

LP2B – Multimedia: Digital Representation

- LP2B Tutorial Class n°2 -

Instantiating, RNG, Text, and Scenes

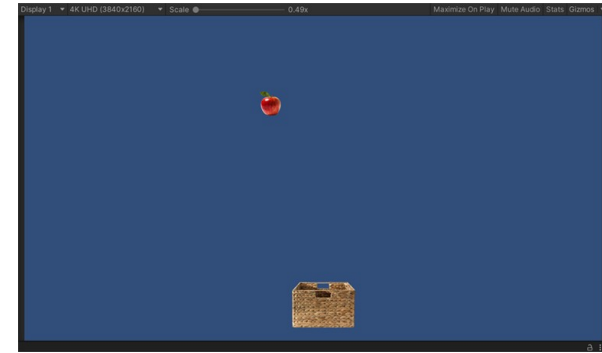
Key concepts that appear in most apps

This class will give you an overview of :

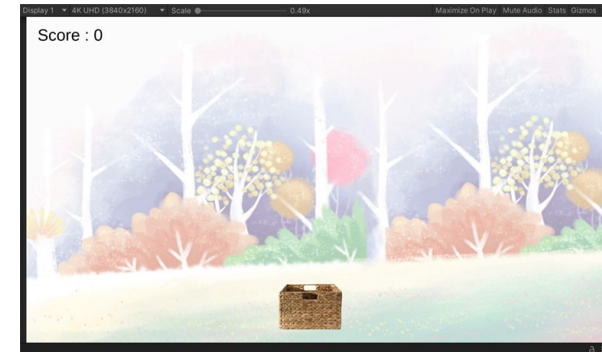
- *Setting the Camera Ratio* -
- *Saving a GameObject as a Prefab* -
- *Instantiating new copies of a prefab* -
- *Random Number Generation (RNG)* -
- *Using TextMeshPro text objects* -
- *Depth on the z axis ("front" and "back")* -
- *Using several scenes* -

Lets go back to our Basket and Apples...

- We had a falling apple and a moving basket. But not a game yet!
- Open Unity Hub and use it to re-open the project from last time
The hub is pretty convenient, isn't it? Its pros usually overweight its few cons
- We are going to create a proper gameplay loop, and a title screen
Then, it will be a super simple, basic game. But a game nonetheless!
- The next step towards getting a proper game is to "Instantiate" apples
"Instantiate" means creating new instances. New "copies" of the apple
- To copy the Apple, we will have to save it as a prefab
We also are going to make some minor edits to its script, as warmup!
- Let's go!



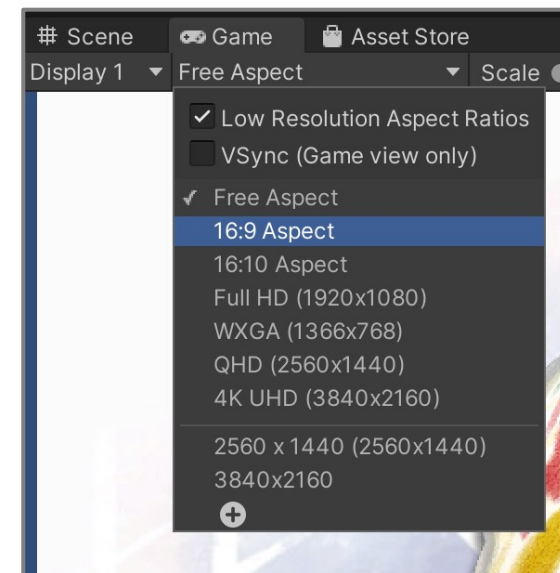
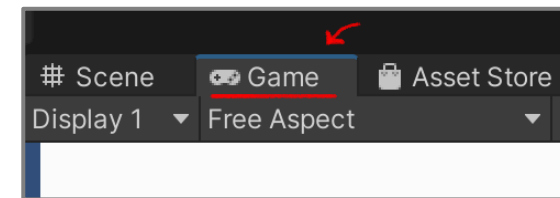
The project in the state we left it in.




What we want to have at the end of this session

Ensuring our Camera has a 16:9 ratio

- By default, Unity sets the camera to the same ratio as the scene preview tab
Advantage : it's WYSIWYG. Inconvenient : the screen of our device won't be like that
- To set a specific aspect ratio, we have to go to the "Game" tab for a moment
The "Game" Tab is the one Unity switches to when testing. But it has other uses
- The "Game" Tab is Right Next to the "Scene" tab. They look very similar.
Scene is the editor, and Game a preview of how it will look as an app
- Once in the Game tab, you have access to the ratio option
By default, it should be on "free aspect", which means "same ratio as the Scene Tab"
- Click on it to see the options. Change to 16:9.
- Your camera will now properly be forced to a ratio adapted to most screens
It is possible to make an app that properly adapts to all ratios dynamically. But not yet!
- You might have to alter your scene a bit because of this change



A few things about C#

- We are going to do some more advanced scripting soon, so I thought I should clarify a few things. *(some of those are reminders)*
- “If - else” statements, the most basic building blocks of code, are obviously in C#
Accolades are used to delimit the block of code if there are several instructions
- All the usual loops are available : “for”, “while”, and “do...while”
Their syntax is similar to other C languages (and to most existing languages in the world)
- You have a basic example of all those loops to the side : 
- Remember that a standard loop will ENTIRELY EXECUTE WITHIN ONE FRAME
Complex loops that execute too many times will make your app lag, as they will occur within 1 frame!
- To have a process happen over several frames, you have to use “Update()”
...and be a bit more clever about it!
Update() can be seen as a loop itself in a way, since it executes each frame
- There are other ways to handle this issue of real-time. Not just Update()
...but they're not suited for beginners, so you'll have to make do for now

```
public void ExampleMethod()
{
    //Do 10 coin toss with a "while" loop
    int i = 0;
    while ( i < 10)
    {
        if (Random.value > 0.5f)
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
        else
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }

        i++;
    }

    //Do 10 coin toss with a "for" loop
    for (int j=0; j<10; j++)
    {
        if (Random.value > 0.5f)
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
        else
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
    }

    //Do 10 coin toss with a "do while" loop
    int k = 0;
    do
    {
        if (Random.value > 0.5f)
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }
        else
        {
            Debug.Log("Coin toss number n°" + i + "is tails");
        }

        k++;
    } while (k <= 10)
}
```

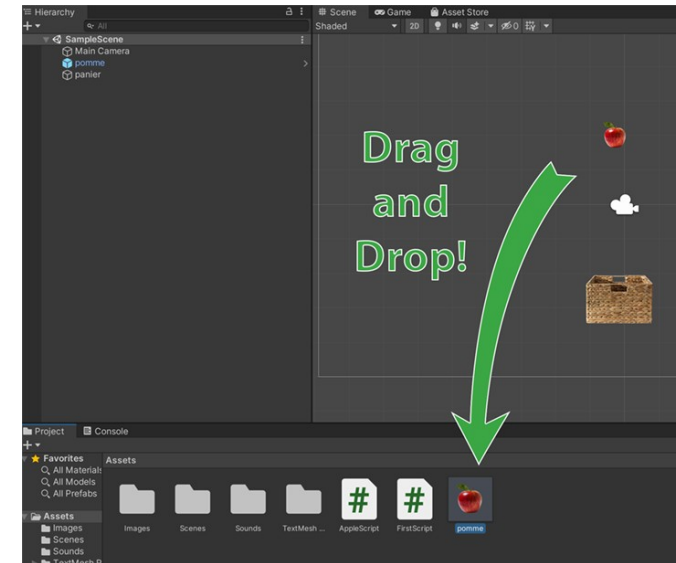
Finishing the Apple and saving it as Prefab

A little thing to finish first :

- Modify the script of the apple so it self-destructs when colliding the basket

Reminder : the code line to destroy the GameObject is : **Destroy(gameObject);**

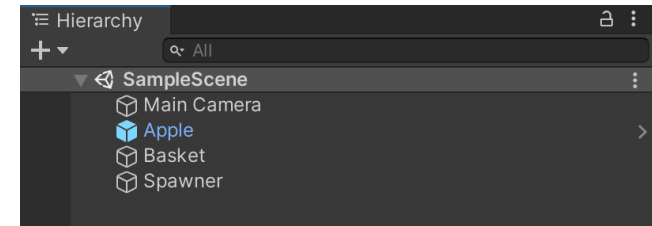
- Before continuing, test the feature requested above by launching the app
- When it works, our apple has everything it needs. We can copy it now!
- Because we want to create many copies of the apple, we must save it as a “prefab”
A prefab is a term used to designate GameObjects that have been saved as assets
- To do so, simply drag your apple GameObject from the scene into the project tab
- To avoid ambiguity, rename the prefab to “Apple_prefab” after creating it
(Right click the prefab in the projects tab => Rename) OR (Select it and press F2 on keyboard)
- Once the Prefab is saved, you can delete the original apple that is on the scene



*Drag and Dropping from the scene to the project tab will save our GameObject as a prefab.
Note that instances of prefabs appear in blue in the Hierarchy Tab.
(the tab with the list of objects on the scene)*

Making a “Spawner” from an invisible object

- Create an empty game object, with no visuals. It will be our spawner
(*right click in hierarchy tab => Create Empty*). Hierarchy tab is the “GameObject list” on the left
- Rename this GameObject. Create a new script for it, and attach them together
Suggested names : “Spawner” for the GameObject, “SpawnerScript” for the script
- This script will need a reference to the Apple prefab, so it can create copies
- To achieve this, create a public attribute in the script, with type “GameObject”
There is no “Prefab” type in code. Prefabs are GameObjects. They are just saved outside the scene
- Drag and Drop your prefab into the attribute field you just created
Because our attribute is public, it can be seen in unity Editor. And because it is GameObject type, it can receive the prefab as a valid value. (*values are transmitted by reference*)



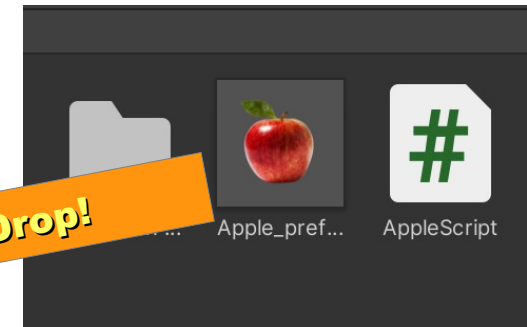
The Hierarchy tab, with the Spawner GameObject already added.

```
public GameObject apple_prefab;
```

Declaration of a “GameObject” type variable, which will allow our script to access a prefab



The apple_prefab attribute of the script, as it appears in the unity interface.
This field was generated automatically because the attribute is public.



The Prefab in our Project Tab

Instantiating copies of our Prefab

- Put the following code in the “Start” method of “SpawnerScript”
The “Start” method only runs once, just before the object becomes active

- The code that allows us to create a copy of a GameObject (prefab) is :

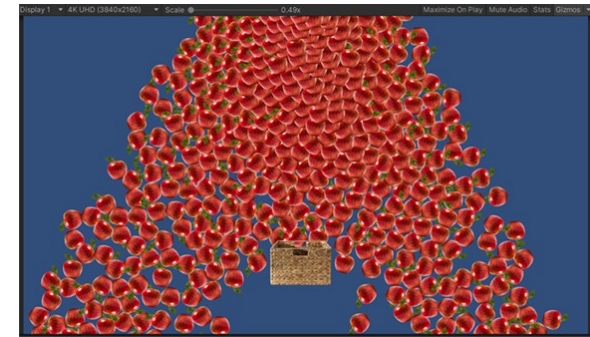
```
GameObject newApple = Instantiate(apple_prefab);
```

- Note that we use a local variable “newApple”, to store the copy
If we didn't do that, the object would be created, but we would **lose the reference**, making it **impossible to customize** the properties of that object!

- Thanks to the variable, we can edit the properties of the new object before it even appears on screen. For instance, position :

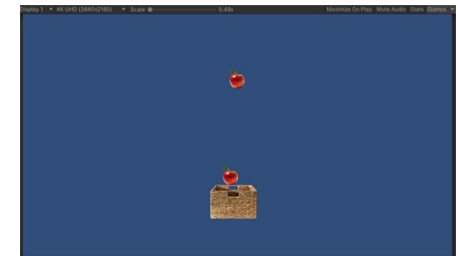
```
newApple.transform.position = new Vector3( 0 , 6.0f , 0 );
```

- Run a test. One apple should spawn at the position (0,6,0), and then fall
If you have time, mess with those numbers for a while and see how it affects the result!



If you put the code in “Update”, one apple will be created every frame. Resulting in this!

(OK. I admit. It's kinda funny to see once...)



*If you see 2 apples, this means you forgot to remove the apple on the Scene!
Our code should only generate 1 right now!*

Improving our Spawner with a Timer

- What we truly want is our spawner to trigger every 1.5 seconds
Neither every frame nor only once will do. We want a timed loop!

Now, it's your turn!

- You actually know everything you need to create this timer!

Reminder 1: `Update()` runs once per frame. `Time.deltaTime` gives the real time since the last frame

Reminder 2: Attributes and their value **persist** between frames. Like a timer would have to!

Reminder 3: “If” statements exist in C#. They could **prevent Update() from doing something every time**

- Try to code this timer yourself in “SpawnerScript”
- We'll correct your attempt together later!
- The image on the right has some extra clues!

```
Script Unity | 0 références
public class SpawnerScript : MonoBehaviour
{
    protected float timer = 1.5f;
    public GameObject apple_prefab;

    // Start is called before the first frame update
    Message Unity | 0 références
    void Start()
    {
    }

    // Update is called once per frame
    Message Unity | 0 références
    void Update()
    {
        //TO DO : decrease the timer with real time

        if ( timer <= 0 )
        {
            //TO DO : Spawn the apple here
        }
    }
}
```

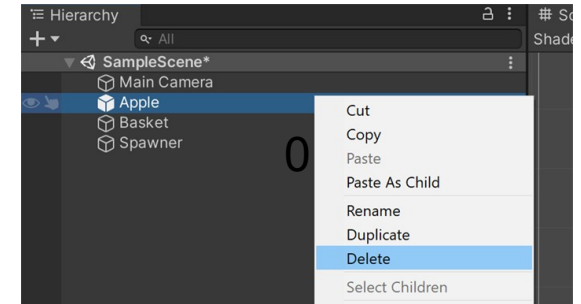
- When we're done, 1 problem left : all apples spawn at the same position
Time to introduce some randomness into this, and our spawner will be finished!

RNG Basics in Unity/C#

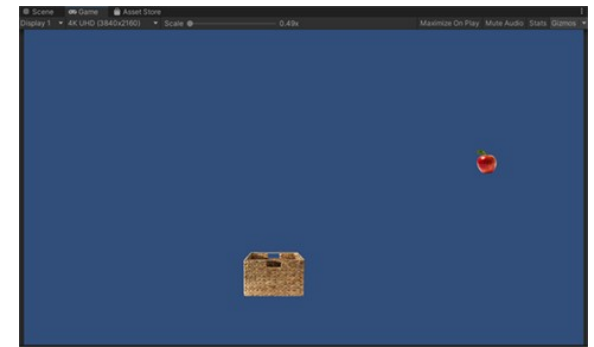
- “RNG” means “random number generation”. It's very useful for apps.
Computers are NOT random, but engineers have found clever ways of doing RNG anyways
- Random.value generates a random float number between 0 and 1
By multiplying that by X, we can get a random float between 0 and X!

Now, it's your turn!

- In the spawner, modify the Vector3 assign as position so its x value is randomized between -8.5 and 8.5
- You need to do some math for that. A local variable can help!
- Remember that in C#, you have to add “f” at the end of float numbers, or the compiler will misinterpret them as “double” type numbers!



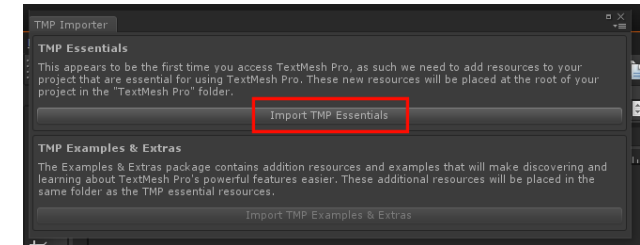
*We no longer need the apple that is on the scene.
if you still have it there, delete it now.
(right click on it => Delete)*



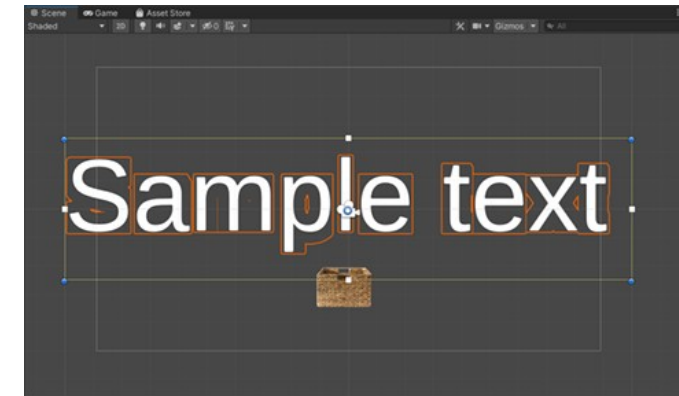
*Once you're done with this part,
You should have an apple every 1.5 seconds, but
with a randomized horizontal position each time!*

How to Create a Text

- All that is left : display the score as text on screen
Score is saved within "FirstScript", which is attached to the basket. But we can't see it!
- There are several ways to handle text in Unity. We'll only see one now
We are going to use a "TextMeshPro" text. It is a 3D object, but it works in 2D too
- To Create a GameObject with TextMeshPro (TMP) Component :
(Right click in the hierarchy => 3D Object => Text - textMeshPro)"
- The first time you create a TMP, a special window will appear
- In this Window, choose "Import TMP essentials", but not the other
This will add some things in your asset folder and your package folder. This is fine!
- Your text should appear on the Scene, ready to be edited



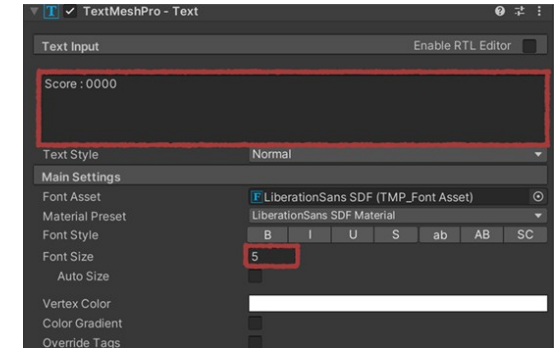
*The window that requests your permission
To import extra stuff required for TextMeshPro.
You only need "TMP essentials" (top option)*



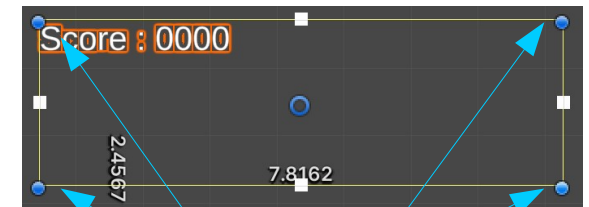
*By default, the text can be pretty big.
No worries, we can fix that afterwards!*

Setting up our Text

- Select the text in Hierarchy tab. Rename it to "Score text"
Renaming works the same as usual : *(right click => rename)* OR *(select => F2)*
- In the Inspector, the Text Mesh pro component can be seen
It has a LOT of parameters. But don't worry, for now we only care about a few
- Change the font size to make the text itself smaller
- You can change the default text by typing in the big box at the top
- Then, edit the size and position of the box containing the text
MAKE SURE YOU USE THE BLUE PINS IN THE CORNERS FOR THAT
The other pins are for editing padding and margins visually, which is different!
- Place the text in an upper corner, where it makes sense
Next, we will connect this text field to the code so it can be changed at runtime



The TextMeshPro component, with the attributes we care about highlighted



Use the Blue Pins to edit the box!



This "T" gizmo tells us this is indeed a dynamic, editable text. It will not appear when the app runs

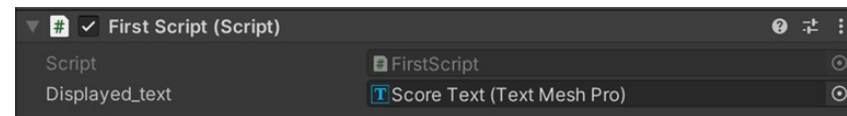
Giving the reference of our Text to the code

- Just like with the prefab, we need a public attribute
That way we will be able to transfer a reference to the TMPro component to the code!
- To use TextMeshPro in the “FirstScript” code, you must import it
To do so, add “`using TMPro;`” at the very beginning of the “FirstScript” file
- Once this is done, create a public attribute of type “TextMeshPro”
You can simply copy what's written on the side. Auto-completion should help you!
- Save your file and return to unity Editor. Select the basket GameObject.
You can see that a “Displayed_text” field is now visible. Awaiting a reference
- Drag and Drop the text field from the scene into this field
You can either pick it up from hierarchy, or directly from the scene itself
- When you're done, the Script should look like this in the inspector :

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

Script Unity | 0 références
public class FirstScript : MonoBehaviour
{
    //-----
    // ATTRIBUTES
    //-----
    public TextMeshPro displayed_text;
    protected int score = 0;
}
```

The “using” and the public parameter are both highlighted in this code snippet



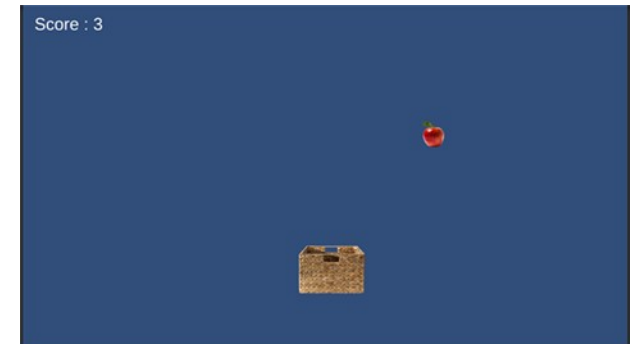
Modifying the Text through Code

- The attribute “displayed_text” stores a reference to the TMPPro on the scene
This means that any modification done to “displayed_text” will, in fact, affect the text on the scene!
- To change the text of a TMPPro to a different string, use the “SetText” method :

```
displayed_text.SetText("My String here");
```

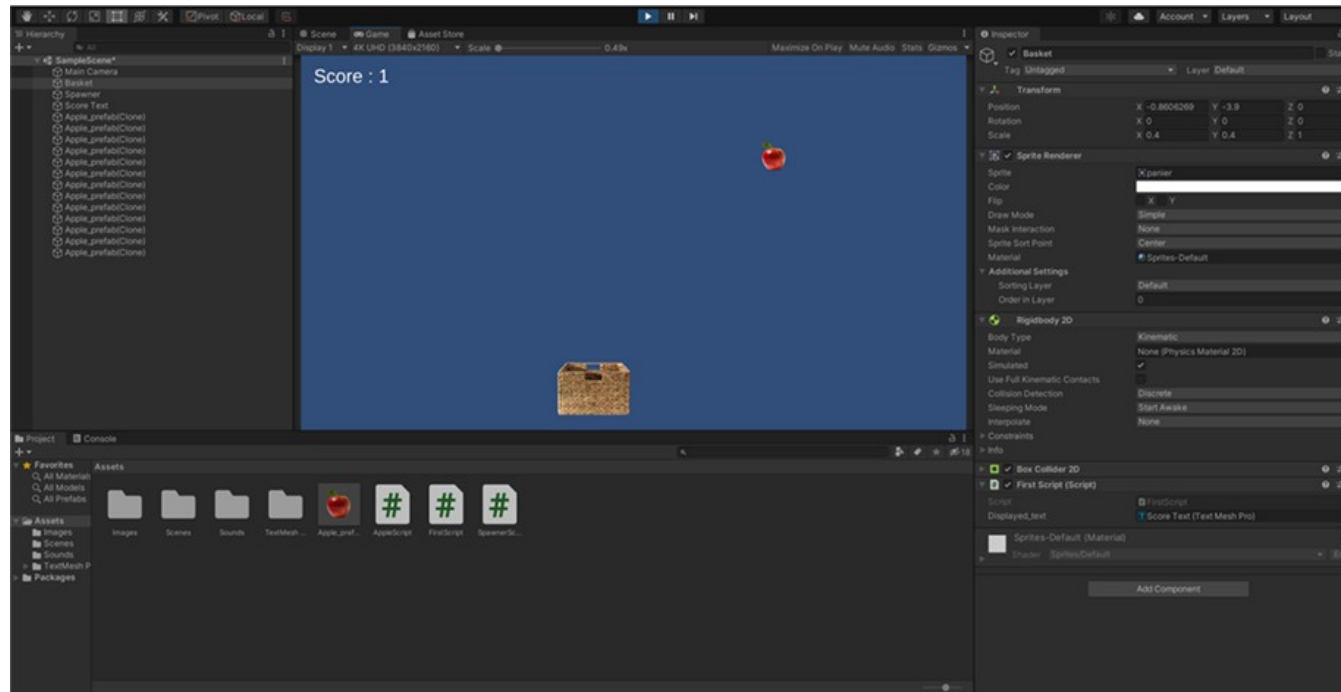
Now, it's your turn!

- Modify the “FirstScript” script so the score is displayed in the TMPPro
- We want the syntax to be “Score : X”. Not a number alone!
- You can concatenate a text and an int variable with the “+” sign.
- You will need to update the text every time the user gains a point



A preview of our goal!

Yay! Our game is playable!



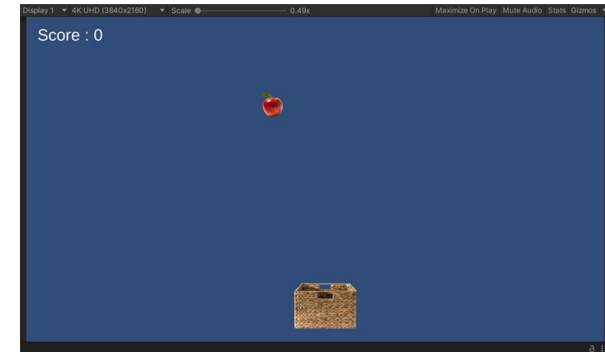
OK... it doesn't look very good... It's super basic... But still!
Congratulations for completing your first Unity app ever!

Let's make this Game even Better!

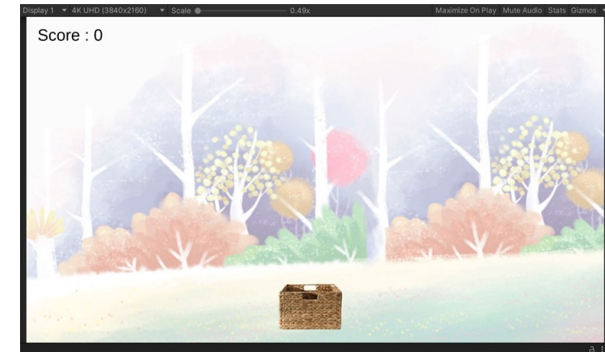
- We are not done with this project. We'll improve it further!
- We will add a background to the game. It should make it much prettier!
The background is an image, so we are going to do that in only one slide!
- Take the background image and put it on your Scene as a GameObject.
Easy, right? We just are adding an image! What could possibly go wrong?
- Change the color of the text to black so it is properly readable
- Everything should be good! Let's test the app!
- Wait? We can no longer see some of our objects! What's going on?
The hierarchy list confirms that apples are here, and yet we are not seeing them

Now, it's your turn!

- Can you guess what is going on here?



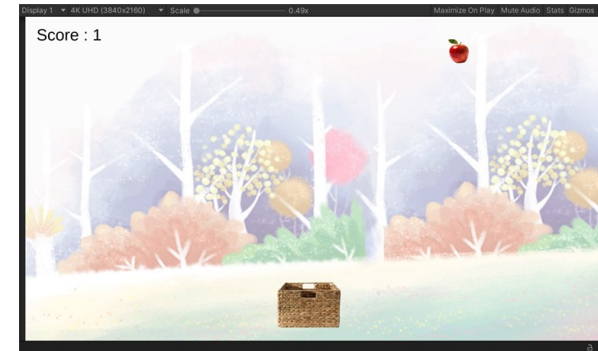
*The project in the state we left it in.
(without any of the fancy "extra objectives")*



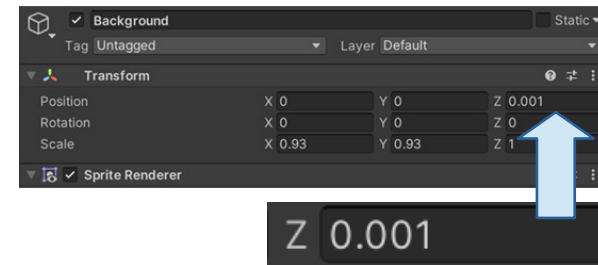
*With the background added, I can no longer
see the apples! What is going on?*

A Matter of Depth

- Our apples are here, but are actually BEHIND the background!
Even though this is a 2D project, depth exists! Things can be behind or in front
- There are several ways to handle this matter : z position, sorting layers...
Those techniques can even be combined together to create specific effects
- Using Z position to control depth is the easiest way :
the higher the z position is, the more "behind" something is
- Change the position of the background on the z axis. Problem solved!
You can do that directly through the inspector, without any code
- Keeping track of the z values is an important part of this method
It can get hard on big projects with many objects. That's why other methods exist
- Our projects being small so far, we do not need anything else for now



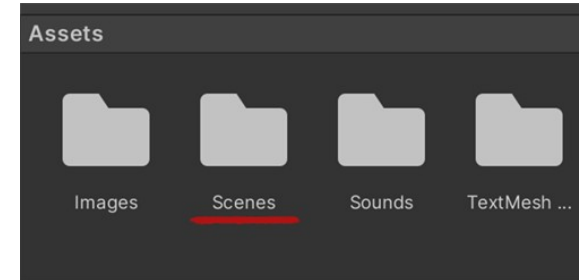
The apples have a position of 0 on the Z axis. So if I put the background at any value above 0 on the same axis, it will be behind. Like here!



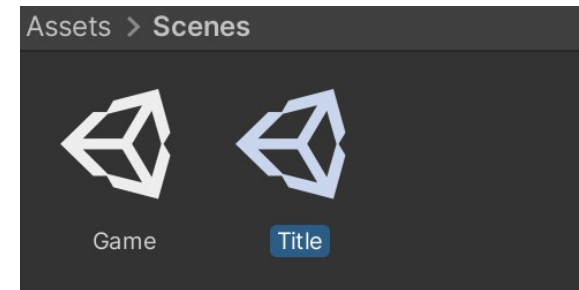
Since 2D sprites are flat, ANY difference in Z position is enough to make an object appear behind or in front. However how small!

Creating a New Scene

- To refine our game further, we are going to add a title screen
It's pretty important to have a "loop" in a game. A title will help with that
- The best way to achieve this is to have our Project use **Multiple Scenes**
In theory, everything could be one only one scene. But this makes things confusing!
- A Scene is a type of Asset. A "Scenes" folder was created with our Project!
Inside, you can see "DefaultScene", the scene we have been working on until now!
- To create a Scene : *(right click in project tab => Create => Scene)*
- Rename that Scene to "Title". Double click to open it
You should then see it : a new, default scene. With just a Camera
- You can also rename the original Scene to "Game"



The Scenes folder, among the others we created to sort our Assets



The Contents of the Scenes Folder after our changes!

Setting up the Title Screen

- Place the background and the image containing the title on the scene

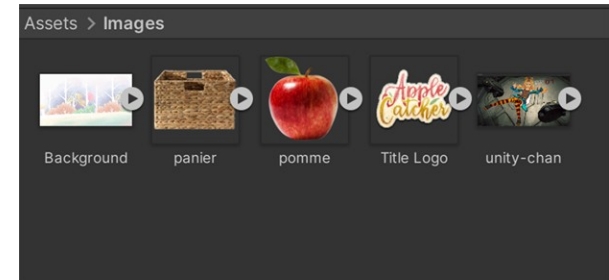
Make sure the z axis is set correctly so the background is indeed behind

- Place a TextMeshPro that says "press any key to begin"

It won't look super pretty now. But don't worry we'll improve that later

Now, it's your turn!

- Set up the Graphics as mentioned above
- Create a new script and call it "Title_Script"
- Attach it to the Title Image GameObject.
- We'll talk about the code to put inside on the next slide!



*Assets are shared between scenes,
So you will still have access to all your
images, scripts, etc...*



*Try to create something that
looks like this!*

The Scene Loading Coroutine

- Changing Scene requires a method in the *“UnityEngine.SceneManagement”* package
So you need to add *“using UnityEngine.SceneManagement;”* at the beginning of the script
- The loading of another scene takes place over several frames... but not in Update()
Remember how I mentioned there were other ways to manage that? This uses one of them
- Loading uses a special type of function called **Coroutines** (*which use IEnumerator type*)
- Understanding Coroutines is NOT part of this class. So we'll use this one “as is”!

Coroutines use return type *“IEnumerator”* instead of a normal variable type or *“void”* (no return)

```
IEnumerator LoadScene_Game()  
{  
    AsyncOperation asyncLoad = SceneManager.LoadSceneAsync("Game");  
  
    // Wait until the asynchronous scene fully loads  
    while (!asyncLoad.isDone)  
    {  
        yield return null;  
    }  
}
```

The String used as parameter here should match the name of the Scene to load

Launching the Coroutine

- Coroutines can not be launched like normal methods
- You need to use "StartCoroutine(**coroutine()**)" to do so
Coroutine() being the name of the coroutine. And yes, you have to put the parenthesis

Now, it's your turn!

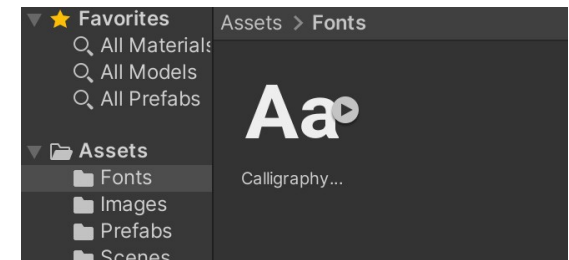
- Code "Title_Script" in order for the game scene to load when the user presses any key
- To detect when "any key" is pressed, use **Input.anyKeyDown**
- Since you have the coroutine, this should be easy!



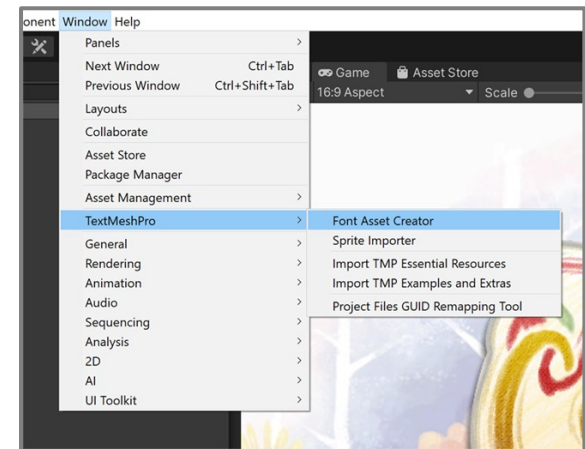
- Good news! Once this works as expected, we have completed our basic objectives!

Using a Custom Font with TextMeshPro (1/3)

- One last touch : we are going to customize the fonts of our texts!
- Add the provided font file to your project as an Asset
So far, it works just like any other asset type. No worries yet!
- But fonts require an extra step for use :
The font needs to be converted to a material with a texture
- But wait... Aren't Material and textures notions that are used in 3D?
They are. But remember? TMPs are "3D objects that work in 2D too!"
- Fortunately, an automatic converter comes with the TMPro package!
- To open it : (*Window => TextMeshPro => Font Asset Creator*)



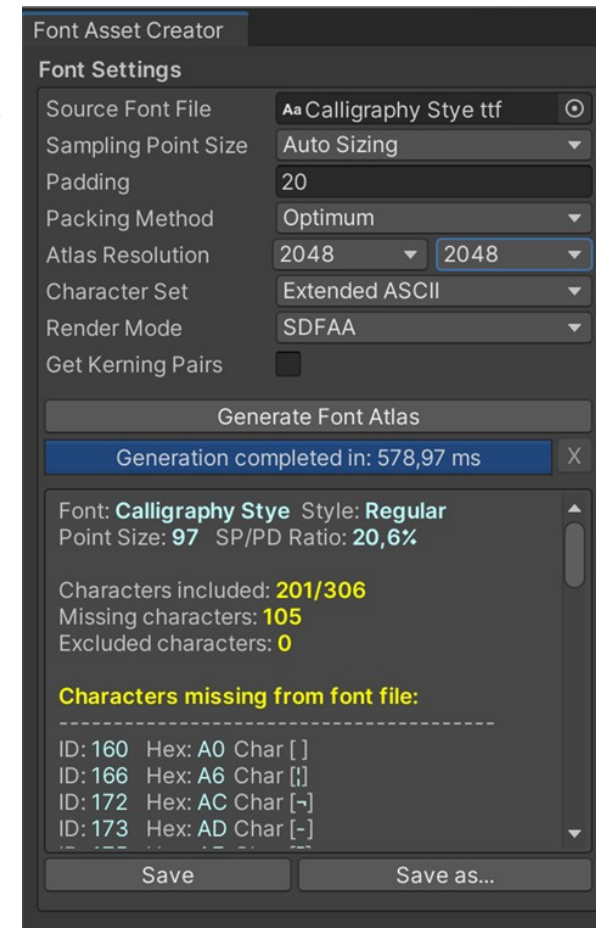
A Font asset as it appears in the Project tab (and, of course, I sorted it in a separate folder!)



How to access the Font Asset creator

Using a Custom Font with TextMeshPro (2/3)

- On the side here, you can see how the window looks AFTER we're done
- First step : drag and drop our font asset into "Source Font File"
- There are a lot of options here, but we'll use a setting that works nicely most of the time. You can learn later how to optimize.
- Copy the settings I used on the screenshot. They are a good "go to" setting
Most important are the "character set" and "atlas resolution" fields
- Click "Generate Font Atlas". The colored text below should now appear and you will get a preview of the generated texture (*hidden here*)
- Once you can see the texture. Make sure the "point size" indicated in the results is fairly high (*more than 60 is a good frame of reference*)
- Then, click "save" to save your generated texture+material.
- You can choose where to save it. Saving it in the "Fonts" folder makes the most sense!

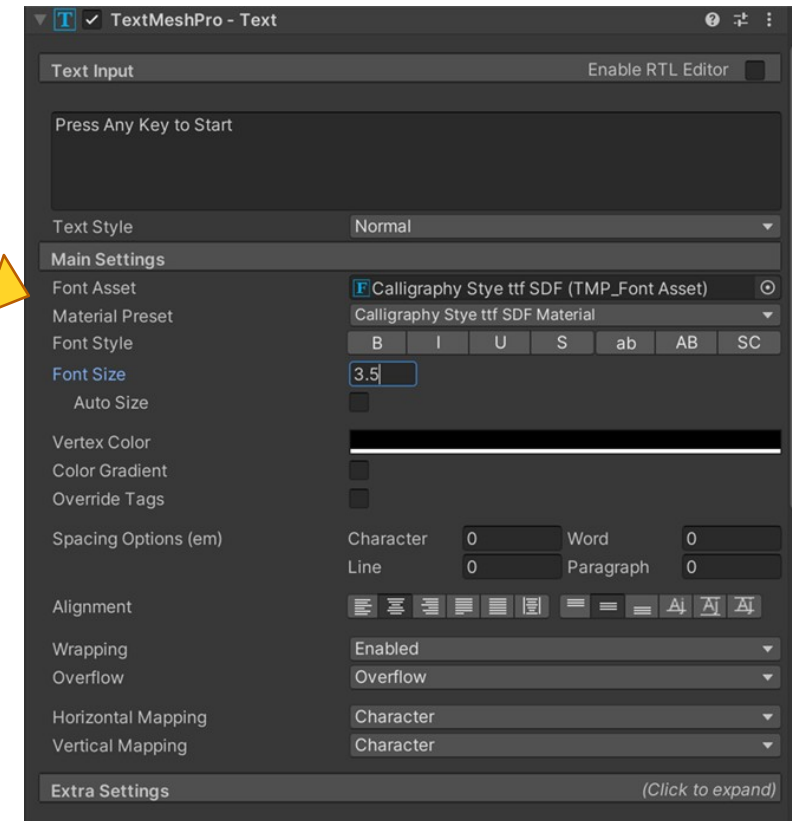


Using a Custom Font with TextMeshPro (3/3)

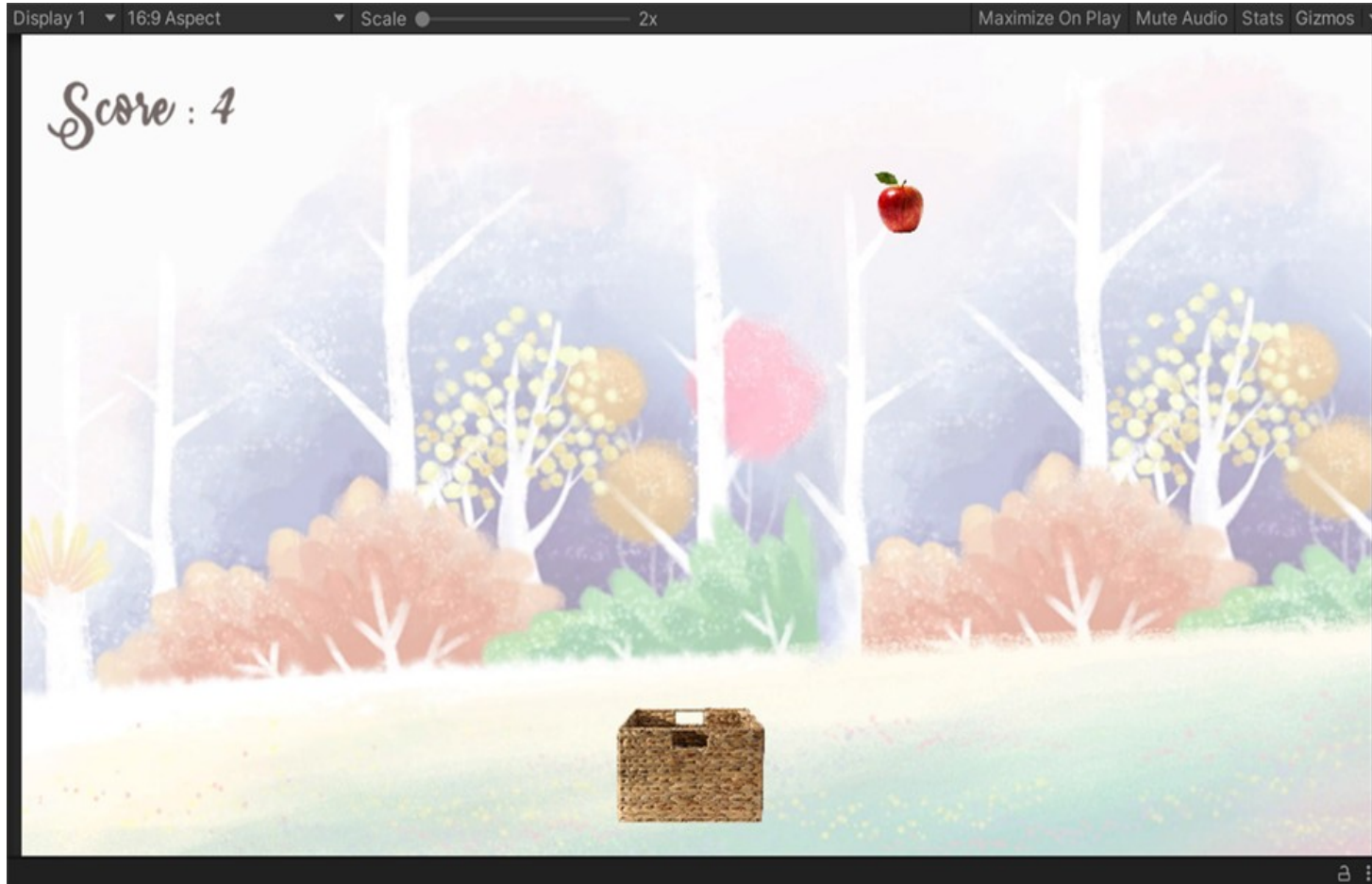
- Now that the material has been generated, we can finally use our font!
- Select a text GameObject to access its TextMeshPro component in Inspector
- Drag and Drop the FONT TEXTURE in the "Font Asset" field
The original font will not work. You NEED the converted texture!
- You might need to adjust the size and color after the change
The new font doesn't have the same base size than the old one
- Your text should now look like this :



- Do the same thing to the score counter in the other scene, and we're done!



Tadah! Our First Unity Minigame!



This time, we have something we can call a game!

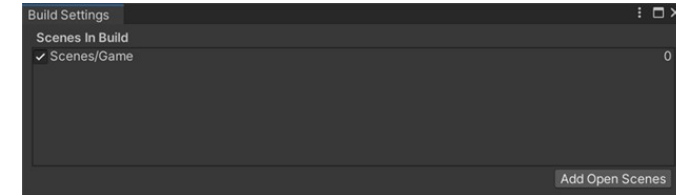
It has a title screen, it counts and displays score, and nothing is obviously missing!

Of course there is still room for improvement. But this is a significant milestone

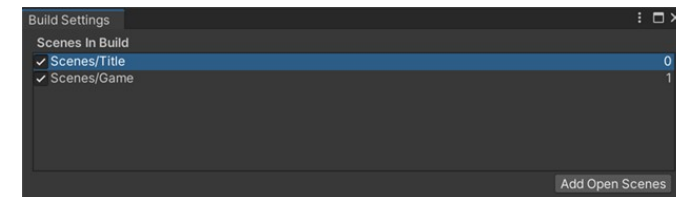
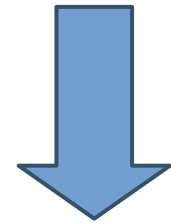
Let's make a build of this game so we can play it outside Unity!

Let's make a Build of our Game! (1/2)

- To celebrate our milestone, let's make a build of our game!
- Go to the build settings menu : *(File => Build settings)*
- You can see at the top what scenes are part of the build : we are missing one!
Only the "Game" scene should be part of the build, because it's the first, default one
- Open the "Title" Scene, then click "Add Open Scenes" to add it to the project
You can open another scene without closing the build settings subwindow
- We want the Title scene to come up first, so it has to be "number 0"
You can change the order of the scenes by drag and dropping within the list
- Remark : Since Scenes can be ignored, **it is therefore possible to use ignored scenes as "Test Scenes" to experiment into**. They won't affect the app and project!
Extremely useful when tackling bigger and more complex scripting!
- There are a lot of other options for building, but the default values should work fine



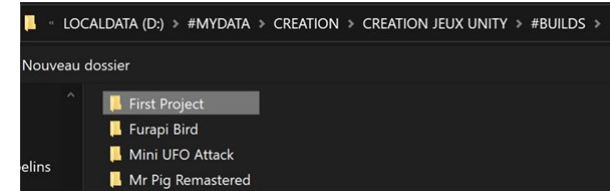
How the "Scenes in Build" should look when you open the window



How it should look once you're done

Let's make a Build of our Game! (2/2)

- Once the scenes are set up correctly, click “Build”
- Unity will then ask in what folder the build should be created
By default, the file browser will be inside the project. **DO NOT BUILD THERE!**
- To avoid confusion, **I STRONGLY RECOMMEND YOU CREATE A SEPARATE FOLDER**
 - 1) Create a “Builds” folder somewhere on your hard drive (*NOT in the unity project*)
 - 2) Inside, create another “First Project” folder. Choose THAT ONE as build location
- After you set up the destination folder, the game will be built
- When it's done, Unity will automatically open the destination folder, showing the result
If you built for Windows, you get an exe file with some data folders and files around
- Launch the exe file, and enjoy your game in glorious full screen!
Since we didn't implement a way to quit the game, you will have to “alt-F4” out of it!



Me, creating a “First Project” folder inside my dedicated “Builds” folder. Doing so, the build files won't get mixed with the project files

Nom	Modifié le	Type	Taille
First Project_Data	18/04/2021 09:44	Dossier de fichiers	
MonoBleedingEdge	18/04/2021 09:44	Dossier de fichiers	
First Project.exe	28/03/2021 03:36	Application	627 Ko
UnityCrashHandler32.exe	28/03/2021 03:31	Application	1 043 Ko
UnityPlayer.dll	28/03/2021 03:38	Extension de l'applic...	21 494 Ko

The files in that folder after the build



FULL SCREEN GLORYYYY !!!

For the brave and motivated out there

If you had some trouble reaching this point, don't worry, we'll talk again about all this and you will have other opportunities to understand those recurring concepts of Unity.
But if you felt like this was too easy, then I have a few little challenges for you!

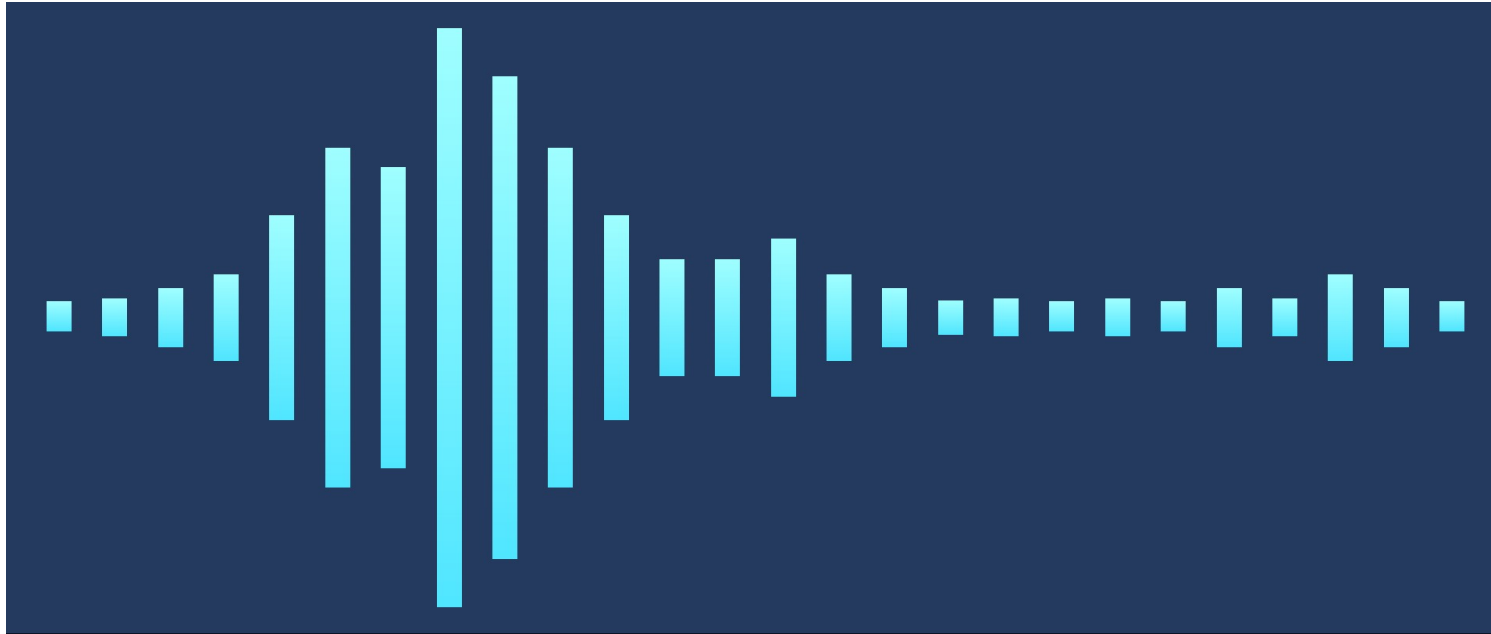
Secondary objectives : A good way to consolidate what you learned!

- Adjust the speed of the basket and the gravity of the apples to make the game more fun *(easy)*
- Introduce randomness in the time interval between apple creations *(easy)*
- Add a global 120 seconds timer. When it reaches zero, stop the game and return to title *(hard)*
- Display that timer using another text field and using the "00:00" format *(hard)*

You have the right to ask questions if you lack information in order to perform those tasks

*** **GOOD LUCK!** ***

Next time...



Sound and Video in Unity!

(make sure to keep a save of this week's project)