Unity 101 To Get Jammin’

The Unity survival guide for game jamming

Yohann Degli Esposti

3D/XR developer, passionate about Unity and game development
I participate in a lot of game jams !

https://github.com/SRWGShinyo

https://srwgshinyo.itch.io/

https://www.linkedin.com/in/yohanndegliesposti

Pre-requisites and disclaimer

- Bases of C# is a plus
- Not an in-depth guide **at all**
- Focuses on 2D, applicable on 3D
- **Practice makes perfect**
  - Unity API : *https://docs.unity3d.com/Manual/index.html*

Introduction

- How to make a basic game in Unity
- Slides with  are practicals
- Slides with  are for our project : we will try to build an easy platformer
- Have fun !

TABLE OF CONTENTS

01

Unity - What is it ?

Unity in a few concepts

A long life

Out since 2005

Game engine

First purpose : create games

Industry-Ready

Strategy to be the engine for industry-grade 3D

2D/3D/XR

Boundless

Pro, Premium, Free

Good support, good tools, often updated

Graphics

Different graphic pipelines for different uses

ALTERNATIVES

Unreal Engine

Well-known, used for AAA games, strong graphic pipeline

Godot

Easy to learn, 2D and 3D, open source, own scripting language

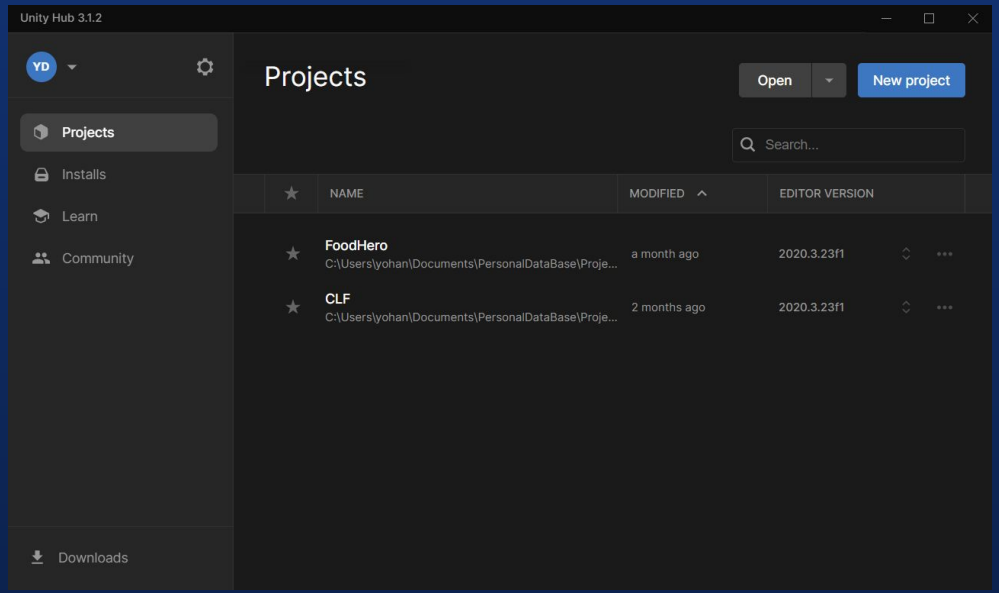
GameMaker Studio

Very professional, Buy to Use, Good support, 2D oriented

02

Unity - Editor

Unity Hub



- Manage Unity Installs
- Gateway to all projects
- Reference to Learn

Let’s create our project !

- Create a new 2D project
- Built-in pipeline, for effectiveness sake
- Name it, store it, let the magic happen !

Tip : A 2D project is only a 3D project on 2 axis…it’s easy to switch !

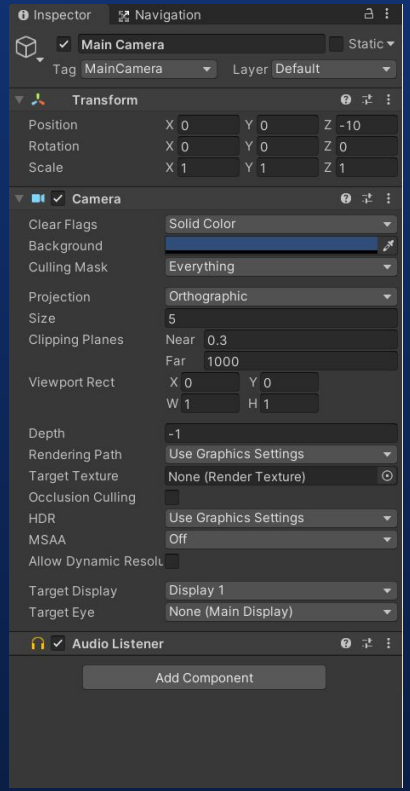
Unity Editor

- Principal view to create
- Can seem very verbose
- Let’s decompose it

Unity Editor - Hierarchy

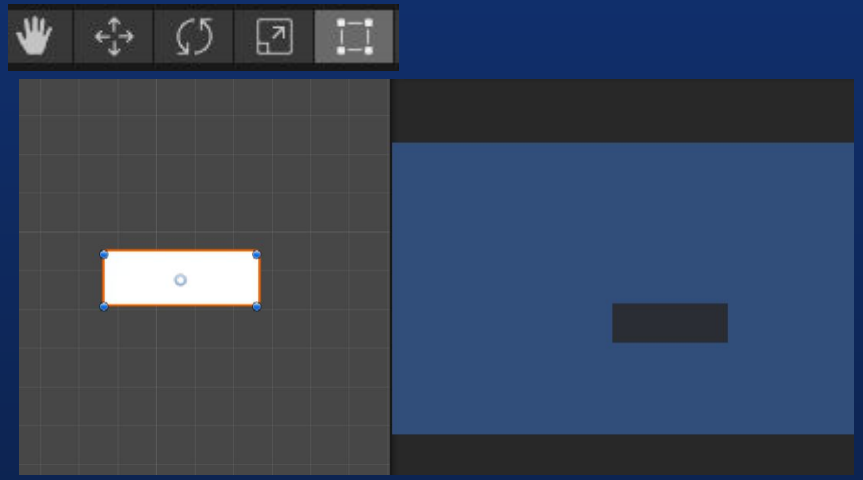
- Displays all object in the scene
- Possible to parent / unparent an object from here

Unity Editor - Inspector



- One of the most important tabs
- Selecting an object in the hierarchy tab will display its information here
- Possible to add/remove components, change values, settings…on an object

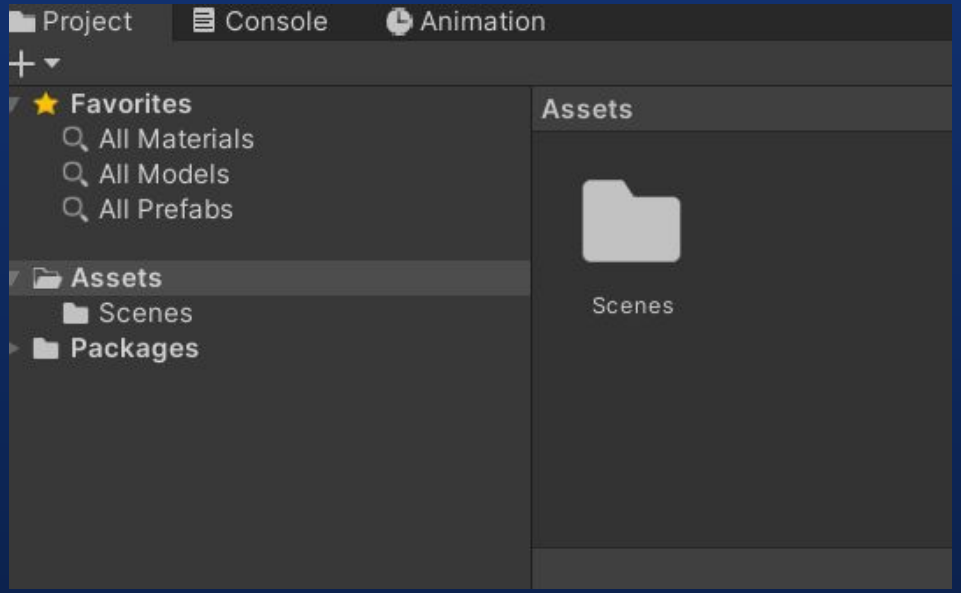
Unity Editor - Scene and Game Views



- Scene view displays the aspect of the scene while editing
- In the scene view, we can **Rotate, Move, Scale** objects as we see fit…
- Game view displays the game when running
- Go into game view with ‘play’

*Modifications of objects **only** apply in editor mode, they are reseted in game mode*

Unity Editor - Project Panel



- Contains all resources imported
- Is used to organize the workspace
- Heart of the project

Unity Editor - Many More

- Console, Lighting…all in due time
- We search effectiveness for a game jam
- Let’s cross that bridge when we come to it !

Fiddle

- Take some time to fiddle with the editor
- Asks questions if needed !

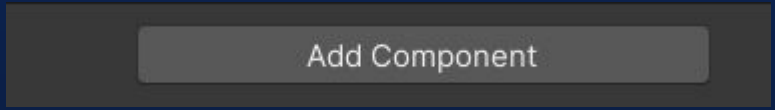
Tip : A 2D project is only a 3D project on 2 axis…it's easy to switch !

03

Library and GameObjects

Library - A word on ECS

- Unity is an ECS : **Entity Component System**
- It means that a **gameobject (GO)** is defined by the components we give it.
- The engine has a lot of pre-made components (camera, colliders, renderers…)
- Add a component by clicking on the **gameobject** and selecting “Add Component” in the inspector



But what if I don’t find THE Component ?

Well, you’re a dev…make it yourself !

Library - Language

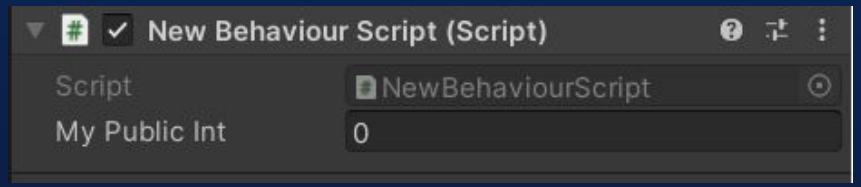
- Unity uses C# to create **scripts**
- Create a script by selecting the option in the menu
- Also possible to create it directly in a folder by right-clicking

Library - UnityEngine

- The scripts we create enable us to use the **Unity Library**
- Extends **Monobehaviour**: the class to identify it as a component
- Any **created component** can be referenced in a script
- A variable marked ‘public’ in a component can be **filled within the inspector** !

```
public class NewBehaviourScript : MonoBehaviour
{

    public int myPublicInt;
```



Library - LifeCycle

- A monobehaviour has a lifecycle composed of **multiple function**
- Most used : *Awake, Start, Update and FixedUpdate*
- Remember there are others !

Tip : The full lifecycle can be found here: https://docs.unity3d.com/Manual/ExecutionOrder.html

Libary - Awake()

- Loaded before the first frame
- Good to initiate variables local to the script / object

DO NOT TRY to access other gameobjects/components of the scene in Awake(), they might not be loaded yet.

Library - Start()

- The first frame for the game object, **fired once**
- Here, we can access **every other** gameobject in the scene.

Library - Update()

- Runs at every **game frame**
- Is very device-dependant, varies according to the game
- Should be used for **repeating computation**

Library - FixedUpdate()

- Runs in sync with the **Physics Engine**
- Can be called **zero, one or more times between** each call to Update()
- Should be used for **physics-based computation** (movement, collisions…)

Tip : https://answers.unity.com/questions/10993/whats-the-difference-between-update-and-fixedupdat.html

You speak about gameobjects...what is ?

Good question you there !

GameObjects- 101

- **Basic bricks** of Unity Logic
- **Containers for components**, **everything in the scene** is a game object
- At least one component **: transform component**
- A monobehaviour has access to its **underlying gameobject** and **transform**

GameObjects- Transform

- **A transform** is a component representing an object in space
- **Composed** of Vector3 (x,y,z)
- One Vector3 for each aspect of the geometry: **position, rotation, scale**

GameObjects- Basic GameObjects

- We can create 3D or 2D **pre-made gameobjects**
- These go from shapes (circle, rectangle…) to 3D objects (spheres, boxes), to light, camera…
- To add one to a scene : **right-click on hierarchy, 2D/3D object -> select the one you want**

Tip : You can also go to the 'Create Gameobject' toolbar menu at the top of the screen.

Play with Transforms

- Create a new 3D box (even though we are in 2D !)
- Take some time to inspect it
- Move it around, scale it, rotate it…

Tip : Can you guess why it appears dark in the game ?

GameObjects- Parenting

- We can make an object **parent of an other**
- This means that every **modification to the parent transform** will affect **the child(ren) transform(s)**
- When parented, an object is moving in it's **local space**, while the top-parent still moves in **world space**

Parenting ! Yay !

- Create another empty GO
- Parent our box to it
- See how the position/rotation/scale of the child are influenced by those of the parent.

GameObjects- Prefabs

- A prefab is a copy of the state of an object at a given time. **Can be reused as is.**
- Modifications on a prefab **are immediately repeated** on all instances of the prefab
- Modifications on an instance **can be repeated** on all instances and on the prefab itself
- Create a prefab simply by **drag'n dropping the GO into the project tab.**

USE THEM !!!!!!!

USE THEM !!!!!!!

Prefabs are a VERY VERY powerful feature

USE THEM !!!!!!!

USE THEM !!!!!!!

USE THEM !!!!!!!

Prefab this, prefab that

- Make a prefab from our parent/child object
- Create a new Instance from it
- Play with the prefab and see how it impacts the instances
- Play with instances and dispatch changes to the prefab

Great...can we code ?

We most certainly can !

GameObjects and Code - Assemble!

- Script can access its underlying **transform and gameobject**
- In the script, use ***transform.**** and ***gameobject.****
- But there are a lot of other useful parts of the API…

Tip : In code, gameobject.transform. == transform.**

GameObjects and Code - Geometry basics

- Vector3 and Vector2 represent a position in space

```
Vector2 vec2 = new Vector2(x, y);
Vector3 vec3 = new Vector3(x, y, z);
Vector3 othervec3 = new Vector3(vec2.x, vec3.y, z);
```

- Quaternions represent rotations it's a very special concept to understand (i won't explain it here, it's long)

```
Quaternion quaternion = Quaternion.Euler(x, y, z);
```

Tip : Learn quaternion the easy way : https://www.youtube.com/watch?v=zjMuIxRvygQ

GameObjects and Code - Geometry API

- UnityEngine imports an API for Vector3, accessible through the class and instances

```
Vector3.RotateTowards(vec2, vec3, 0,0);
Vector3.Angle(vec2, vec3);
Vector3 newvec3 = new Vector3(x, y, z);
newvec3.Normalize();
newvec3.Scale(vec2);
```

- Also true for quaternions

```
Quaternion.Euler(vec3);
Quaternion.RotateTowards(q1, q2, 0);
Quaternion quaternion = Quaternion.Euler(x, y, z);
quaternion.Normalize();
```

Tip : Check the Vector3 API and Quaternion API

GameObjects and Code - Transform API

- Unity Engine gives us a lot of function to work with Transform through its Transform API

- Some are more used than others : *Transform.Rotate(), Transform.RotateAround…*

```
transform.position;
transform.rotation;
transform.localScale;
transform.up; // local Y direction of the object
transform.right; // local X direction of the object
transform.forward; // local Z direction of the object
transform.Rotate(new Vector3());
transform.RotateAround(transform.position, transform.up, 0);
...
```

GameObjects and Code - GameObject API

- Unity Engine gives us a lot of function to work with GameObject through its GameObject API

- Some are good to know: *GameObject.Instantiate(), GameObject.Find()...*

```
gameObject.GetComponent<COMPONENT_TYPE>();
gameObject.activeSelf = ACTIVE_OR_NOT;
GameObject.Instantiate(OBJECT_TO_CREATE, POSITION, ROTATION);
GameObject.Find("OBJECT_NAME");
GameObject.FindGameObjectsWithTag("TAG");
```

Tip : Use gameObject.SetActive(bool) to activate/deactivate a GO.

Let’s move

- Create a new script
- In it, let’s make the object of the script rotate on its ‘Z’ axis
- Put the script on the parent / the children, and observe !

Tip : You might want to check the RotateAround function…

04

Rendering

“We will try to be quick, as jammers don’t usually have the time to play a lot with rendering (even though they should)”

— Me, at some point in life

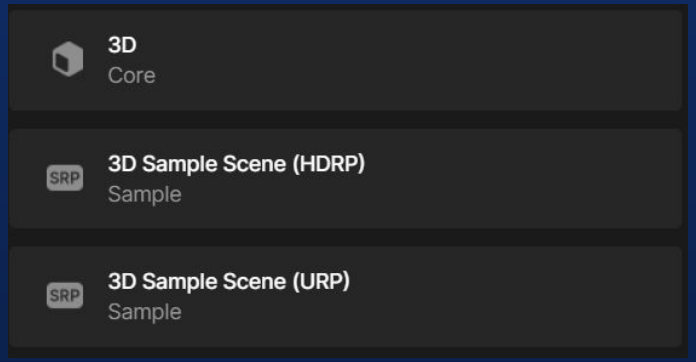
Rendering - Pipelines

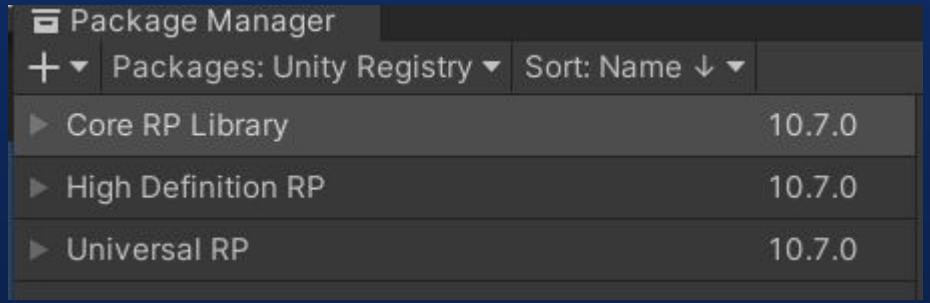
- Unity has different graphic pipelines to use
  - Built-in : *the default one we will use.*
  - URP: *Universal Render Pipeline*
  - HDRP: *High-Definition Render Pipeline*

We use the Built-in for quickness sake, but at some point you will want to use URP/HDRP for better graphics.

Rendering - Pipelines

- Choosing a pipeline is very important.
  - Can be done through the Unity Hub Templates or through package manager





Tip: All 3 are available for 2D and 3D as well. See Brackeys Video.

Rendering - Foreword

- Rendering topic **is large**
- We don’t **have the time** to dive in it

The more you play with the engine, the more you will want to learn, as good graphics are key to a very good user experience !

Rendering - Textures

- Textures are an image/movie that lay over a **GO to give a visual effect**
- Combined with **Texture Map, Normal map**...it can really create depth and effects !
- As we are in 2D, we will use simpler textures for our games

Tip : More in-depth material in the Unity Documentation

Let’s move

- Import the texture ‘ship.png’ by drag and dropping it
- Look at the import settings
- Drag and drop it on the box…what happens ?
- Change the texture type

Rendering - Materials

- Materials are used to describe how lighting influences our objects
- We won’t use a lot of their features
- Find more in the documentation

Materialise

- Create a new material
- Drag and drop it on our box
- Fiddle the settings

Rendering - Meshes

- For 3D, we use *meshes* to render a shape
- It's a collection of triangles
- Unity uses Mesh Filters and Mesh Renderers

We will not use meshes much, as they usually represent 3D shapes.

Let’s pause here...breathe in...breathe out...

How are you handling it ?

Rendering - Sprites

- To display 2D pictures, we will use **sprites**
- It's simply a type of texture, optimised for rendering
- Unity uses the Sprite Renderer component to display sprites

Spriterise

- Change the ship texture type into sprite
- Drag and drop it in the scene
- Look at what happens : *unity creates a game object for us !*

Rendering - Spritesheets

- Spritesheet are a collection of sprites
- It allows to optimize space
- Useful for animations

Rendering - Sprite Editor

- Is used to slice and modify sprites in spritesheets
- Can be installed through the package manager in the *2D Sprite* package

Prepare our character !

- Let’s start our project, by our character !
- Import the alien.png sprites, if not set, set texture type Sprite/UI
- Check you have 4 sprites : idle, walk (2) and jump

Create our character

We will put our graphic assets as a separate game object, to allow for modifying them without touching the ‘physics’ part of the character.

- Create an empty GO “Character”
- Create an empty child GO “Graphics” to “Character”
- Add a Sprite Renderer Component to “Graphics”
- Drag’n drop the character idle sprite into the component

Let’s pause here...breathe in...breathe out...

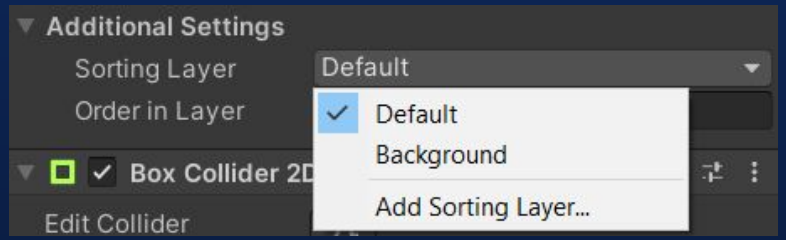
How are you handling it ?

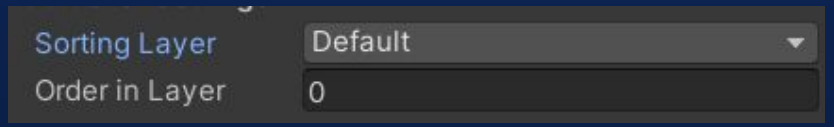
Create coins !

- Create a new GO ‘Coins’, add a ‘Graphics’ empty child GO
- Add a Sprite renderer to Graphics, drag and drop the ‘coin.png’ sprite

Rendering - Layers

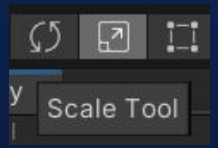
- Possible to put sprites in **Layers**
- Will allow us to tell Unity which Sprite displays above which
- Top layers display in the back, bottom layers in the front.





Put a background

- Create an empty GO “Background”
- Add a sprite renderer to it, and drag and drop the *“background.png”*
- Create a new sprite layer ‘Background’, and add it to our sprite renderer
- Rescale the object using the scale tool

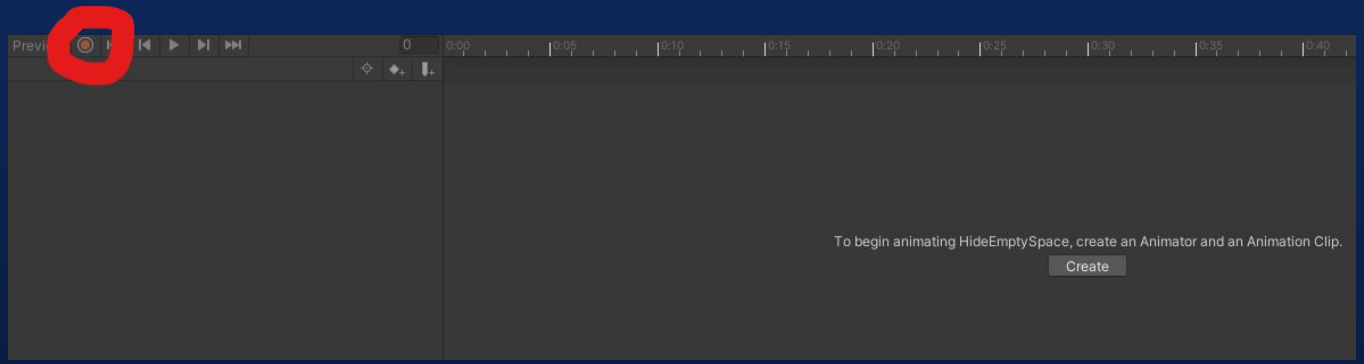


Rendering - Animations

- Unity uses *Animations* object to describe an animation
- They are played and coordinated by the *Animator Component*
- Animations are just modifications of an object

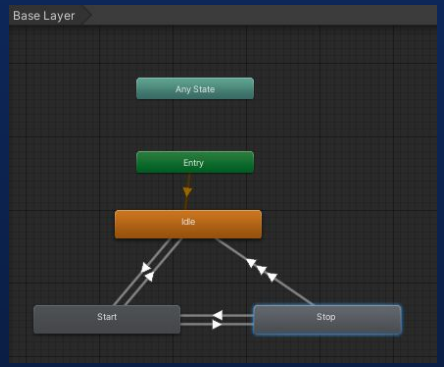
Rendering - Animation window

- The animation window can be opened through *Window -> Animation*
- It allows to start animating an object. **It will automatically create an animator for this object.**
- We can simply **start recording** the animation, animate, and stop



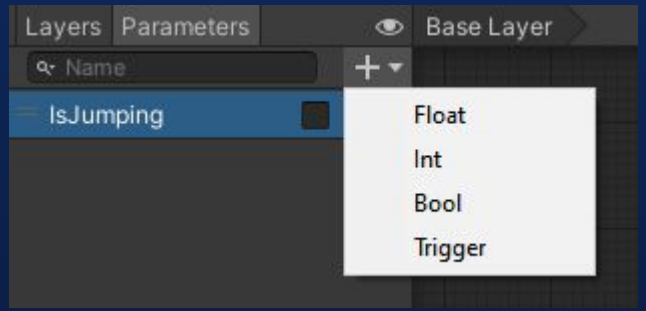
Rendering - Animator window

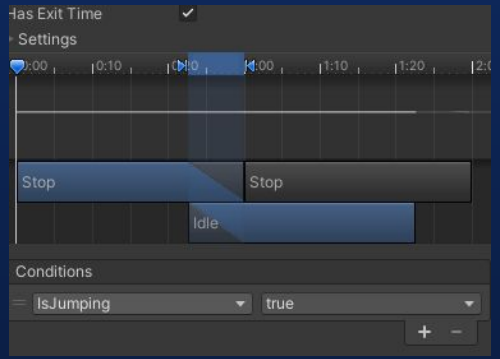
- The animator window can be opened through *Window -> Animator*
- It allows coordinate the animations, and set up triggers to play / switch from one to another
- The orange one is the **default animation, and is played at the start of the object.**



Rendering - Animator Parameters

- A parameter is used by the animator to **make a transition** from an animation to another **depending on a condition**
- Can be **a trigger, a bool, a float…**
- We can tweak the transition settings, and of course, set **the parameter through code.**





Animate our character

- Select our graphics GO
- Open animation window, select ‘Create’
- Press record
- Change the sprite in the component and create the walk animation
- Rinse and repeat for Idle and Jump animations !

Rendering -Lighting

- Unity supports different lighting types: **directional, point, area…**
- Strongly customizable
- By default, 2D sprites **aren't affected by lighting**.
- We can change that by using **an other material**, or using a **more advanced rendering pipeline**

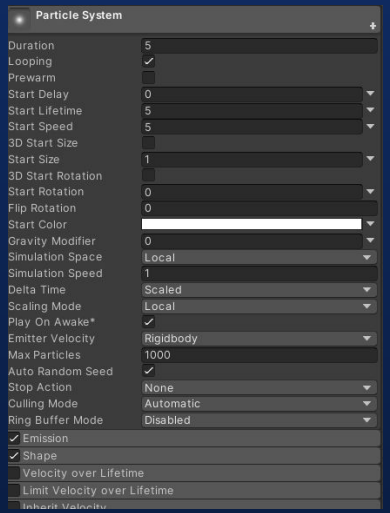
Tip: Making 2D sprites affected by light can be useful to create ambiances.

Lighting

- What would happen if we added a directional light to our scene ?
- Add one, and comment the result

Rendering - Particle Systems

- To make a game more juicy: we can use Particle Systems
- Right-Click on hierarchy -> Effect -> Particle System



Rendering - Particle Systems

- Like any component, a particle system can **be referenced through code**
- We can make them burst, loop…just play with them !
- **Start and stop them through code**

Create collect particle system !

- Create a new Particle System, it will be played when our player collects a coin
- Untick *'Looping', select 'Stop action: destroy'*
- Fiddle with the effect until you've got yourself a good 'collect' effect and make a prefab out of it !

Tip: Take a look at the 'Emission' section, mostly the 'Burst' part...

Rendering - Foreword

- We **barely scratched the surface** of what unity has to offer
- Take some time to search for code and references
- **Shadering, materials, pipeline…a whole new world !**

05

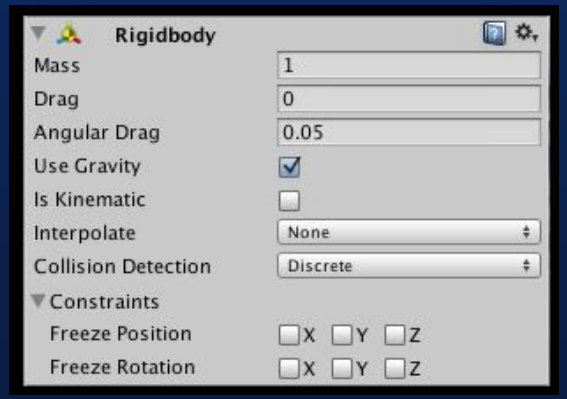
Physics

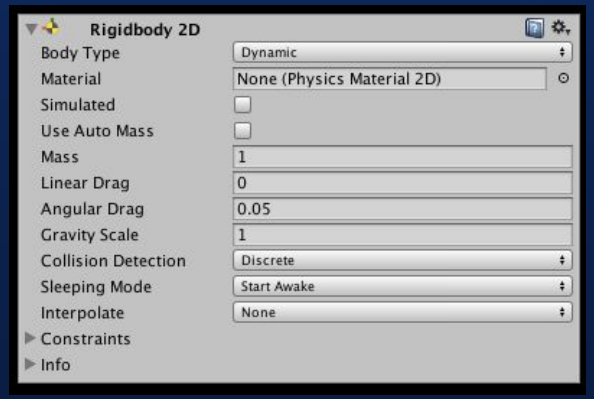
Physics - A whole new world

- Using an engine gives us *physics*
- Will **mimic real life physics**, can be fiddled with
- A lot of physics components exist for **2D and 3D development**

Physics - Rigidbodies

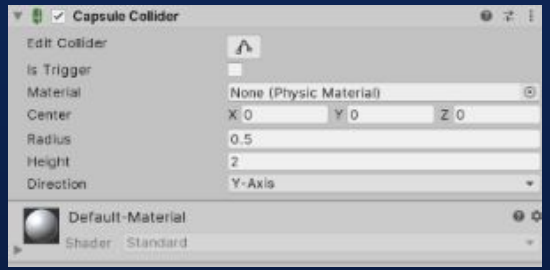
- A rigidbody is used to tell Unity : *I want this object to be influenced by Physics*
- At least **one rigidbody** to enable physics between **two game objects**





Physics - Colliders

- A collider is a component that defines **an area of collision** around an object
- Can be of any shape, Unity gives us a few *(Box, Circle, Sphere, Mesh…)*
- Colliders are **2D or 3D** specific. *(Use sphere for 3D, circle for 2D)*



Let’s get physical - 1

- Put a rigidbody on our Character’s GO
- Create a new Cube GO, put the character above it.
- Start the game. What happens ? Why ?

Let’s get physical - 2

- Replace the Box Collider with a Box2D Collider
- Start the game. What happens ? Why ?
- Replace the Rigidbody with a Rigidbody2D
- Start the game. What happens ? Why ?
- Add a Collider2D to our player

Physics - Triggers

- Each collider has a *'IsTrigger'* value
- A trigger is an area that will **detect collision** but not emulated them.
- Useful feature for events **that need to get triggered** at a specific time.

Let’s get triggered

- Set the *IsTrigger* value on the box to true
- Start the game and comment what happens

Physics - Through code

- The Unity API gives us function to **detect the components of a collision**
- Are part of the **life** of a component.
- Are defined for 3D and for 2D : use the proper ones !

Physics - Functions

- ***OnCollisionEnter/OnTriggerEnter:*** to detect when the event starts
- ***OnCollisionStay/OnTriggerStay:*** to detect when the object maintains the collision
- ***OnCollisionExit/OnTriggerExit:*** to detect when the collision stops

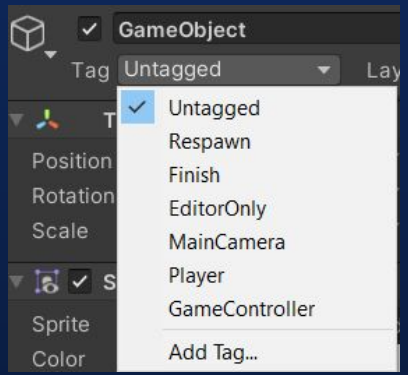
If working in 2D, use their 2D counterparts.

Code our Physics

- On the box, add a new script.
- Implement ***the OnTrigger* and OnCollision**** functions (whichever you want)
- Play with prints to see when events are fired.

Physics - Tags

- GameObjects can be **tagged**
- It allows to identify them **easily**, and group **similar objects in one logic**
- Can find all objects with a tag by using **GameObject.FindGameObject(s)WithTag**

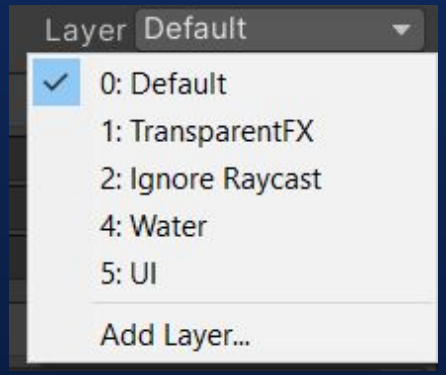



```
GameObject[] gameObjectsTaggedPlayer = GameObject.FindGameObjectsWithTag("Player");
```

Physics - Layers

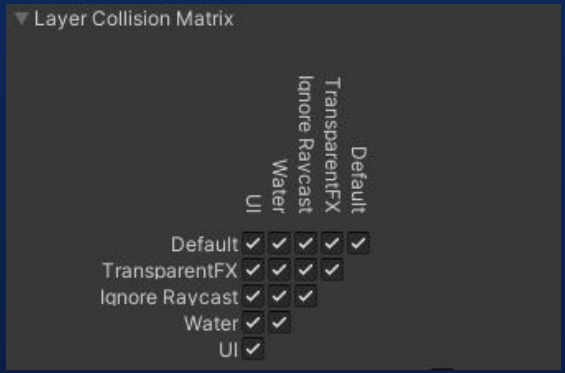
- Like sprites, objects in themselves can **have layers**
- It allows to put similar objects ***in the same 'physic space'***

- We can easily **create or remove layers**



Physics - Collision Matrix

- In the ***Edit -> Project Settings -> Physics (2D)*** tab
- Allows us to define **which layer collides with which**
- By default, a new layer collides with **everything**



Fifty layers of Physics

- Put a new ‘box’ layer on the box and a new ‘character’ layer on the character
- Uncheck the collisions between those in the matrix
- See what happens

Let’s pause here...breathe in...breathe out...

How are you handling it ?

Physics - Joint

- Components used **to connect two rigidbodies, or one rigidbody and a point in space**
- Particularly useful for pendulums, hooks, chains…anything **that implies moving depending on something else.**
- We won't practice them, but <u>check them out !</u>

Physics - Even more

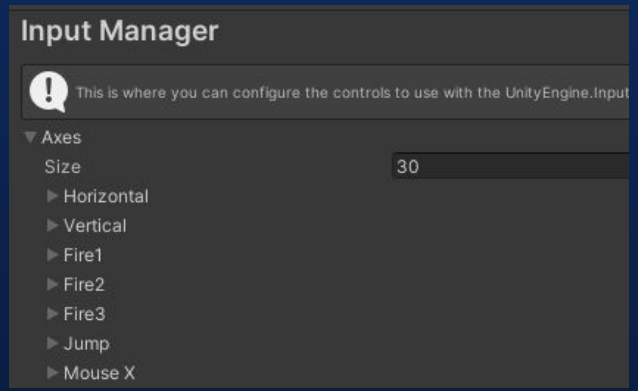
- Unity has much more to offer in terms of physics *(cloth, ragdoll…)*
- Check the documentation out
- You can play with the engine for a long time !

Physics - Inputs

- Let’s briefly talk about **getting input from the player**
- Unity has released **a new input system 4 years ago**
- We will not use it, as it is **more complicated** (but arguably **more powerful**)
- You should **try it at some point**

Physics - Input Mapping

- The old system uses a mapping through project settings
- You can define new buttons, axes…
- Unity will ***associate an input to a name,*** and let you use it
- Axes are **useful for movement**, indicating **where the player is going.**



Physics - GetButton

- ***Input.GetButtonUp("name")*** will fire when said button is released.
- ***Input.GetButtonDown("name")*** will fire when said button is pressed.
- ***Input.GetButton("name")*** will fire each frame while said button is held.

```
if (Input.GetButtonDown("Fire1"))
{
    Debug.Log("The Player fired !");
}
```

Physics - GetAxis

- **Axes** are defined in the Input Map, and will vary from -1 to 1, depending on the direction.
- ***Input.GetAxis("name")*** return the axis value smoothed with sensitivity, gradually changing.
- ***Input.GetAxisRaw("name")*** returns the axis non smoothed. It will return -1, 0 or 1 and only those. (used for 2D mostly)

```
if (Input.GetAxis("Horizontal") == 1)
{
    Debug.Log("The Player is moving in the positive horizontal direction !");
}
```

Physics - GetMouseButton

- ***Input.GetMouseButtonUp(number)*** will fire when said button is released.
- ***Input.GetMouseButtonDown(number)*** will fire when said button is pressed.
- ***Input.GetMouseButton(number)*** will fire each frame while said button is held.

```
if (Input.GetMouseButtonDown(0))
{
    Debug.Log("The Player left-clicked !");
}
```

Physics - GetKey

- ***Input.GetKeyUp(KeyCode)*** will fire when said key is released.
- ***Input.GetKeyDown(KeyCode)*** will fire when said key is pressed.
- ***Input.GetKey(KeyCode)*** will fire each frame while said key is held.

```
if (Input.GetKeyDown(KeyCode.Space))
{
    Debug.Log("The Player pressed space !");
}
```

Physics - Go further

- You might not want to bind gamepads for your first gamejam.
- But you certainly will at some point !
- When this time comes, I ***strongly invite you*** to ***use the new input system.***
- Learning curve is stiffer, but the system is **way more powerful and versatile.**

Tip: See this quick-start guide and this in-depth video.

Let’s pause here...breathe in...breathe out...

How are you handling it ?

Create our platforms - Visual

This is a long practical part. But at the end of it, you will have a playable game ! Hurray !

- Create an empty GO “Platform”
- Create an empty child GO “Graphics” to “Platform”
- Add a Sprite Renderer Component to “Graphics”
- Drag’n drop the *platform.png* sprite into the component

Create our platforms - Physics

- Add a BoxCollider2D to our “Platform”
- Create a “Platform” tag and tag it
- Create a “Ground” Layer, and layer it
- Create a prefab from it…and voilà !

Create our coins - Physics 1

- Take **the coin GO** we created earlier
- Add the collider to it which **seems the most appropriate to you.**
- Check IsTrigger.

Create our coins - Physics 2

- Add a new script to the coin **“Coin behavior”**.
- In it, implement OnTriggerEnter() to **instantiate the particle effect and destroy the coin when the player touches it**
- Can you guess why we have to instantiate the effect by hand ?

*Tip: Take a look at the Destroy() function and Instantiate function. Think about how you could know **the player** (and only the player !) touched the coin.*

Create our flag - Visual

- Create a new flag GO, with graphics, and use the ‘flag_Up.png’.
- Also import the ‘flag_down.png’
- Create a little animation with these two frames !

Create our flag - Physics

- Add a collider, and a new script to the flag object.

- Now, implement the following logic :
  - ➔ When the player touches the flag, if there are no more coins in the scene, print (“You won “!)
  - ➔ Else print (“There are <n> coins left”) with n being the number of ‘coins’ object still in the scene.

Tip: You have all you need to make it through !

Let’s pause here...breathe in...breathe out...

How are you handling it ?

Create the player controller

Hang in there for this one. It's a bit hard, but we'll make it through !

- A player controller can be done **in a multitude of ways.**
- We will use a simple, easy one. It has flaws, but it works for what we want.
- Jamming is also **aiming for effectiveness**
- The more **you play with the engine, the more you will have refined and solid scripts.**

Create the player controller - Move 1

- Add a new script to our player GO : *PlayerController*
- Put a BoxCollider2D on our player. Fiddle with it.
- Put a rigidbody2D on our player. In constraints, fix Z rotation
- Get a public reference to that rigidbody in the script.
- Put the rigidbody in the proper slot in the editor.

Create the player controller - Move 2

- Add a public reference to a layer, called "*WhatIsGround*"
- In the editor, select the *'Ground'* layer for *'WhatIsGround'*

Create the player controller - Physics

- Add a BoxCollider2D to our character
- Add an empty Gameobject ‘Feet’ to our Character
- Put it around the feet of our Character
- Get a reference to ‘Feet’ in the script

Create the player controller - Move

- Add a public float variable : speed
- Get the value of the “Horizontal” Axis in update()
- Move the character along the X axis
  - Take the character transform.position
  - Add it a new Vector3, with movement as X
  - Multiply it by Time.deltaTime and Speed

Tip: Time.deltaTime returns the time since last frame. Good for smooth movement !

Create the player controller - Grounded

- Add a private boolean ‘isGrounded’, a private float ‘groundDistance’
- In Update, update the value of ‘isGrounded’

  - Use the *Physics2D.OverlapCircle* function
  - Pass in the position of ‘Feet’, ‘groundDistance’ and ‘whatIsGround’
  - ‘isGrounded’ is the result of the function

Tip: We use Physics.CheckSphere to check if the ground is below us.

Create the player controller - Jump

We will use the properties of our rigidbody to jump. It can be of course done differently !

- Add a *public float JumpForce*
- Let's implement the following logic in Update()
  - ➔ If the player presses "Jump" key and "isGrounded" is true
  - ➔ Call the rigidbody.AddForce() function
  - ➔ Pass In a new Vector2, with our JumpForce as Y parameter
  - ➔ Pass In the ForceMode: *ForceMode2D.Impulse*

Tip: Impulse will give the player a little push in one direction. See ForceModes

Let’s pause here...breathe in...breathe out...

How are you handling it ?

Create the player controller - Animations

- Get a private reference to our animator
- In the Start() function call *GetComponent<Animator>()*
- Now, in Update()
  - ➔ When the player Jumps, call *Animator.Play("Jump")*
  - ➔ When grounded and Horizontal != 0, call *Animator.Play("Walk")*
  - ➔ When grounded and Horizontal == 0, call *Animator.Play("Idle")*

Tip: Replace "Jump", "Walk" and "Idle" by the corresponding animation names for your project.

Create the player controller - Rotate

Let's use the Horizontal Axis value to rotate our player !

- Let's implement the following logic in Update()

  - ➔ If the HorizontalAxis value is not zero (*see Mathf.Approximately*)
  - ➔ Take the *transform.rotation* of our player
    - ★ If HorizontalAxis < 0, it equals Quaternion.Euler(0,180,0)
    - ★ Else, it equals Quaternion.Identity

Tip: Quaternion.Euler(0,180,0) makes an object rotate 180 degrees on its Y-axis. Effectively, this makes it turn around !

Create the playground

- We will aim for something simple
- Create a platform, a big one, with player and flag over it
- Put some coins between the flag and the player

You did it !

- You now have a playable little game ! Well done !
- Of course, we aimed for effectiveness, but things can be done in better ways.
- What if our player gets out of bound, for example ?
- But hey…you will do that yourself !

Tip: Bravo !

06

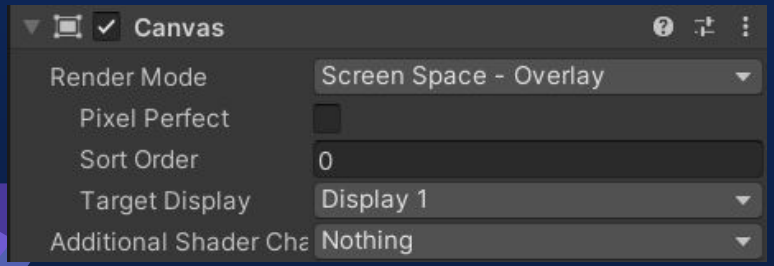
User Interface

User Interface - Foreword

- Alike Inputs, Unity gives us the UI Toolkit for UI
- We will however use the ‘standard system’, easier to learn
- You have to add the *UnityEngine.UI* dll for it to work

User Interface - Canvas

- The Canvas is the main UI component add can be added like any other object
- It can in screen space (for basic UI) or world space (for embedded UI)
- It can be fixed or scaled as you ask it to.

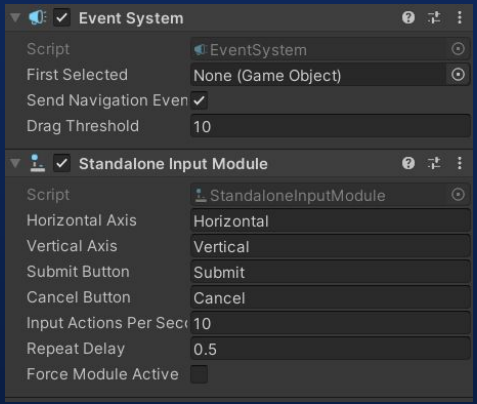


Paint me like one of your UIs

- Add a new canvas to our game
- Make it render in screen space
- Make it scale with screen size

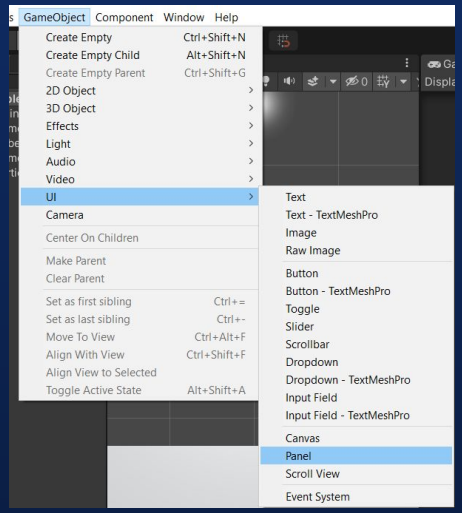
User Interface - Event System

- When you added the Canvas, it also added an ES
- This component will handle UI events and interaction
- Without it, the UI won't be interactable with



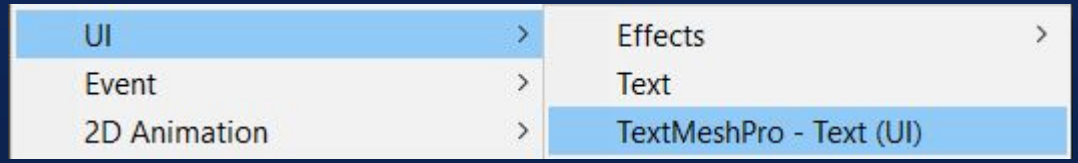
User Interface - Panels

- Panels are simple Images, good for UI backgrounds
- Can be added through GameObject -> UI -> Panel
- Color can be changed, transparency, scale…



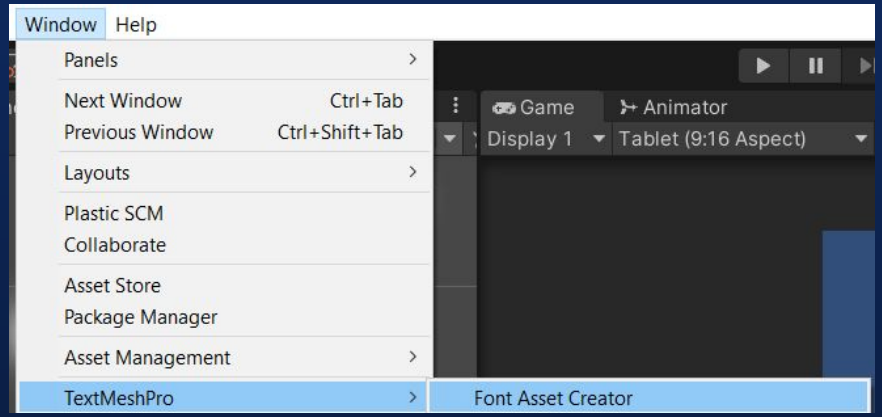
User Interface - TextMeshPro

- TextMeshPro is a package that enables RichText
- It has been integrated to Unity, but was not until 3 years ago
- TextMeshPro allows for html tags, custom fonts, more responsive Text...



User Interface - Fonts

- TextMeshPro enables us to use custom fonts
- We just have to import them like any other assets
- And then use the font creator !

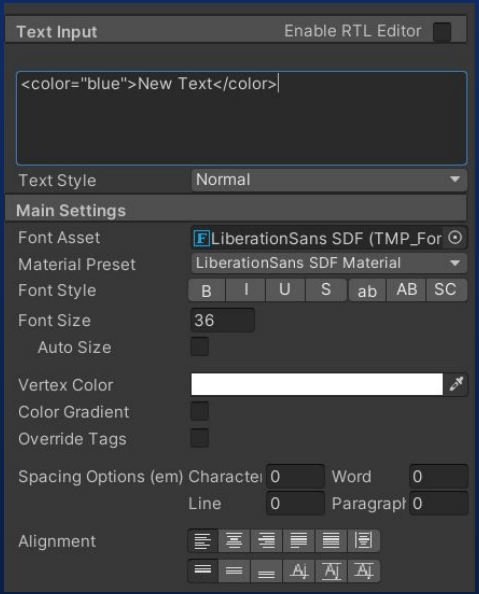


Add a custom font

- Create a ‘Fonts’ folder and import ‘game-font.ttf’
- Use the font asset creator to create the proper font

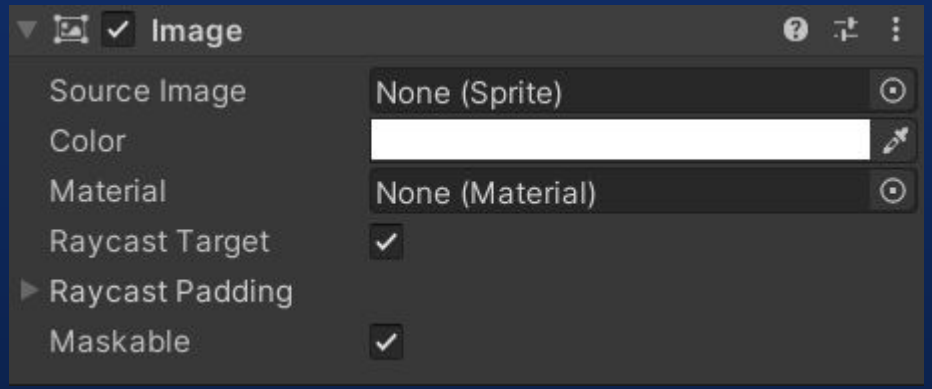
User Interface - Texts

- Texts are highly customizable
- Allows to display information to the user



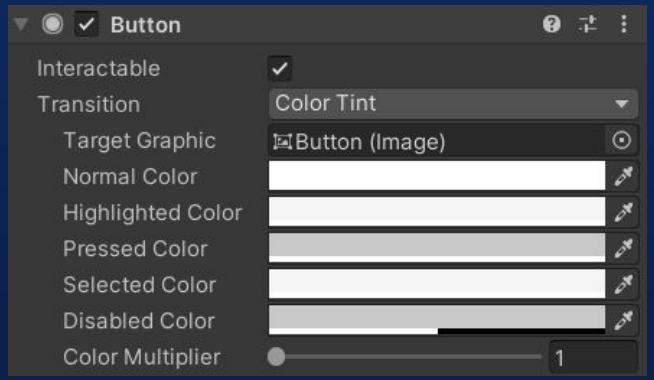
User Interface - Images

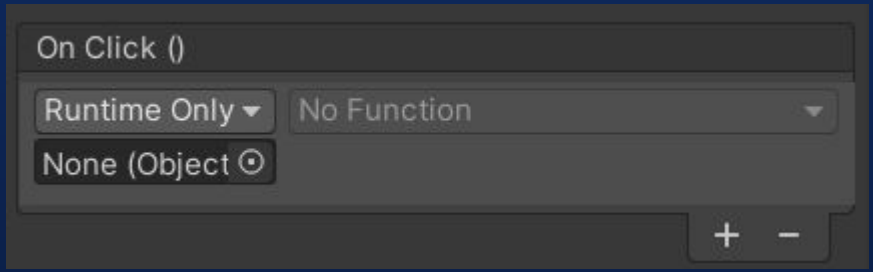
- Allow to display a sprite in the UI
- Can be scaled, colored, interacted with…



User Interface - Buttons

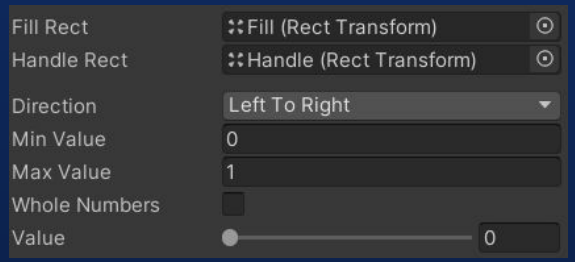
- An image and a text, with animations
- Actions can be animated directly on the component
- Possible to bind a public function to a click event

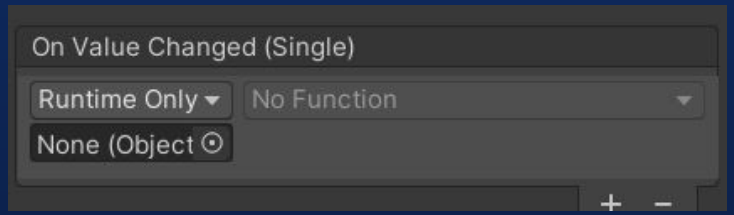




User Interface - Sliders

- Images, highly customizable
- Can be used for health bar
- Bind an action on the component or through code

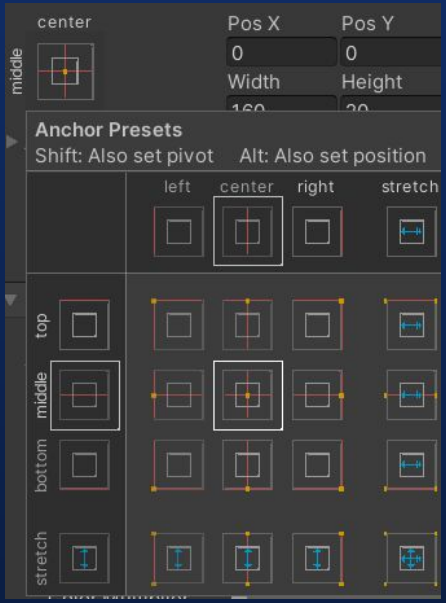





```
Slider slider = FindObjectOfType<Slider>(); // Never do that please
slider.onValueChanged.AddListener((newValueOfTheSlider) =>
                                { Debug.Log("Here is a new value " + newValueOfTheSlider); });
```

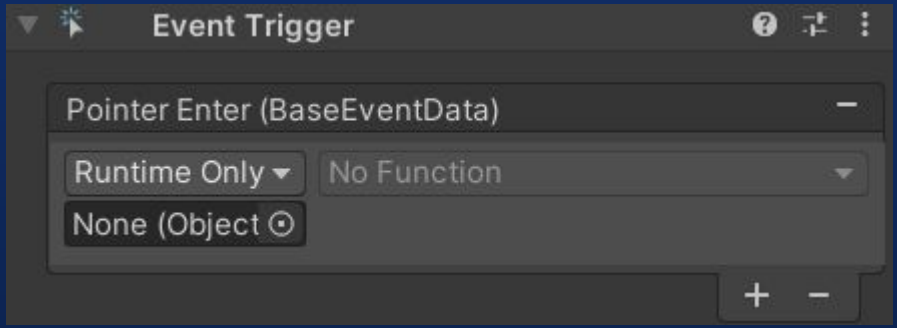
User Interface - Anchors

- Tell Unity where to put the UI element, thanks to a pivot
- Anchoring means the engine will try to put the element ‘around here’ when the screen resizes.



User Interface - Event Triggers

- Add an Event Trigger component to a UI element
- Allows you to bind public function to special actions



User Interface - Foreword

- There are **a lot more components**
- Way to organize your UI, make dynamic lists…
- You will have to search for yourself when needed !

Tip: Keep in mind that Unity has extensive documentation…even for UI !

Create me a winning screen

- Add a panel to our canvas
- Inside it, create a winning screen (use texts, images…whatever you want !)
- Use the editor to deactivate the panel gameobject

Appear me a winning screen

- In our flag behaviour, add a GameObject public variable, named ‘winPanel'
- Drag’n drop the panel in the editor
- When the player wins, activate the GO !

Tip: UI object are GameObjects with Components ! Hence, they can be activated/deactivated, and enabled/disabled

No move when you win !

- In our flag behaviour, make a public reference to our Character Controller
- When the player wins, disable the character controller

07

Audio

“A game without SFX is just an empty shell. SFX without a game is SFX.”

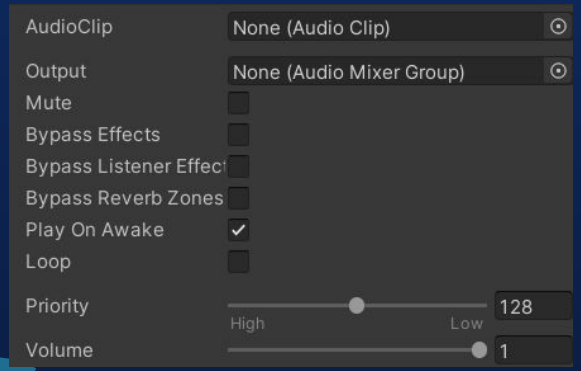
— Socrate

Never forget music and sfx

- Music and sfx are key to a good experience
- They can save a game, or burn it to the ground
- Easy with Unity

Audio - Audio Source

- Basic component to put audio
- Can be played on awake, or referenced through script
- Can be looped, 3D, change volume…customizable



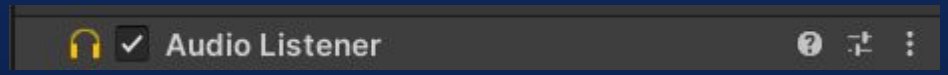

```
0 references
public void MakeMyAwesomeSound(AudioSource source)
{
    source.Play();
    source.Stop();
}
```

Audio - Audio Mixer

- Found in Window -> Audio -> Audio Mixer
- Allows to create group of musics, and put effects on them
- A lot of possibilities, check the documentation

Audio - Audio Listener

- One (and only one !) audio listener has to be in the scene to hear audio
- A new Camera object has one by default



Yo, make some sound !

- Create a new ‘Audio’ folder, import the *ambiant.mp3, coin.mp3, jump.mp3, victory.mp3* and *menu.mp3* sounds
- Create a new AudioSource ‘Ambiant’, add the ‘*ambiant.mp3*’ clip
- Tick the *PlayOnAwake*, tick the *Loop*, adjust the volume
- Play the game, adjust as needed

Ting make the coins

- Create a new AudioSource Coin, add the ‘coin.*mp4*’ clip.
- Untick the *PlayOnAwake*, untick the *Loop*, adjust the volume.
- Make a reference to the audio source in your coin behaviour
- Before the coin is destroyed, play the audio source
- Add the source to all coins in the scene

Tip: Do you see a better way to add the source to all coins without drag and dropping ?

Make more sounds !

- Do the same for jump and win !

08

Export and Scenes

Let’s finish it, and publish our game !

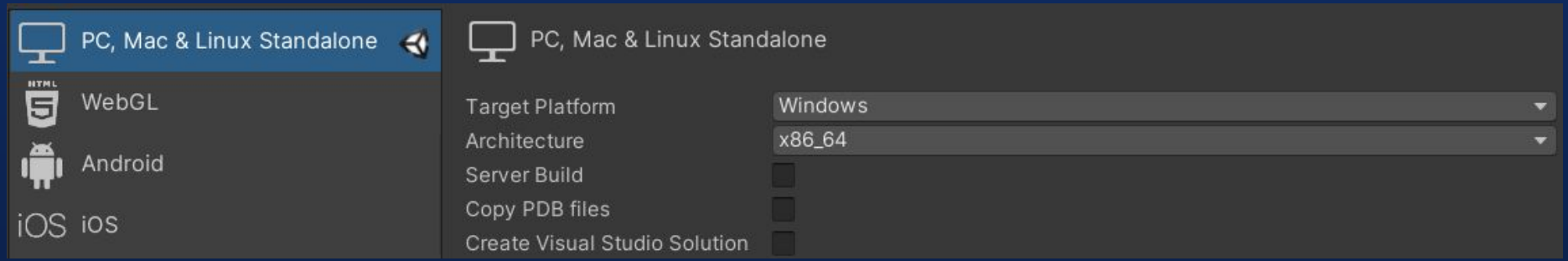
Scenes

- You will most probably want to do more scenes than one
- Every coherent ensemble should be grouped in a scene.

- You will want to connect scenes together

Build Settings

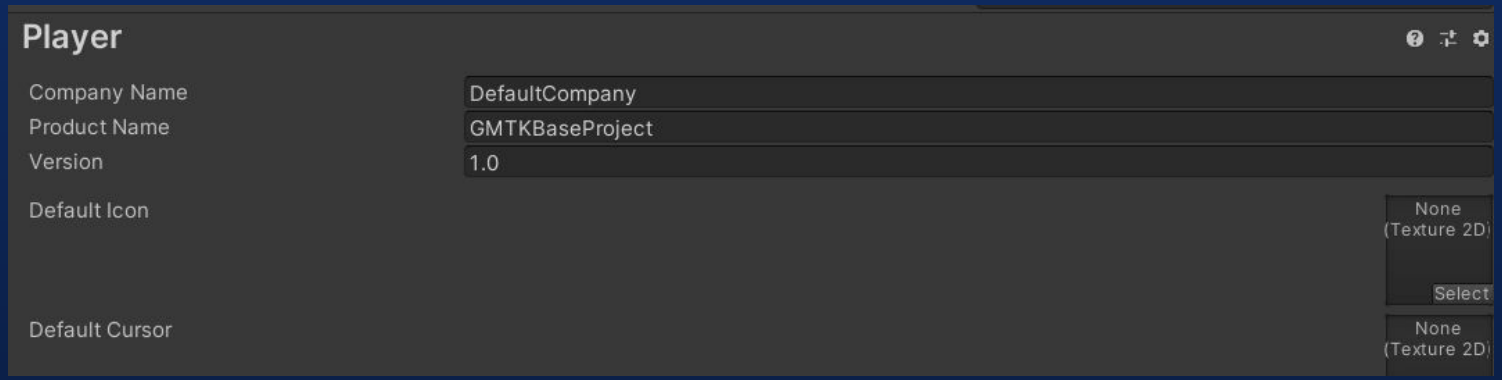
- In File -> Build Settings
- Can change the export platform
- It will recompile the project and assets



For Android build, you will have to download the Android SDK

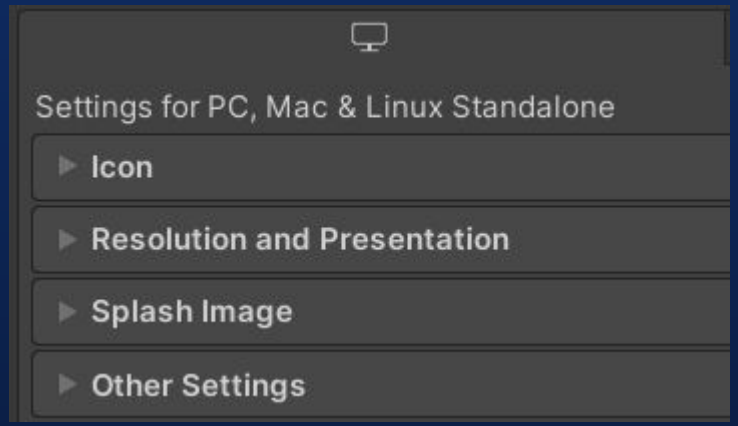
Player Settings - General

- Allows you to set CompanyName, Product Name…
- Change the game Icon, the cursor Icon…



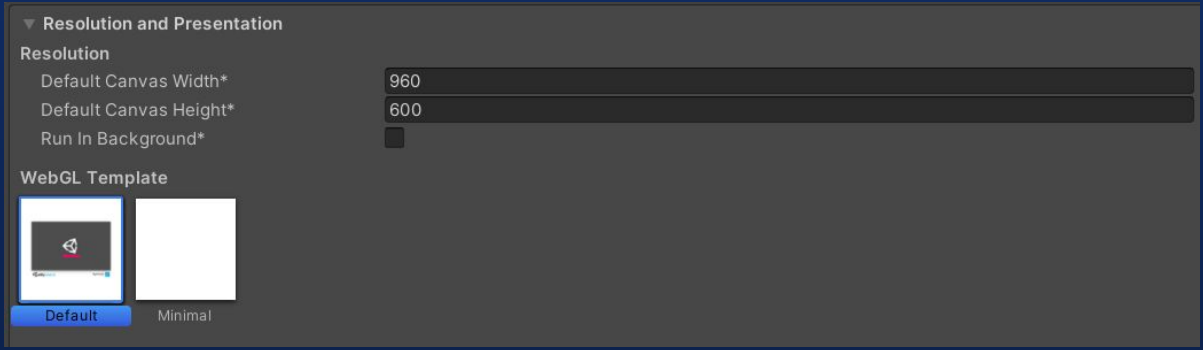
Player Settings - Windows

- Change Icon, Resolution, Window Type
- Optimization Settings
- You will mostly keep it default



Player Settings - HTML5

- Build in HTML5 will give you a WebGL compiled project with an index.html file by default
- Icons, Resolution, Window size, Optimization Settings
- You can choose a template for the index.html



To create your own templates, just check the documentation !

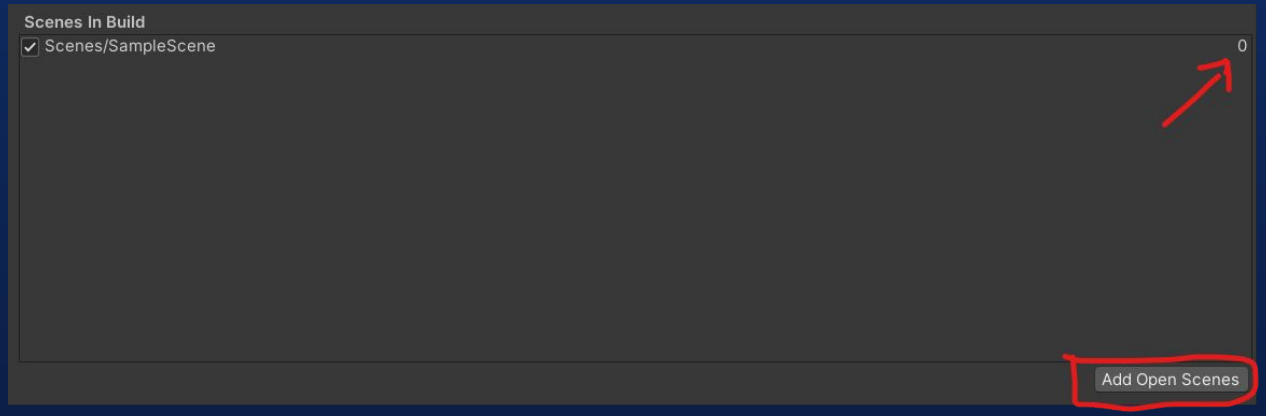
We will go on the web !

- For a game jam, you will most of the time want **at least** a HTML5 release

- Go ahead, and change the build platform type to HTML5

Build Settings - Scene

- Scenes are not added by default to the build settings
- Go into your scene and use proper option
- Move up/down scenes to change their index. ‘0’ is the first scene loaded in the game.



SceneManager

- To manipulate scenes in code, use the Unity.SceneManager
- Import *UnityEngine.SceneManagement* in your code
- *SceneManager.LoadScene(index)* allows to change scene !

```
SceneManager.LoadScene("MainMenu"); // Will load the scene named 'Main Menu', case sensitive
SceneManager.LoadScene(0); // Will load the first registered scene
```

When changing scenes, every gameobject is destroyed. If you need a GO to persist, check DontDestroyOnLoad.

Tired of coding...

- Create a new scene ‘Main Menu’, rename our scene ‘Level 1’
- In that main menu, add a canvas, create at least a button
- Add an onclick on this button that will load our ‘Level 1’ scene
- Into the ‘Level 1’ end game panel, add a button to load our ‘main menu’

We did it ! We finished a little game !

But why would we stop here...

09

Advanced Notions

Disclaimer

- I put here things I didn’t have time to include
- Some are very important concepts, others are just ‘to know’
- Have fun !

Coroutines - Explanation

- Coroutines are a code execute asynchronously.
- Uses the '*yield*' keyword
- Mostly useful for loops and things that need to wait some time
- Called with the *StartCoroutine()* method

Tip : Also check WaitForSeconds(), WaitWhile(), WaitUntil()...

Coroutines - Example

This code will make a gameobject appear smoothly, by scaling it from 0 to 1. (Check .md for in-depth explanation)

```
Unity Message | 0 references
void Start(){
    StartCoroutine(scaleMyGameObjectToOne(gameObjectToAppearOnScreen));
}

1 reference
private IEnumerator scaleMyGameObjectToOne(GameObject gameObjectToAppearOnScreen)
{
    while (gameObject.transform.localScale.x < 1){
        Vector3 actualScale = gameObjectToAppearOnScreen.transform.localScale + new Vector3(0.01f, 0.01f, 0.01f);
        this.gameObjectToAppearOnScreen.transform.localScale = actualScale;
        yield return new WaitForSeconds(0.01f); // we wait a hundreth of a second
    }
}
```

Tip: Go to make a quick appear/disappear visual effect !

DOTween

- A package that uses Digital Twin to emulate movement
- Good package to move, rotate, scale objects quickly
- Look at their documentation

Tip: Every Dotween function can be recoded if you wish !

Camera

- A Camera is added to a new scene by default
- It can be moved, enabled/disabled, rotated, apply effect on it…
- More than one camera can be in a scene
- As it is a very important object, check the documentation

Post Process

- Using pipelines, you will be able to add post-processing effects
- Allows for bloom, vignette, gaussian blur, … even custom effects !

See Post-Processing volume documentation

See Post-Processing for URP

See Post-Processing course from Unity Learn

Cinemachine

- Cinemachine is a Camera Engine
- It allows to make beautiful cutscenes, camera movements…
- So powerful it has been added to the Unity Registry

ShaderGraph

- Added to URP and HDRP
- ShaderGraph is a visual tool to create powerful shaders and materials
- Hard to learn, but worth the time for beautiful effects

VFX Graph

- Powerful and stunning effects for URP and HDRP
- Particles are spawned on GPU, through nodal editor
- Allows for masterpieces and modularity

VFX graph presentation

VFX Graph tutorial

Asset Store

- You can find thousands of assets in the Asset Store
- Directly connected to a game, through Window->Asset Store
- Some are free, 2D, 3D, scripts…take the time to search !
- You don’t need to code everything !

Console

- Open the output from Window -> Console
- Print messages with *Debug.Log(), Debug.LogWarning(), Debug.LogError()*
- Not the only way to debug !

C# Debugging

- In Visual Studio, you can press ‘attach the script’
- This will allow you to use the standard VS debugging tools
- Not the only way to debug !

GitHub

- There is a Git plugin for unity, for better collaboration
- You might however not need it, and it takes time to learn and put in place
- Try it out and make your own mind !

DontDestroyOnLoad

- Call the DontDestroyOnLoad function on a GameObject to make it persist between scenes
- Very useful to embark informations from one scene to another

```
Unity Script | 1 reference
public class GameManager : MonoBehaviour
{
    public static GameManager activeGameManager;

    Unity Message | 0 references
    private void Awake()
    {
        if (activeGameManager != null){
            DontDestroyOnLoad(gameObject);
            activeGameManager = this;
        } else{
            Destroy(gameObject);
        }
    }
```

Photon - PUN

- PUN (Photon 2) is a good way to start multiplayer
- You can check their documentation
- Pretty limited for AAA uses, but good for little games.

PlayerPrefs

- PlayerPrefs allows for saving int, strings, float…
- Useful to store little bit of data between session (a leaderboard for example)
- Check the documentation

JSON

- JSON might be necessary for more data
- JsonUtility is the default Unity API for that
- Check the documentation

10

Conclusion

Difficult

- It’s hard to give so much information
- Take the time to read again the parts you didn’t understand
- It comes with experience !

Practice

- If Unity interests you, **try some side projects !**
- Make prototypes, search for answers, test, test, test !
- It’s very difficult, but you will manage ! Hang in there !

Thank you for following, I hope you liked it !

There is always something to learn !