


















MAJ 05/24

UNITY3D

-  **Développement de jeux vidéos multiplateformes**
-  **Environnement de travail multimédia**
-  **Environnement de codage : script C# sous VisualStudio**

UNITY3D – Présentation

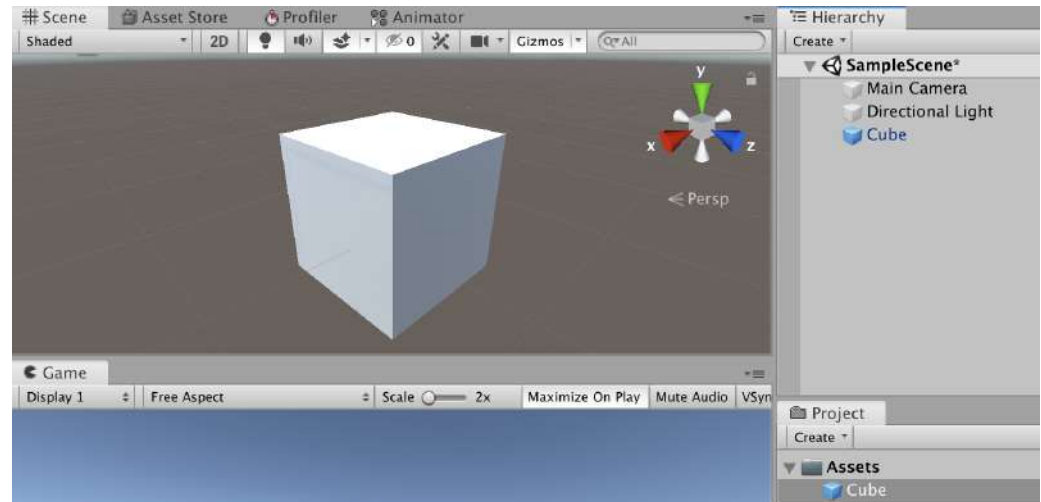
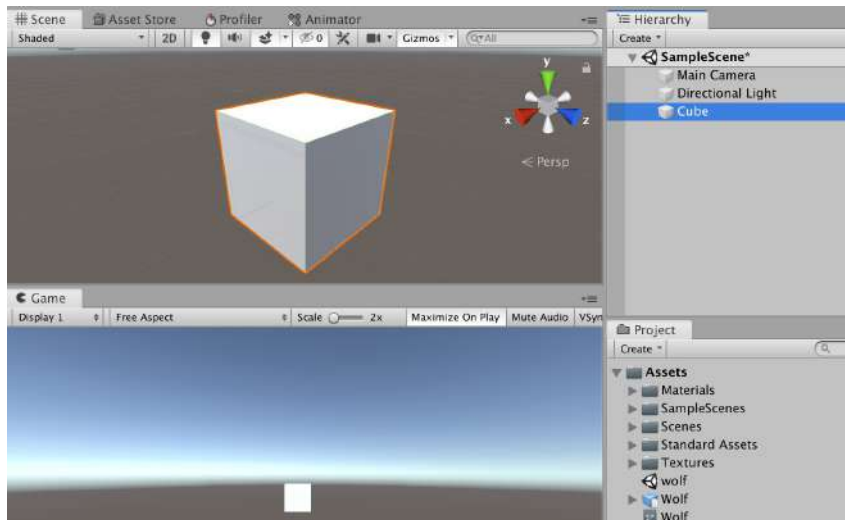
-  Environnement de travail
-  GameObject
-  Prefab
-  Script CS / Le cycle de vie
-  Documentation : Textures / Material
-  Assets et Asset Store
-  Modèles 3D :
 -  Blender (rigging)
 -  MagicaVoxel
-  Collisions (RigidBody) et Trigger
-  Gestion des touches
-  Scènes et UI
-  Production (débogage/travail collaboratif/Build)
-  Base de données externe MySQL : <https://www.youtube.com/watch?v=HwHeaH7Da9A>

Environnement de travail

- Offre Student ou Personal
- Création d'un compte unity
- Choix de la version (Version 6 déconseillée)
- Choix du projet 2D/3D
- **EDI**
 - Organisation des fenêtres
 - Barre d'outils (gestion de la souris)
 - Menu rechercher
 - Organisation du projet et de la scène
 - Project : ressources
 - Hierarchie : instanciation

GameObject/Prefab

- 🌐 **GameObject** : Toute instance dans la scène
- 🌐 **Prefab** : Genre de classe → réutiliser des objets



Script CS/Cycle de vie

- 🌐 Lorsqu'on crée un script CS on hérite de MonoBehaviour (classe de base dans unity). Cela permet d'exécuter un certain nombre de méthodes héritées selon un cycle de vie :

- 🌐 <https://docs.unity3d.com/Manual/ExecutionOrder.html>

- 🌐 Start()
- 🌐 Update()
- 🌐 OnCollisionXXX() : collision physique (on verra plus tard)
- 🌐 OnTriggerEnterXXX() : Zone de collision invisible (ex : dialogue) (on verra plus tard)

Démo : cube qui tourne

Time.deltaTime = constante de temps

pour éviter les saccades = 16ms (60 FPS)

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

public class Tourne : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(Vector3.up * Time.deltaTime*100);
    }
}
```

Script

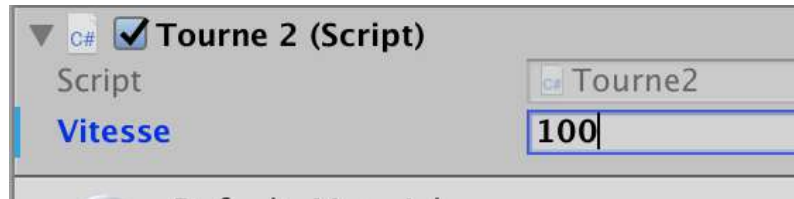


Principe d'encapsulation :



Equivalent des propriétés : [SerializeField]

Demo Cube :



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

public class Tourne2 : MonoBehaviour
{
    [SerializeField] // on peut mettre un nom mais par défaut c'est le nom du champ privé
    private int vitesse;
    // Start is called before the first frame update
    void Start()
    {






    }

    // Update is called once per frame
    void Update()
    {
        transform.Rotate(Vector3.up * Time.deltaTime * vitesse);
    }
}
```

Script

- **Principe d'encapsulation :**
 - L'utilisation des propriétés est possible voir nécessaire pour dialoguer entre scripts
 - Il peut s'avérer nécessaire de faire un constructeur
- **Singleton :**
 - Utile pour conserver des données
 - Evite les variables statiques

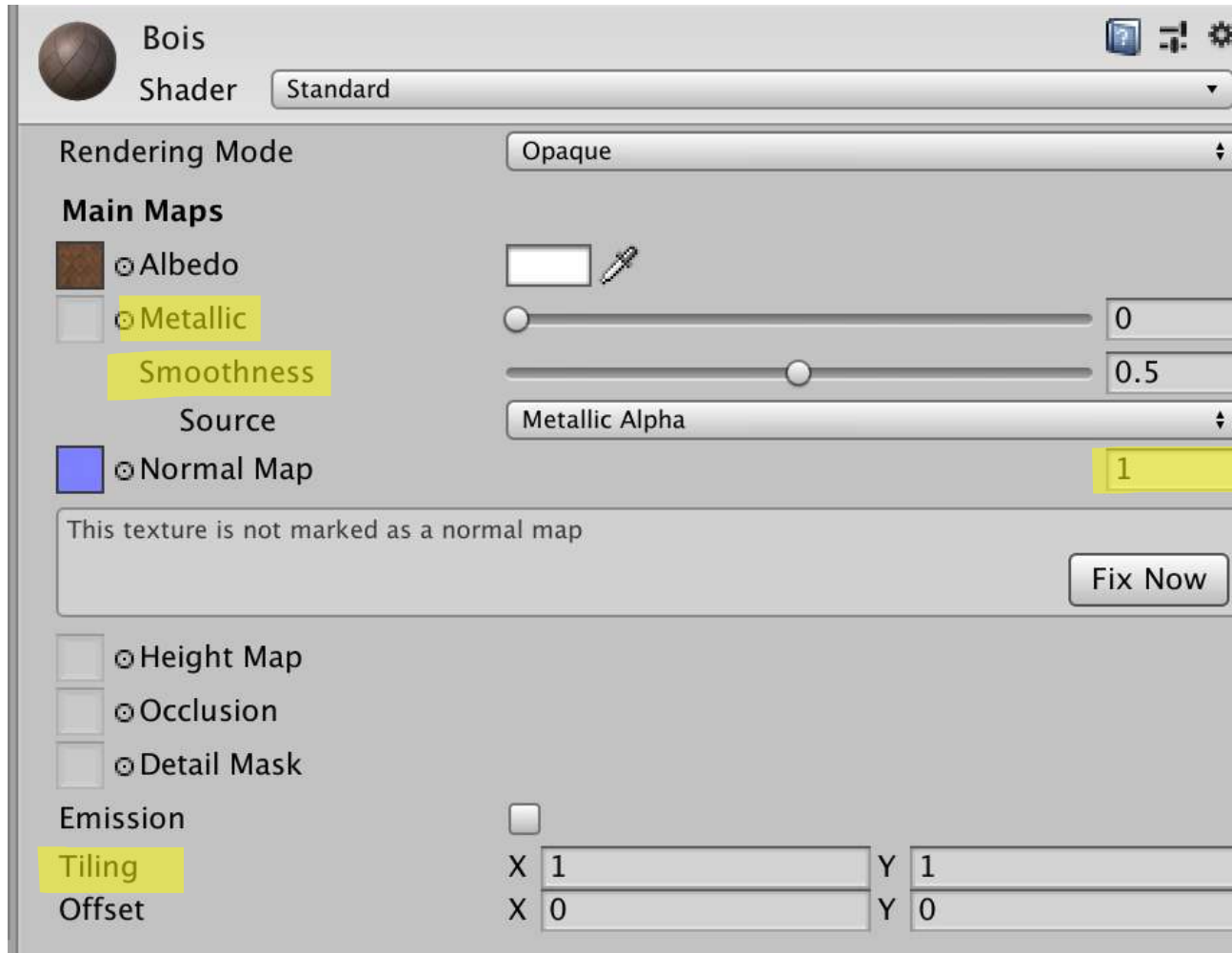
Textures / Material

-  Les textures ne sont simplement des images collées sur un gameObject.
-  On utilise au moins une normal Map pour définir un effet 3D
-  On crée ainsi un Matériel (Material) constitué d'une texture de base (albedo) et d'une normal Map (il existe d'autres techniques). Le Material utilise un shader (code très complexe) qui permet d'appliquer la texture sur l'objet 3D et ainsi que la gestion de la lumière.
-  Quand on utilise des textures, on charge aussi des normal Map avec.
-  On peut créer des normal Map avec Gimp par ex ou des outils en ligne : <https://cpetry.github.io/NormalMap-Online/>

Textures / Material

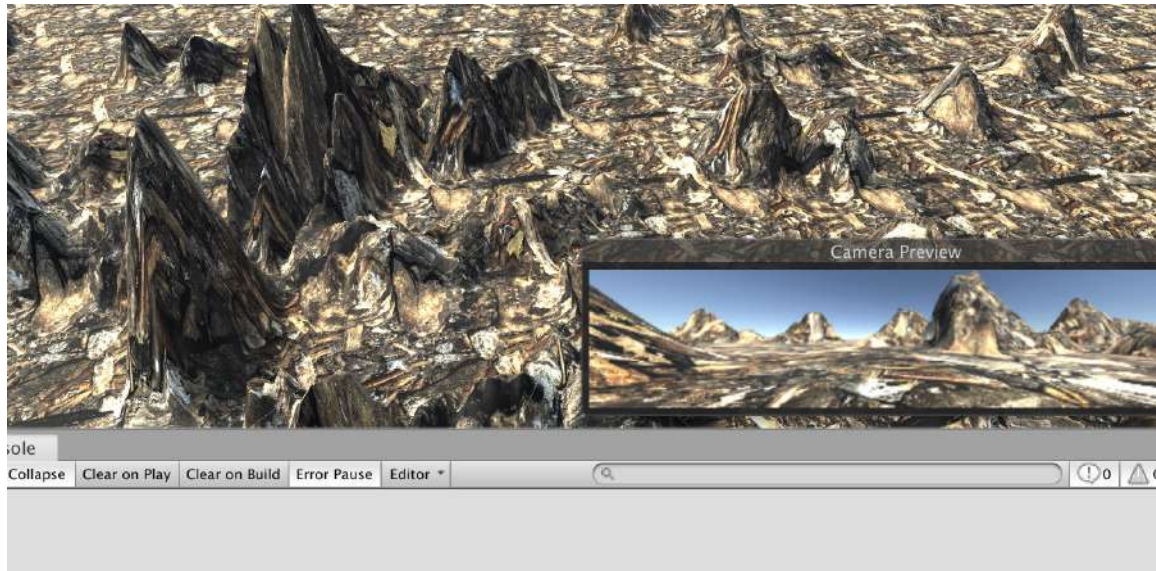


Ex : Texture Bois



Terrain

- Le GameObject terrain permet de générer un terrain :



- Possible importation Blender : direct ou en fbx
- Intégration avec SketchUp 3D :

https://www.youtube.com/watch?v=kVz8u_1Y7fQ

Asset et Asset Store



Asset :



C'est un package qui contient tout et n'importe quoi !



Asset Store est le shop de Unity :

 **AssetStore**



Search for assets



Il permet d'importer des Assets développés par d'autres personnes : très utile !



Ils contiennent généralement une scène de démo

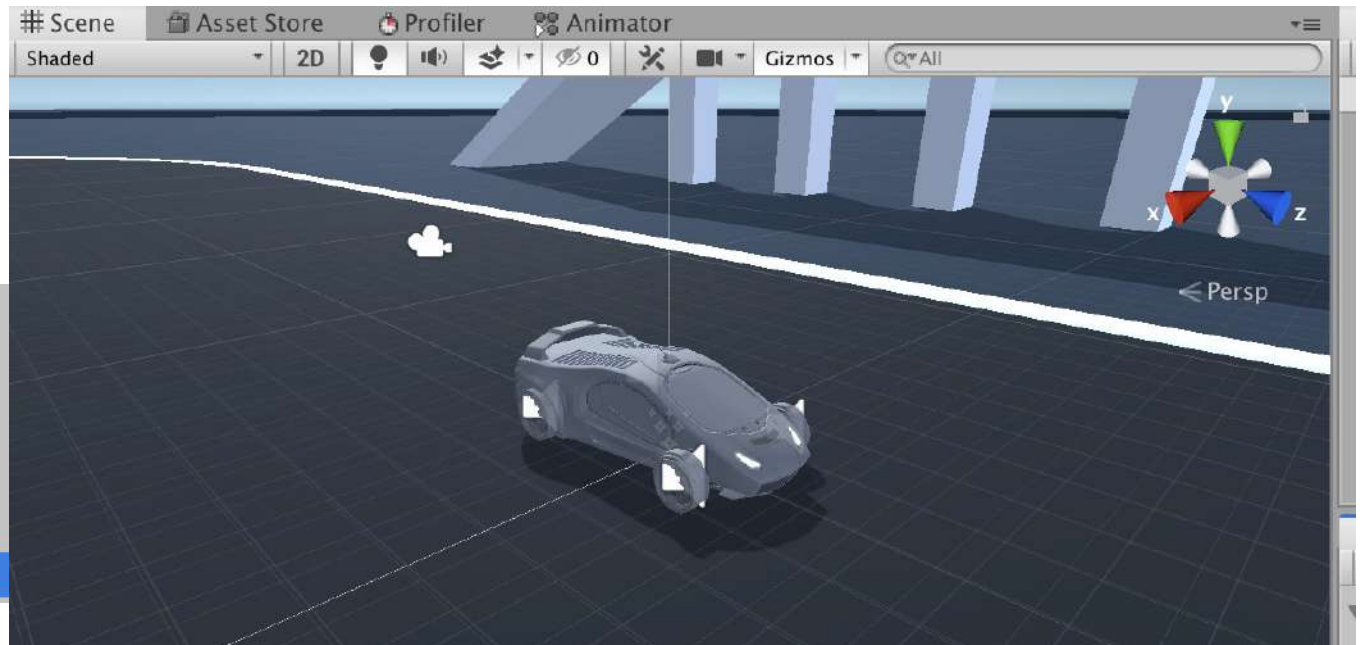
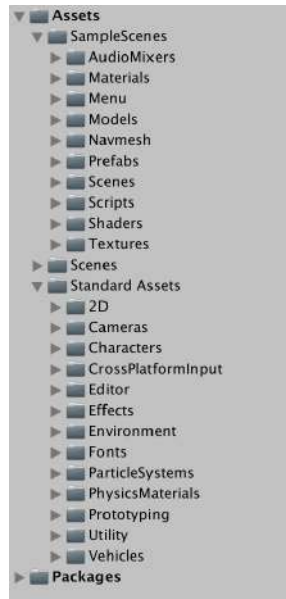


Asset indispensable : standard Assets de Unity
(Starter Asset plus récent)



Les outils les utiles sont les contrôleurs d'objet

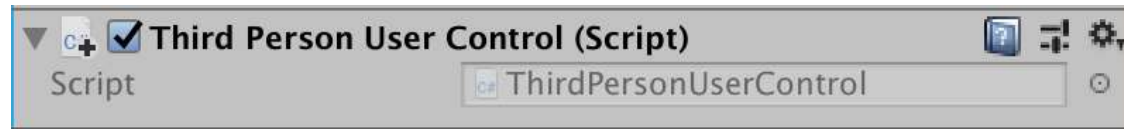
Asset et Asset Store



Ex : Contrôleur

 **Demo** : Déplacer un cube en utilisant un contrôleur

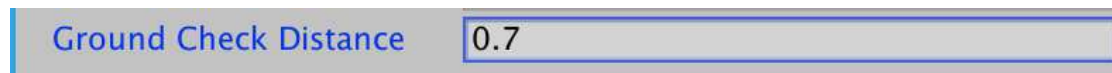
 Ajouter un Third Person User Control



 Modifier l'Animator



 Modifier le Ground Check Distance (pour éviter des bug de contrôle) :



 Le cube se déplace comme un personnage avec détection de collision !

Modèles 3D

De nombreux modèles disponibles dans l'Asset Store :

Pack de particules standards :



UNITY TECHNOLOGIES

Unity Particle Pack

Tutorial Projects



De nombreux modèles disponibles sur le Web en import (import new Asset) :

<https://free3d.com/fr/3d-models/>

Demo (Assets) : Loup (import Package par ex)

→ réutiliser script rotation avec gameObject nul comme parent

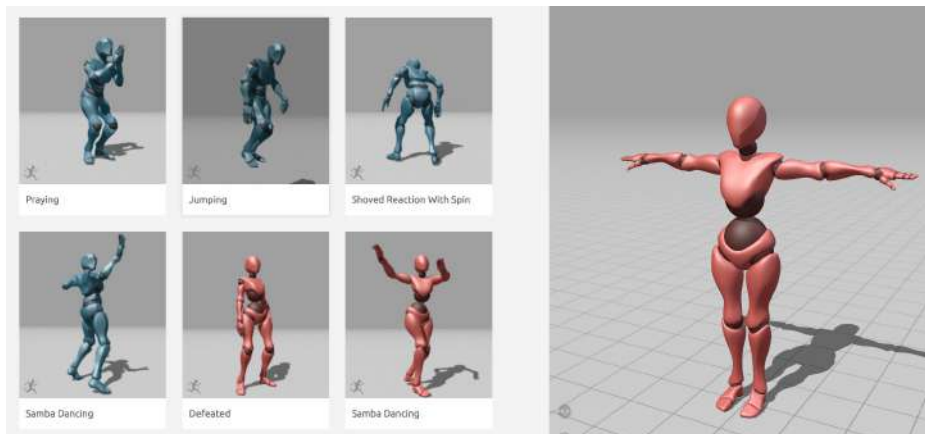
Modèles 3D

On peut aussi créer ses propres modèles en 3D

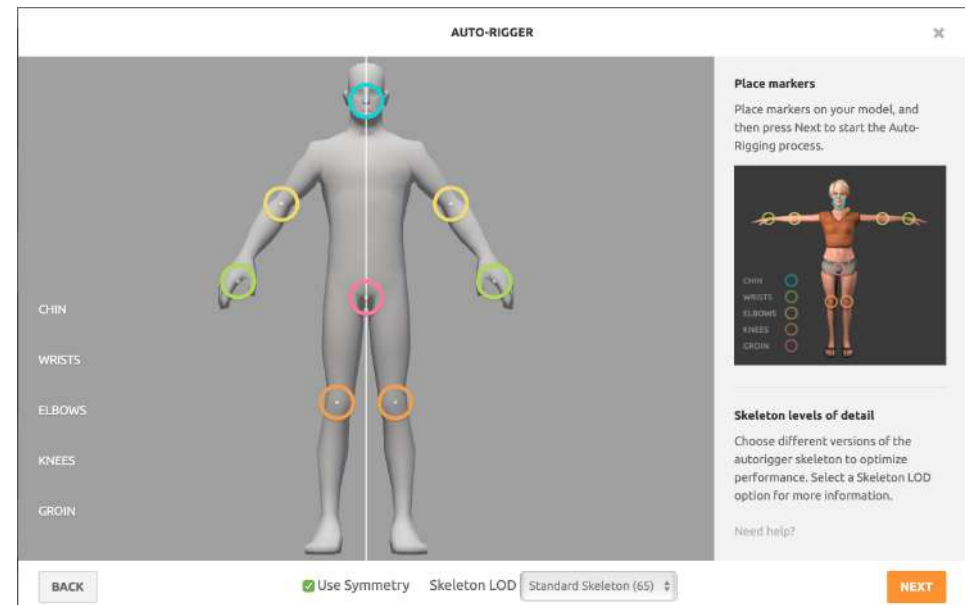
Blender

C'est un travail long et fastidieux car pour un personnage on doit créer un rigging pour l'animation

Mixamo :



Autorigging possible (upload)
très efficace



MagicaVoxel https://www.youtube.com/watch?v=kT6_loHgoH4&feature=youtu.be

Important : pour animer on utilisera un animator

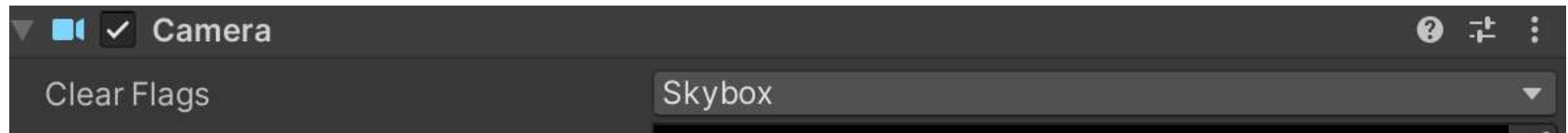
Skybox

- 🌐 Le fond du jeu peut être complété avec un skybox pour rendre le rendu plus réaliste :



Skybox

- 🌐 Il suffit d'importer un skybox depuis l'Asset Store et de l'appliquer sur la caméra :

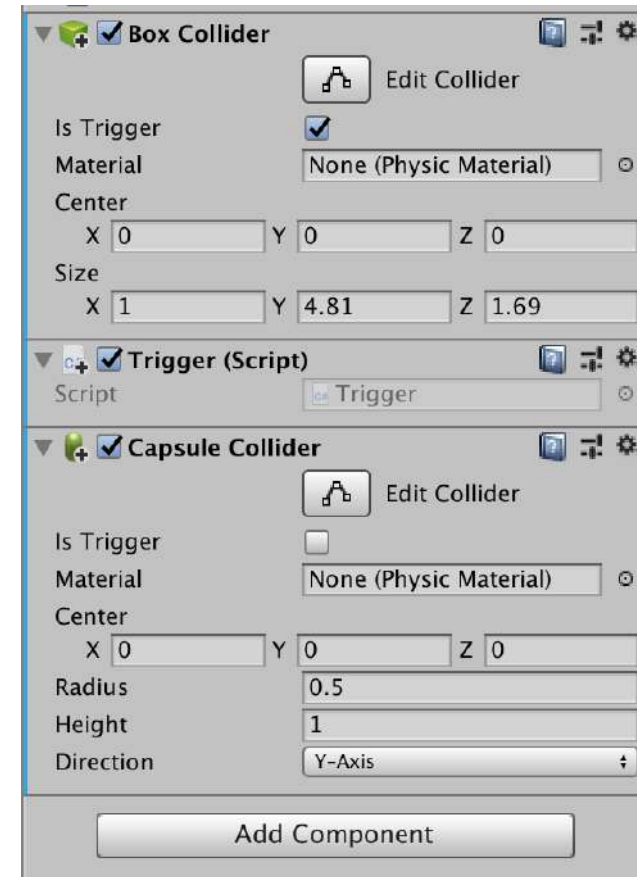
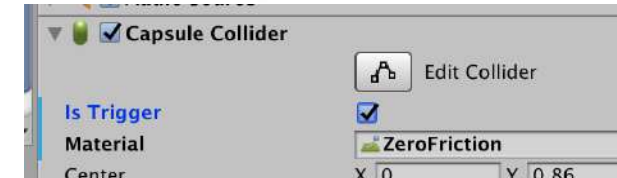
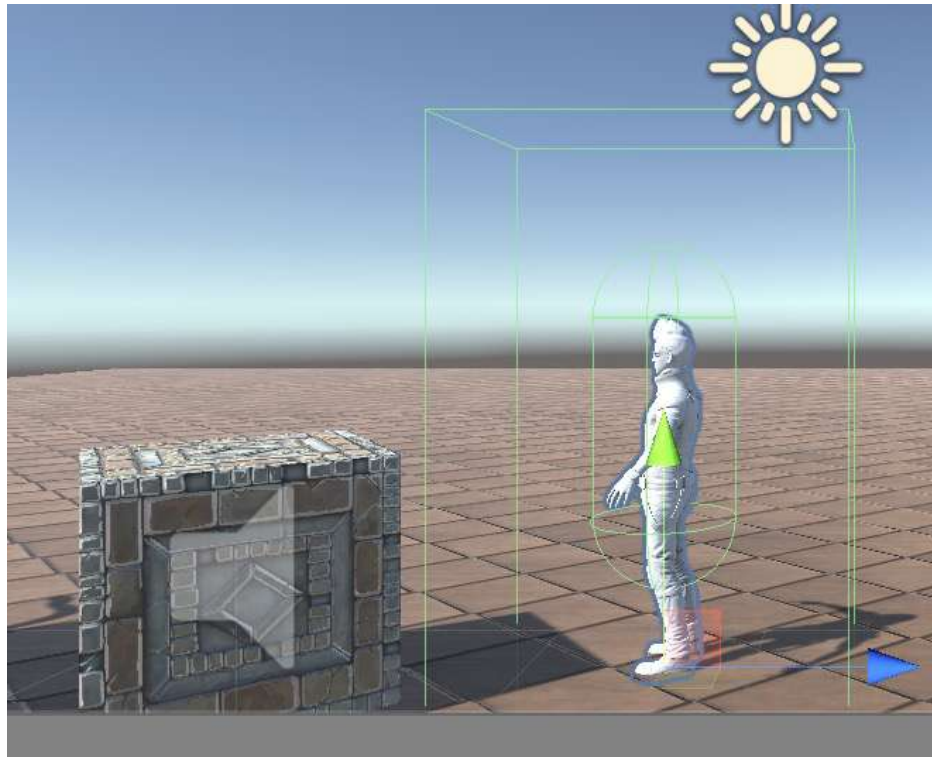


- 🌐 Générateur de skybox :

<https://skybox.blockadelabs.com/>

Collision / Trigger

- **Collider** : collision physique (Zero friction évite de glisser)
- **Trigger** : collision de zone (ex : dialogue)
- **Détection de collision sur un collider**
- **Éviter le mesh collider très gourmand en ressources**



Collision



Méthodes de collision :

<u>OnCollisionEnter</u>	OnCollisionEnter is called when this collider/rigidbody has begun touching another rigidbody/collider.
<u>OnCollisionExit</u>	OnCollisionExit is called when this collider/rigidbody has stopped touching another rigidbody/collider.
<u>OnCollisionStay</u>	OnCollisionStay is called once per frame for every collider/rigidbody that is touching rigidbody/collider.
<u>OnTriggerEnter</u>	OnTriggerEnter is called when the Collider other enters the trigger.
<u>OnTriggerExit</u>	OnTriggerExit is called when the Collider other has stopped touching the trigger.
<u>OnTriggerStay</u>	OnTriggerStay is called almost all the frames for every Collider other that is touching the trigger. The function is on the physics timer so it won't necessarily run every frame.

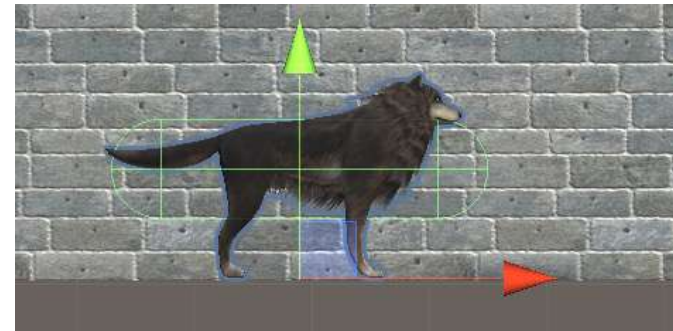
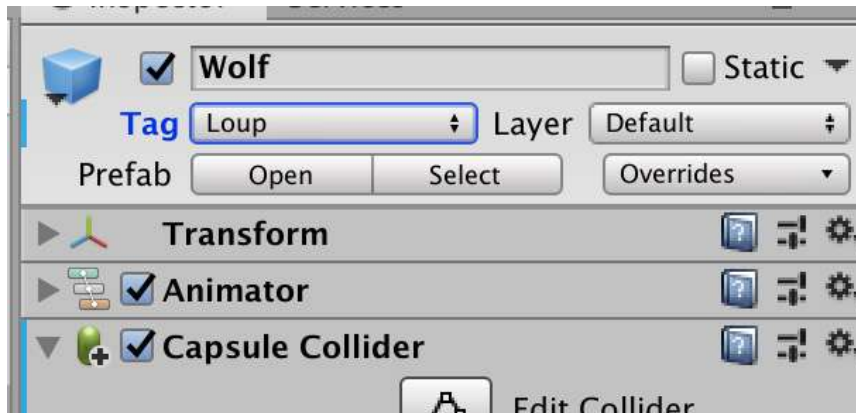
Collision



Demo (Assets) :



Capsule collider avec un tag pour le loup :

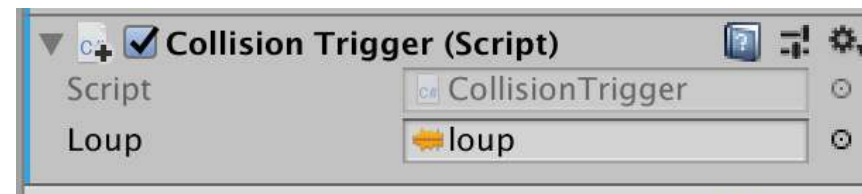


avec du son (utiliser des wav) !

https://www.sound-fishing.net/bruitages_combat.html



Script de détection :



Collision

```
AudioSource m_Source;

[SerializeField] AudioClip m_Loup;

// Start is called before the first frame update
void Start()
{
    m_Source = GetComponent();
}

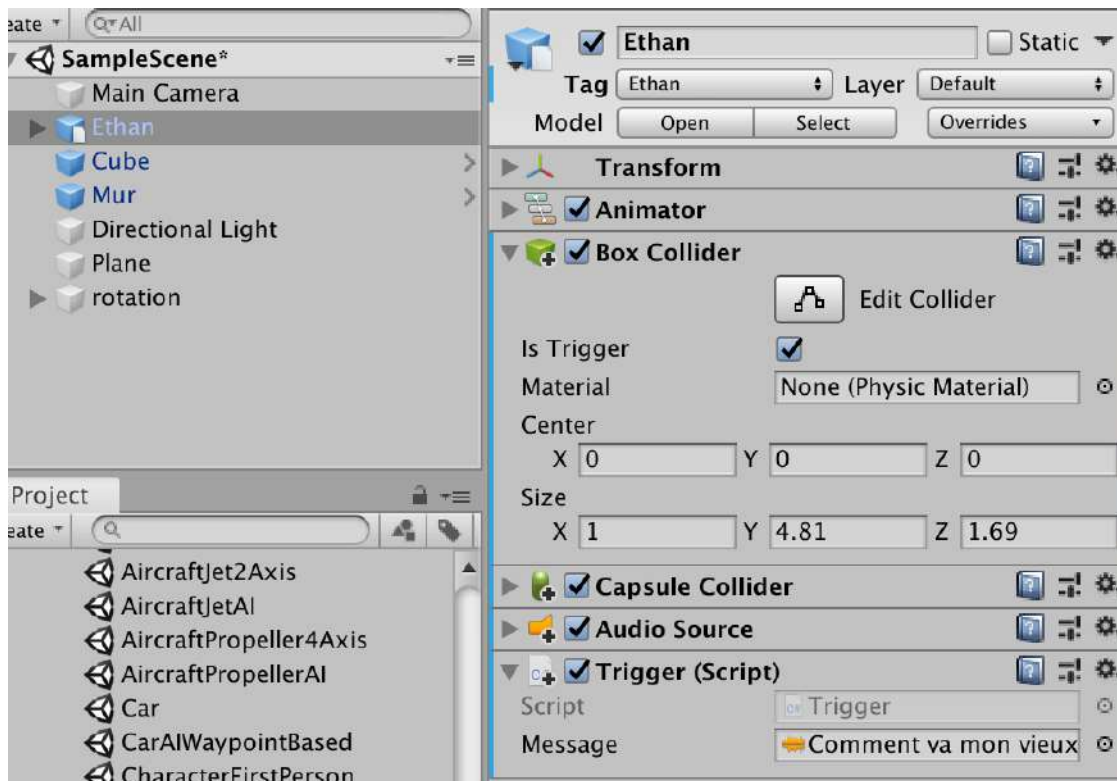
// Update is called once per frame
void Update()
{
}

private void OnCollisionEnter(Collision collision)
{
    if (!m_Source.isPlaying)
    {
        if (collision.gameObject.tag == "Loup")
        {
            m_Source.clip = m_Loup;
            m_Source.Play();
        }
    }
}
```

Trigger



Démo : Trigger sur le personnage



```
void OnTriggerEnter(Collider other)
{
    Debug.Log("Trigger");
    if (!m_Source.isPlaying)
    {
        if (other.gameObject.tag == "Cube")
        {
            m_Source.clip = m_Message;
            m_Source.Play();
        }
    }
}
```

Input

 **Gestion du clavier :**

Ex : touche ESC

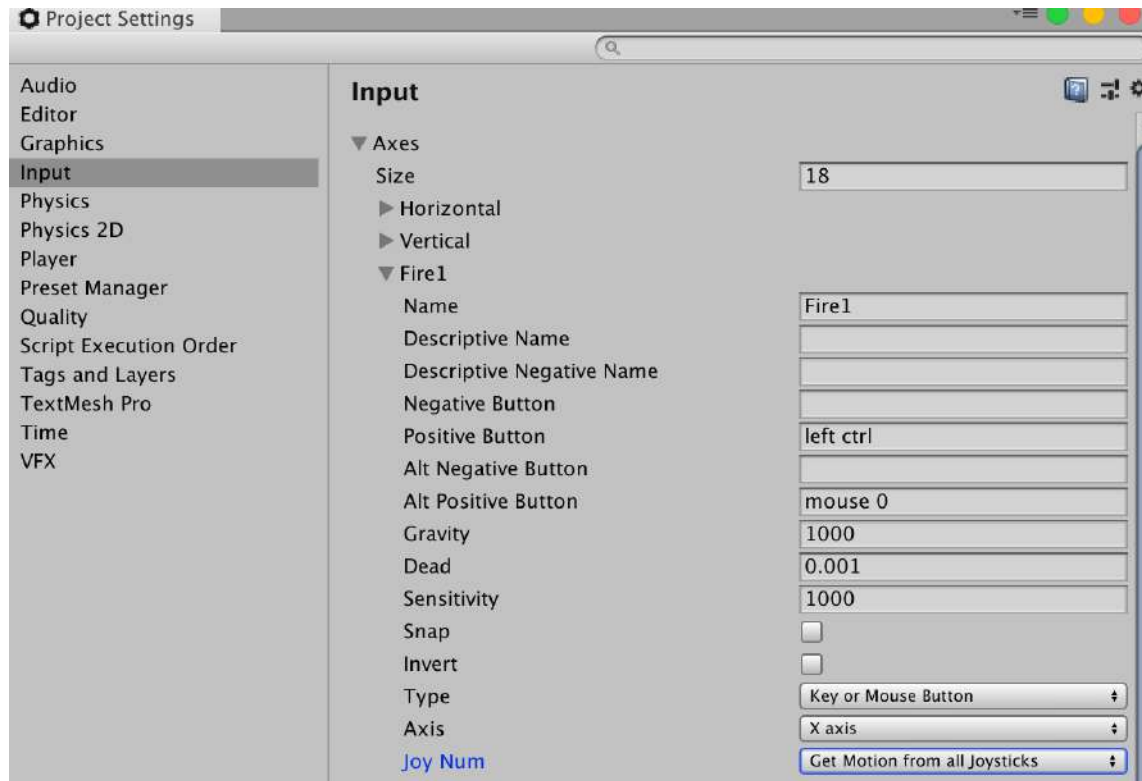
```
public class ExampleClass : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKey("escape"))
        {
            Application.Quit();
        }
    }
}
```

Rq : Build nécessaire pour voir le résultat

Input prédéfinis



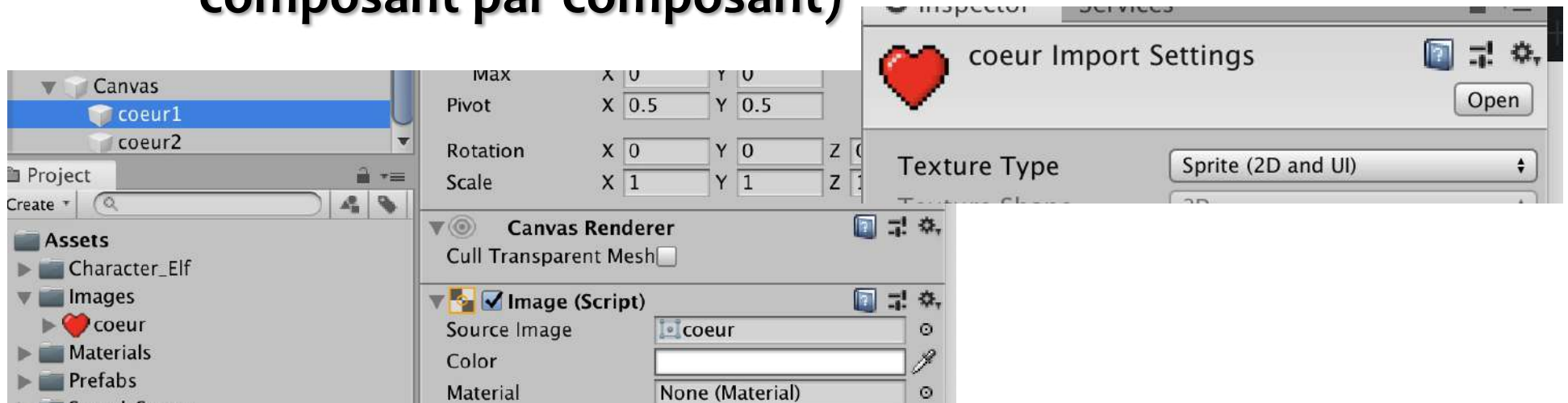
Dans Project Setting des input sont déjà définis :



On peut donc les utiliser voir les redéfinir

UI

- 🌐 **User Interface : 2D → GameObject UI**
- 🌐 **Beaucoup de composants (voir la doc)**
- 🌐 **Utile pour les menus, scores, sous-titres textes ...**
- 🌐 **Représente un grand rectangle virtuel en superposition (l'alignement peut se paramétrer composant par composant)**



Animation



Animator



Animation pour de la cinématique par ex. On pourra alors déclencher sur certains événements (cf dernier exercice)

Méthode OnGUI



Gère les évènements GUI :

Demo (InputUIScene) :

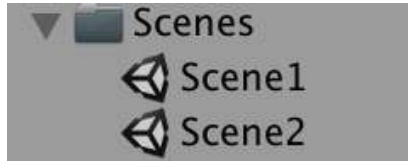
```
[SerializeField] GameObject m_GO;  
  
// Display the changing position of the sphere.  
void OnGUI()  
{  
    GUIStyle fontSize = new GUIStyle(GUI.skin.GetStyle("label"));  
    fontSize.fontSize = 24;  
    fontSize.normal.textColor = Color.red;  
    GUI.Label(new Rect(20, 20, 300, 50), "Position: " + m_GO.transform.position.z, fontSize);  
}
```

RayCast

- Faisceau lumineux allant en ligne droite sur une distance donnée.
- Permet de vérifier s'il rentre en contact avec un autre objet, un tag ou des calques.
- Le Raycast permet donc de :
 - Vérifier une distance entre le joueur et un objet/sol
 - Créer un système de visé pour des armes
 - Améliorer les collisions.
- Demo : **DemoGIT**
- <https://youtu.be/vzlvxuxNgK4>

Chargement de scène

 On place nos scènes dans le dossier scène :



 Attention les scènes doivent être dans le build :

Scenes In Build	
<input checked="" type="checkbox"/> Scenes/Scene1	0
<input checked="" type="checkbox"/> Scenes/Scene2	1

 Le changement de scène se fait en nommant la scène ou en utilisant son index :

```
if (Input.GetKeyDown(KeyCode.A))
{
    SceneManager.LoadScene("Scene2");
}

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
```

Chargement de scène

- Conservation des objets d'une scène à l'autre :
- Ex simple avec un script sur les objets à conserver

```
public class DontDestroy : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        Object.DontDestroyOnLoad(this);
    }

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

- Ce mode de chargement est synchrone, cela pose des soucis (pour le clipping par ex)

Unity3D / Debogage

🌐 Débogage

🌐 Classe Debug

- 🌐 Debug.log (infos)
- 🌐 Debug.LogWarning (Warning)
- 🌐 Debug.LogError (Erreurs)
- 🌐 Meme principe que la console javascript

🌐 Console avec filtrage :



🌐 Log dispo à l'exécution de l'exé localement :

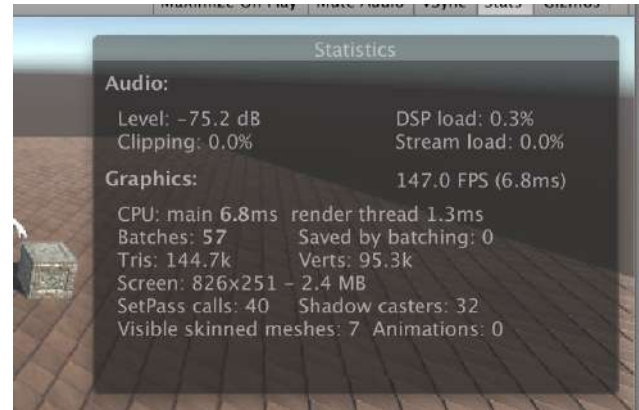
<https://docs.unity3d.com/Manual/LogFiles.html>

🌐 Debugueur VisualStudio

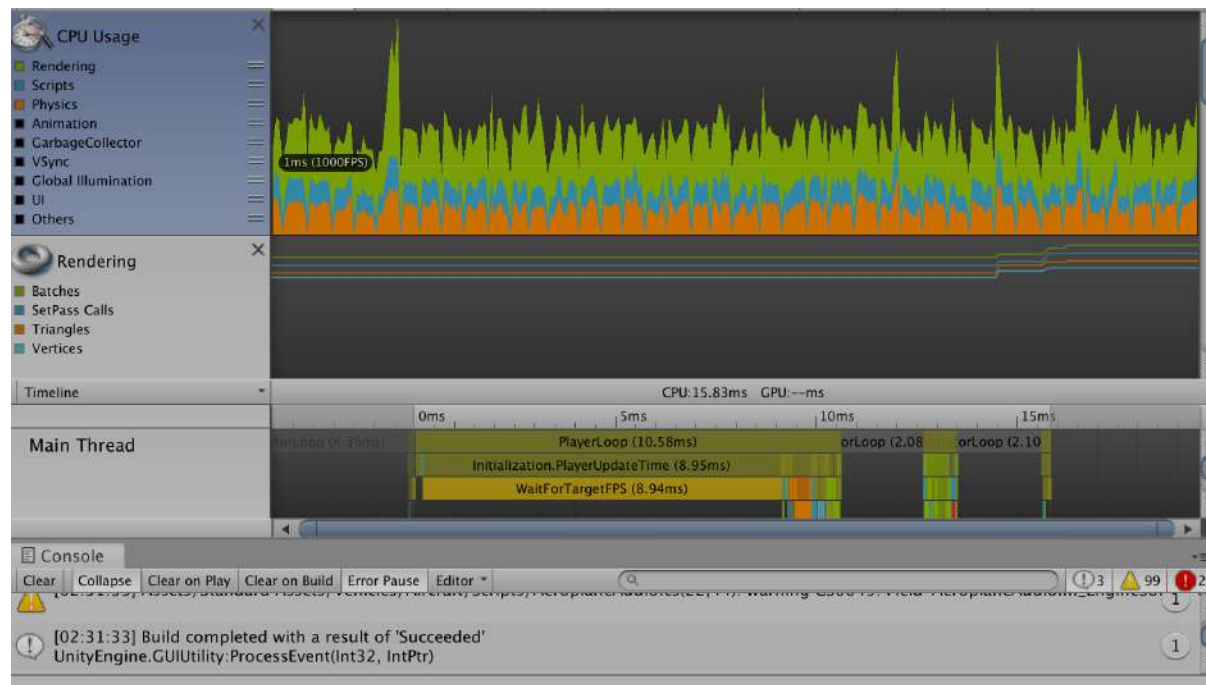
Unity3D / Mesures



Stat de base



Stat très élaborées



Travail collaboratif



Plastic SCM (3 utilisateurs + 5Go gratuit)

- <https://www.plastic SCM.com>
- <https://www.youtube.com/watch?v=5jCCpoQdyME>



Git (pas de limite mais un peu de configuration)

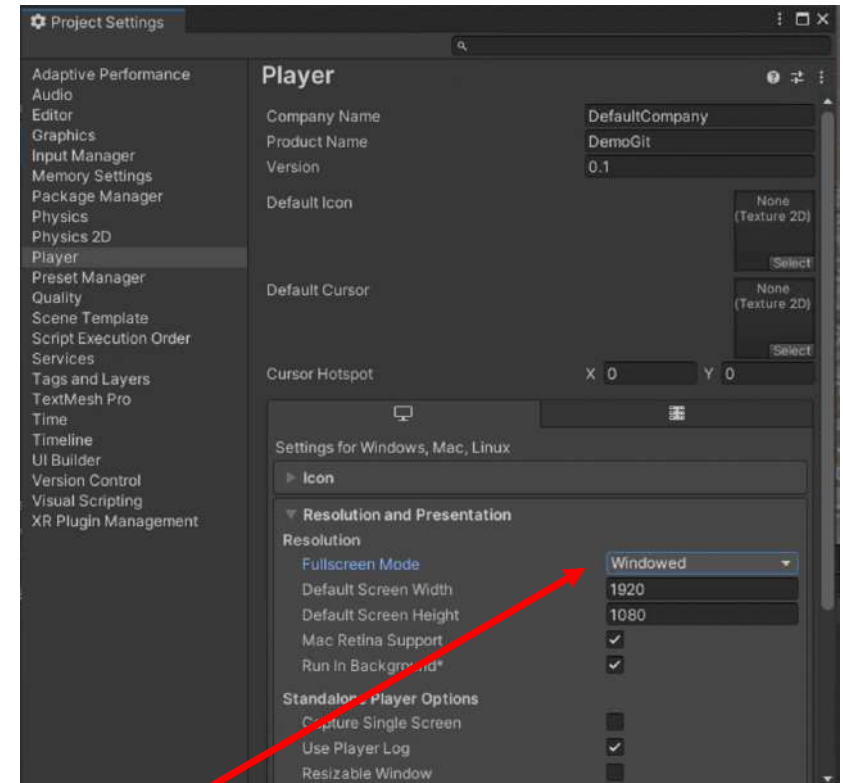
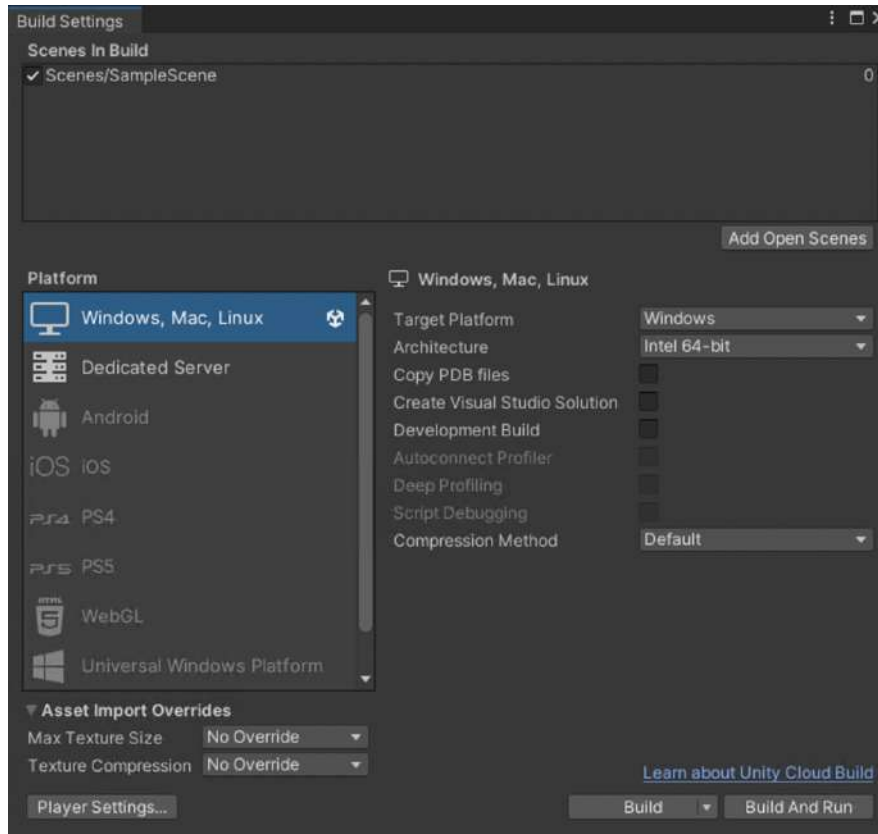
- <https://www.youtube.com/watch?v=PJwXxtJDDZQ>

Dans la mesure du possible ne pas travailler en même temps sur la même scène (cela reste possible). Utiliser un compte GitHub

Conseil : installer GitHub Desktop (possible en local sur les PCs de l'IUT)

Demo : DemoGIT

Build



- Ne pas oublier d'ajouter les scènes.
- Il est possible de configurer en mode fenêtré dans les paramètres du projet
- Création d'un exe et d'un répertoire de données

DemoGit_Data
MonoBleedingEdge
DemoGit.exe
UnityCrashHandler64.exe
UnityPlayer.dll

UNITY3D - ressources



Vidéo rapide sur les bases :



<https://www.youtube.com/watch?v=vTBVlxK2xdk>



Documentation officielle indispensable avec de nombreux exemples :



<https://docs.unity3d.com/ScriptReference/>



Tutoriels de UNITY3D (long) :



<https://unity3d.com/fr/learn>