typed. Like with **TypeToGet** before

Manipulating our AudioSource with Code

- As we just saw, manipulating a reference to an AudioSource is the same as manipulating the original
...But what can the original actually do? What are its attributes and methods?
- AudioSource wasn't coded by us. Its specs were created by the Unity team. ...So we have to learn them!
- The code below shows the attributes and method you will need to use most often :

```
sfx_player.loop = false;  
sfx_player.volume = 0.7f;  
sfx_player.clip = mainMusicLoops[0];  
sfx_player.Play();
```

→ "loop" attribute : if true, the sound will loop when played. If false, it won't.

→ "volume" attribute : A float. Determines the % volume when the sound will play later. For instance, 0.7f = 70% volume.

→ "clip" attribute : A reference to the audioClip to play. No need to set this up if you already did with a drag-and-drop.

→ "Play()" method : Plays the clip in "clip" attribute, according to the other settings

- Full specs of the class are available here : <https://docs.unity3d.com/ScriptReference/AudioSource.html>

Now, it's your turn!

- Edit the Script of the basket so it plays the "collect" sound every time an apple is collected

Creating a Component entirely through Code

- Adding the components through the GUI and then doing GetComponent() works well, but often, a project just needs the flexibility of adding and removing components on the fly
- If you coded in C# or other languages, you might expect the keyword “new” to be the solution...
- But in the case of components, this **DOESN'T WORK!** *(the reason being that they can't exist without a GameObject)*
- The solution is to use a specific method from the GameObject class : `gameObject.AddComponent<TypeToAdd>()`
- Just like before, the reference to that new component will be lost if it is not saved. So we do this :

```
TypeToAdd ma_variable = gameObject.AddComponent<TypeToAdd>()
```

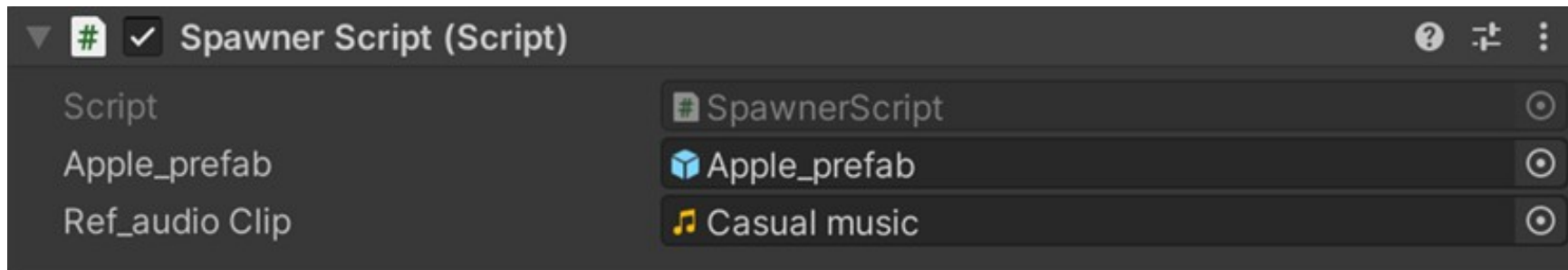
Warning : Do not confuse “GameObject” *(the class)* with “gameObject” *(a reference to the GameObject attached to our script)*

Let's try Component Creation

Delete the AudioSource that plays the “casual music”. We are going to generate it though code instead!

Now, it's your turn!

- Edit the “Spawner Script” so it generates an AudioSource through code on Startup
- Set up that audio source so it loops, and tweak its volume as you wish
- You will have to provide the AudioClip's reference through code.
- To do that, use a public AudioClip field, and drag the audio clip into it
- In the end, the AudioSource will be code generated, but the AudioClip will be drag-and-drop!



Remark : there is a way to import the asset entirely through code as well, but we won't see it this time

Nice! You now can use Sound in Unity!

Good job on understanding the basics of playing Sound in Unity!



Of course, there are more advanced features, such as sound mixers, reverb, 3D sound...
But you'd be surprised how much can be achieved with only what you just learned!

And what about Video Media ?

- Video media uses the same logic as all the others we seen before :

An Asset + A Component + A GameObject = A medium used in Unity!

- The first step would be to import the video as assets, as usual

And then we feed that data to a Component on a GameObject. Same logic indeed!

- For videos, the chance of a format not being supported tends to be higher

If the video doesn't show properly as a preview in assets, either download more codecs, or convert the video to another format with another software

Tip : Kdenlive is a free video software you can use for your conversion needs

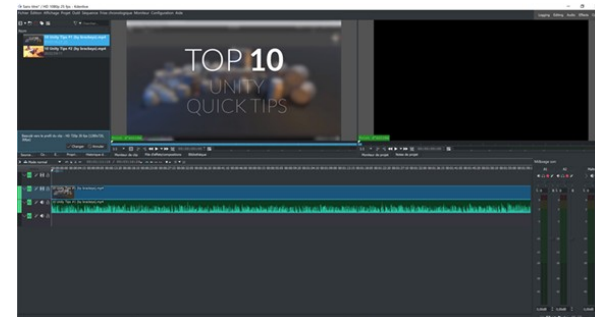
- Imported Video Files become instances of the **VideoClip** class

That's why Unity users call videos "VideoClips". C# code also uses that name

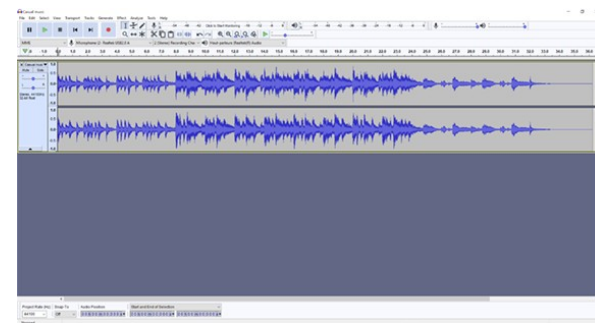
- If this feels familiar. That's because it is. It's the universal Unity logic!

- However, we do not really need video in our "basket and apples" game...

*When working on Multimedia Projects,
It's a good idea to be aware of the (legally)
free software available to edit content...*



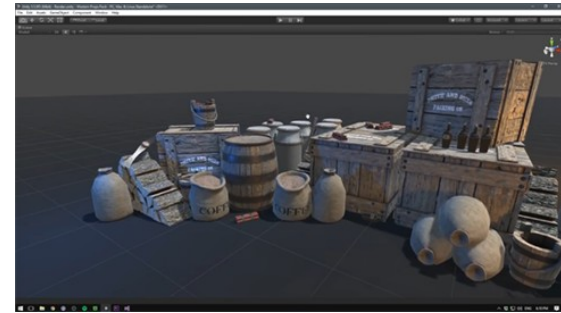
*Kdenlive is one of those choices
When it comes to videos!*



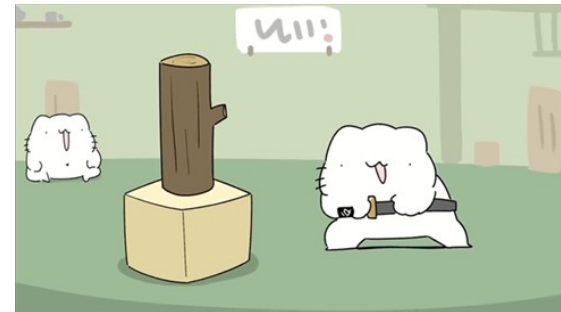
*...And for the audio files we talked about
Before, there is Audacity!*

Let's make a New Project for that...

- Save your current Project and close it. Go back to the Hub.
- Create a new project with the "2D" template. Call it "Test Video"
We are going to make a mini player that plays various short funny videos
- Import the provided videos into the project as Assets
The videos were tested before and should be compatible. If not, download codecs
- Make sure to organize the assets in folders properly!
And yes. I will keep annoying you with that until you do it (you will be glad later!)
- Drag-and-drop a clip into the scene. Test the app. It already works!
Just like a "SpriteRenderer" was created automatically when we did that for images,
A "Video Player" was created automatically by doing the same with a video
- Our video might work, but we have no control at all over it!
Which is why we'll talk more about the Video Player component now!



Two of the videos provided are Unity tips by the Brackeys Youtube Channel. They are a few minutes long, which can be a demerit for testing "end of video" events

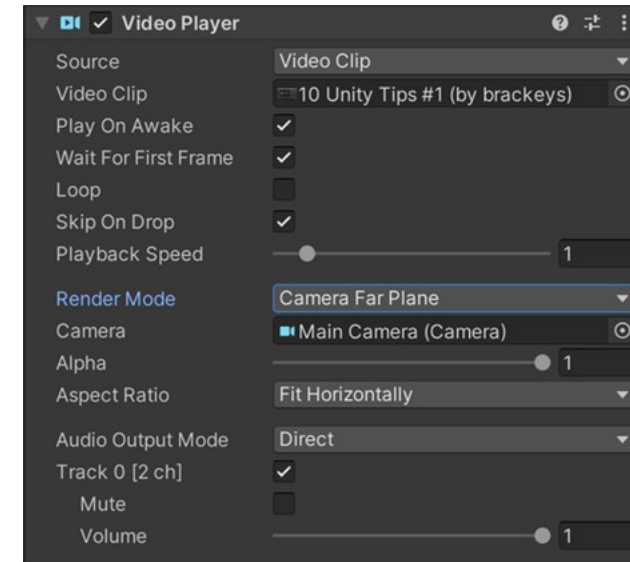


The other Videos are by Japanese animator Karameru. They aren't study material, but are very short.

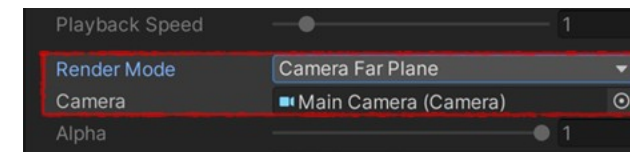
Please don't get too distracted watching The videos!

The Video Player Component

- The component that handles VideoClips is VideoPlayer
Amongst the components we seen so far, it has the most obvious name!
- This component has a lot of attributes that will alter its behavior
Leave your mouse cursor motionless over an attribute name to get an help tooltip
- Experiment with those attributes for a while, most are easy to understand
But there is one that seems to break everything when changed : render mode...
- “Render Mode” is an important attribute : it sets what the “screen” is
By default, the screen is a Camera plane, which is why the video is full screen
- When you change “Render Mode”, other options in the window change
Logical. Parameters can not be the same if the “screen” is a texture or a Camera
- Let’s see how we can use that to play the video in a sub-window...



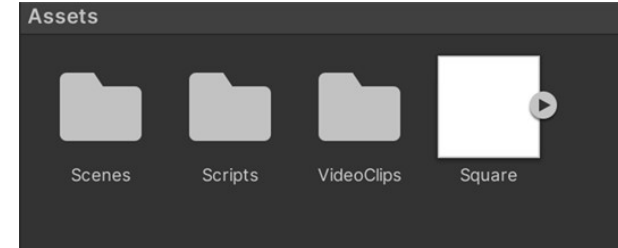
The video Player's attributes as they appear in the inspector



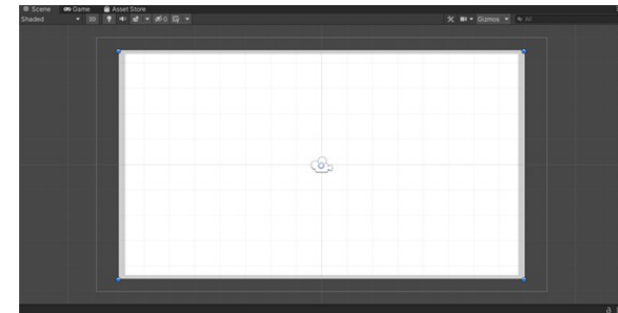
Render mode : a bit harder to understand, but very powerful and useful!

Using Material Override to Play a Video inside a Sprite

- “Material Override” allows the video to replace the material of any renderer
And that includes the SpriteRenderer, which is the one you already know
- Place a SpriteRenderer Component on the same object as Video Player
The SpriteRenderer should be invisible : it has no sprite to render
- Unity can generate Sprites of basic shapes. Use this to create a Square
Here is the path : (Right Click project Tab => Create => 2D => Sprites => Square)
- Set the Square sprite as the SpriteRenderer's Sprite.
Also, confirm that the SpriteRenderer is used as the “Renderer” of VideoPlayer
- Test your app. You should see the video play inside the square
The video will be deformed to fit inside that square, but you will see it
- Deform the square on the scene to give it a roughly 16:9 aspect ratio



The Square Sprite generated by Unity, as it appears in Project tab



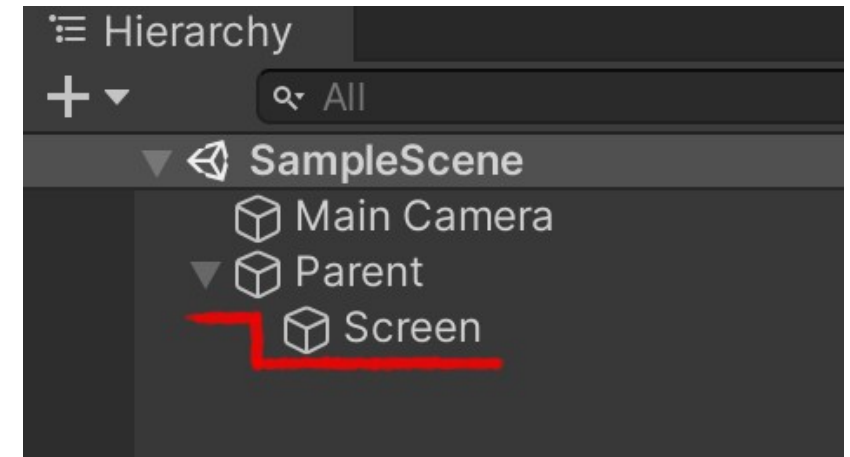
A Square deformed into a 16:9 rectangle, ready to use as screen!

First use-case of a Child GameObject

- We would like our ratio of 16:9 to be perfect, rather than approximate
- Set scale of the GameObject at 16 on X axis and 9 on Y axis (?). You'll get 16:9 that way
...But the screen can only be one size! What if we want a smaller screen?
- Solution: If the Screen object had a parent, scaling it would scale its child, the screen, too!
If 2 GameObjects have a parent-child relation, their transforms affect one another!

Now, it's your turn!

- Create an Empty GameObject (GO), name it "Parent"
- If not done already, rename your Screen GO to "Screen"
- Drag-and-Drop the "Screen" GO on the "Parent" GO
- "Screen" is now a Child of "Parent" !
- Alter the transform of "Parent" and see what happens
- Use this to create a smaller 16:9 screen. Test it.



Once the "Screen" GameObject is a child of "Parent" GameObject, it should look like this

Controlling a Video Player through Code

- Create a Script and add it to the same GameObject as the VideoPlayer
- To use the VideoPlayer class, you need to import the video package with :
 - You can then fetch the reference to the VideoPlayer with GetComponent
 - Of course, you should save it in an attribute, so the reference can be reused any time
- Now that we can access VideoPlayer in the code, we'll code a pause feature!

Now, it's your turn!

- Create a Pause feature on the SpaceBar
- Pushing SpaceBar should toggle between pause and play
- You will need those methods : `VideoPlayer.Play()` , `VideoPlayer.Pause()`
- You will need this attribute : `VideoPlayer.isPaused`
- Use `Input.GetKeyDown()` to detect the first frame a key is down

```
using UnityEngine.Video;
```

You can not use VideoPlayer without using this import at the beginning

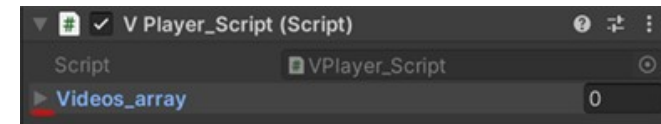
```
// Update is called once per frame
@ Message Unity | 0 références
void Update()
{
    if (
    {
        if (
        {
            //
        }
        else
        {
            //
        }
    }
}
```

The Pause code, with the interesting parts "censored". It's your job to figure it out!

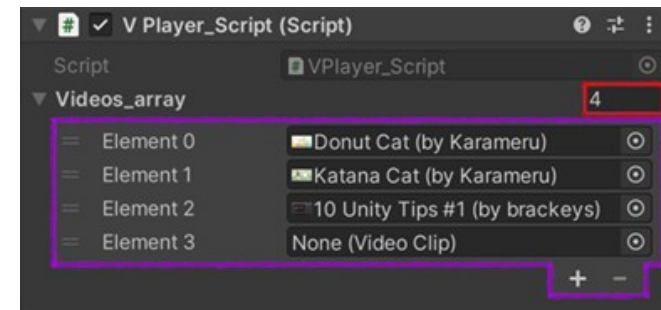
Using an Array as a Public Attribute

- We now want a button that will change the current VideoClip
If the user pushes the "R" key, a random, different clip is loaded and played
- To do that, we need to give references to several video clips to our script
We could use 6 different public attributes. But that's not very efficient...
- Fortunately, Arrays exist in C#, and Unity can display them in the Inspector!
(As long as they are set to public, of course)
- Adding [] after a type will turn it into an array of that type. Like this :

```
public VideoClip[] videos_array;
```
- The Array being public, it can be edited through the Unity GUI
Look at the images on the right to see examples of how that works
- Use the array to store references to at least 4 different videos




Because Arrays can be pretty big, their contents can be hidden in the inspector. The triangle on the left allows you to toggle.




The Number Field is the size of the Array. When you change it, it creates lines in the Content Area, which you can fill with Your VideoClips through drag-and-drop!

Using the Array in the Code

- The way arrays work in C# isn't much different than in most languages
- You can access "entry number i" of an array with array[i], like this :
- A very useful property is Array.Length, which gives the size of the array :
- The "i" number must be an integer. To select a random integer, you can use The method Random.Range(). See example on the side.
- **REMEMBER! : an array's first entry is number 0, not number 1 !**



```
VideoClip myClip = videos_array[1];
```



```
videos_array.Length
```

```
Random.Range(0, 4);
```

This example chooses an Integer number between 0 (included) and 4 (excluded)

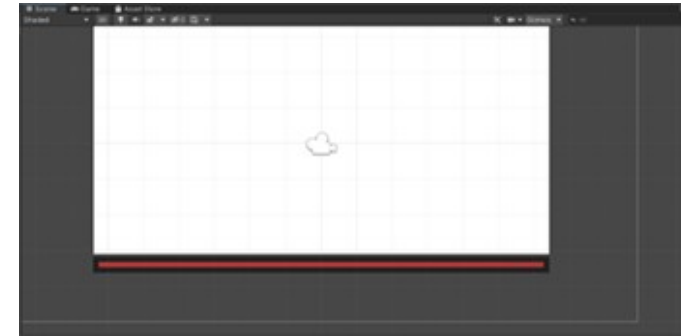
=> This means the result can be 0,1,2 or 3

Now, it's your turn!

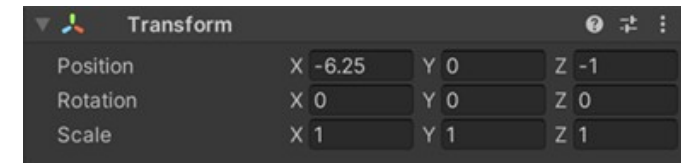
- Edit the script so that when the user presses "R", another clip is loaded and played
- The loaded clip has to be different that the one that was playing before
- Either keep the "paused" status when loading a new clip, or have a new clip always start playing
- The script should adapt automatically if I edit the array by adding or removing videos
- Even better : separate this random selection process in a custom method

Creating a Video Position Bar

- As a final Challenge, we are going to create a progress bar for our video!
- Begin by creating the graphic part with the square sprite :
 - 1) Create a black rectangle to act as the background
 - 2) Create a colored rectangle inside to act as the progress indicator
- `VideoPlayer.frame` gives you the number of the frame displayed
- `VideoPlayer.frameCount` returns the total number of frames in the clip that is currently playing. *(That number can be quite big for long videos!)*
- Because a video can have a LOT of frames, those are double type numbers instead of float type numbers. **This can matter when writing code.**



Try to create something that looks like this!



Make sure the red part is in front by using the Z axis. (negative numbers are possible)

Now, it's your turn!

- What is the math that will give you the percentage of advancement in the video?
- Try to code it. What problem is preventing you from completing this code?

Type Casting and Scaling with Code

- To scale an element through code, you have to use `transform.localScale`
- Just like `transform.position`, this is a `Vector3` that needs to be modified “all at once”
The “x”, “y” and “z” can be READ individually. But they can not be MODIFIED individually
- Considering the math we figured out before, the code to scale our progress bar should look like this :

```
progressBar_content.transform.localScale = new Vector3( ref_vplayer.frame/ref_vplayer.frameCount , 1, 0);
```

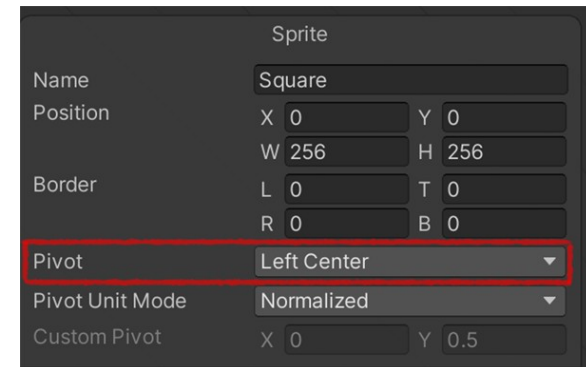
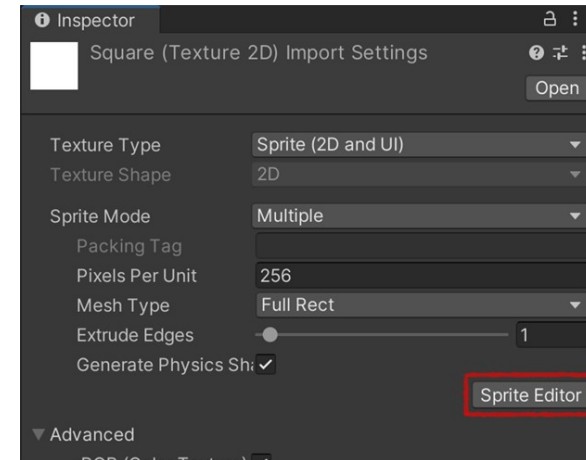
- But there is an error! Why? ... Because `Vector3` expects float numbers, and we are providing a “double”
Since we are making a division between two “double” numbers, the result is a “double” number as well
- The solution is to do a type cast to float. This is done by adding `(float)` in front of the number to convert :

```
progressBar_content.transform.localScale = new Vector3( (float)ref_vplayer.frame/ref_vplayer.frameCount , 1, 0);
```

- Now that you have this info, you should be able to make the bar scale according to video progress!
...But is that scaling on the x axis behaving entirely as we expected?

The Pivot of a Sprite

- The problem remaining is that scaling occurs from the **CENTER** of our object!
This causes the bar to expand from the center, rather than from the left!
- We can change that by editing the “pivot” of the Square sprite.
The pivot is the point that is used as origin for math transforms, such as scaling
- Select the square sprite **IN THE PROJECTS TAB**.
The inspector will then show its import properties. There, click “Sprite Editor”
- Look for the “pivot” option in the sub-window, and set it to “left center”
Don't forget to click “Apply” at the top right of the sprite editor, before closing it!
- Note that when you do that, all our Sprites change position on the scene!
That's because all graphics here use the “square sprite”, and since its origin point has changed, they will appear at different positions even though their coordinates are the same
- Correct the position of the sprites. Test. The bar should scale as expected now!

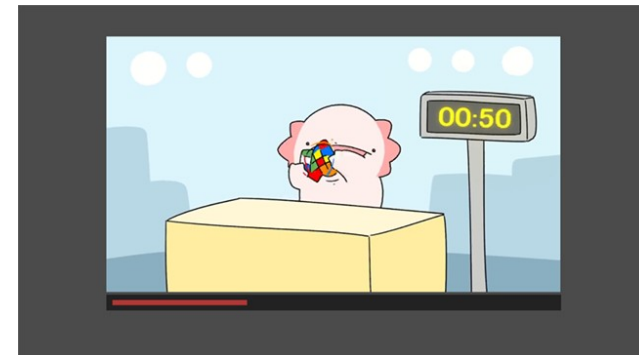
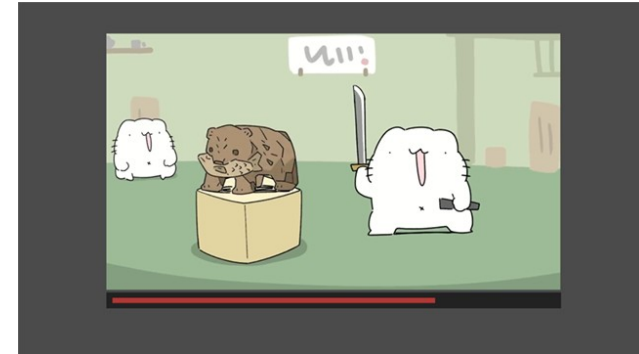


Changing the “Position” of a video

- `VideoPlayer.frame`, which we used for our progress bar, isn't read-only
Meaning that if we set its value to a number, we'll “teleport” to the frame with that number!
- As usual in computer code, the first frame **isn't** frame n°1. It's frame n°0!

Now, it's your turn!

- Create a public method that will rewind the video by changing its frame
 - Add a keyboard shortcut that triggers this rewind method (*backspace key*)
-
- Note that, if we were able to detect clicks, we would use the same principle to allow the user to move inside the video with the progress bar!
We could set the video to any frame, not just n°0. Allowing us to move anywhere
 - Which points out a big problem of our player : it has no mouse controls!
This is why, next time, we are going to talk about mouse interaction and UI!

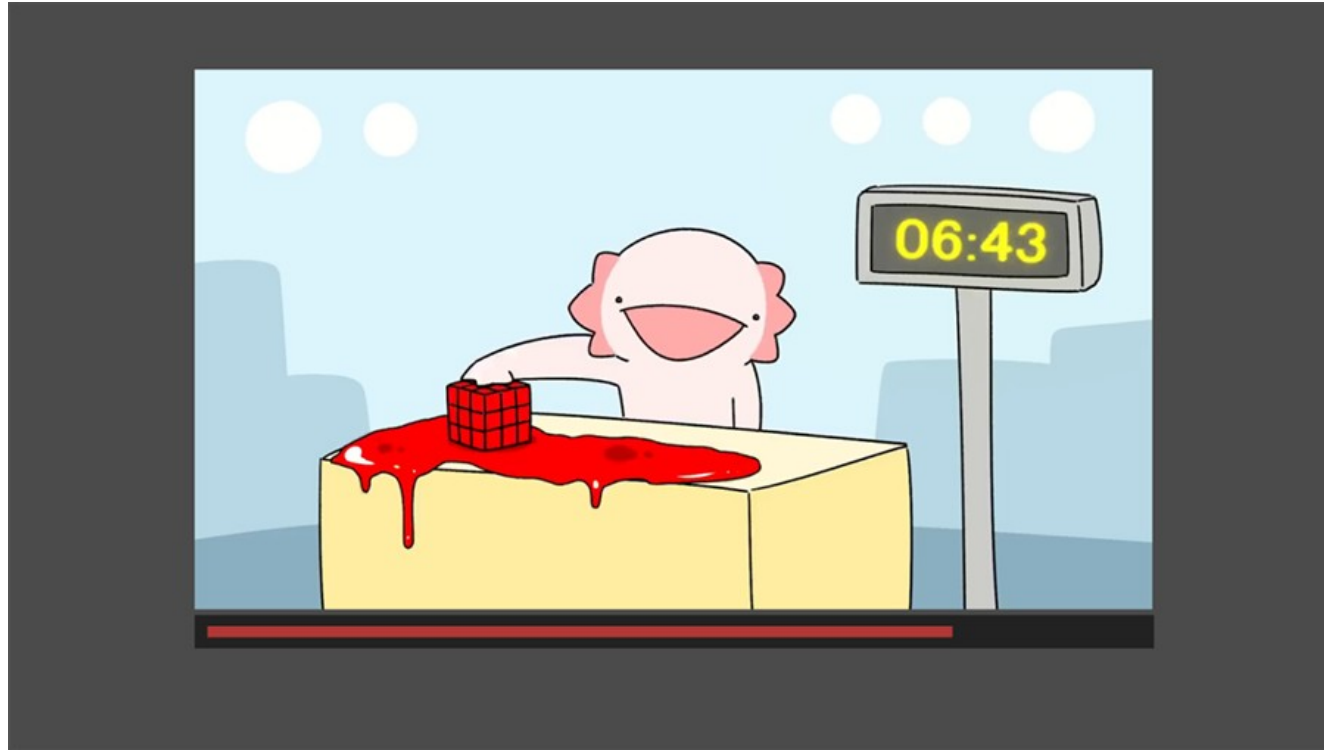


Our Player is starting to have some nice features. But, because there are no buttons, it's hard to use!

Next time we'll look into solving that problem!

Now, you are able to use Video!

That's another medium added to your toolbox!



(Of course, there is much more to learn about video in Unity. But this basic knowledge will definitely allow you to use some video in your final project!)

Next Time...



UI Canvas : Buttons and Mouse Interaction

So we can finally click on things!