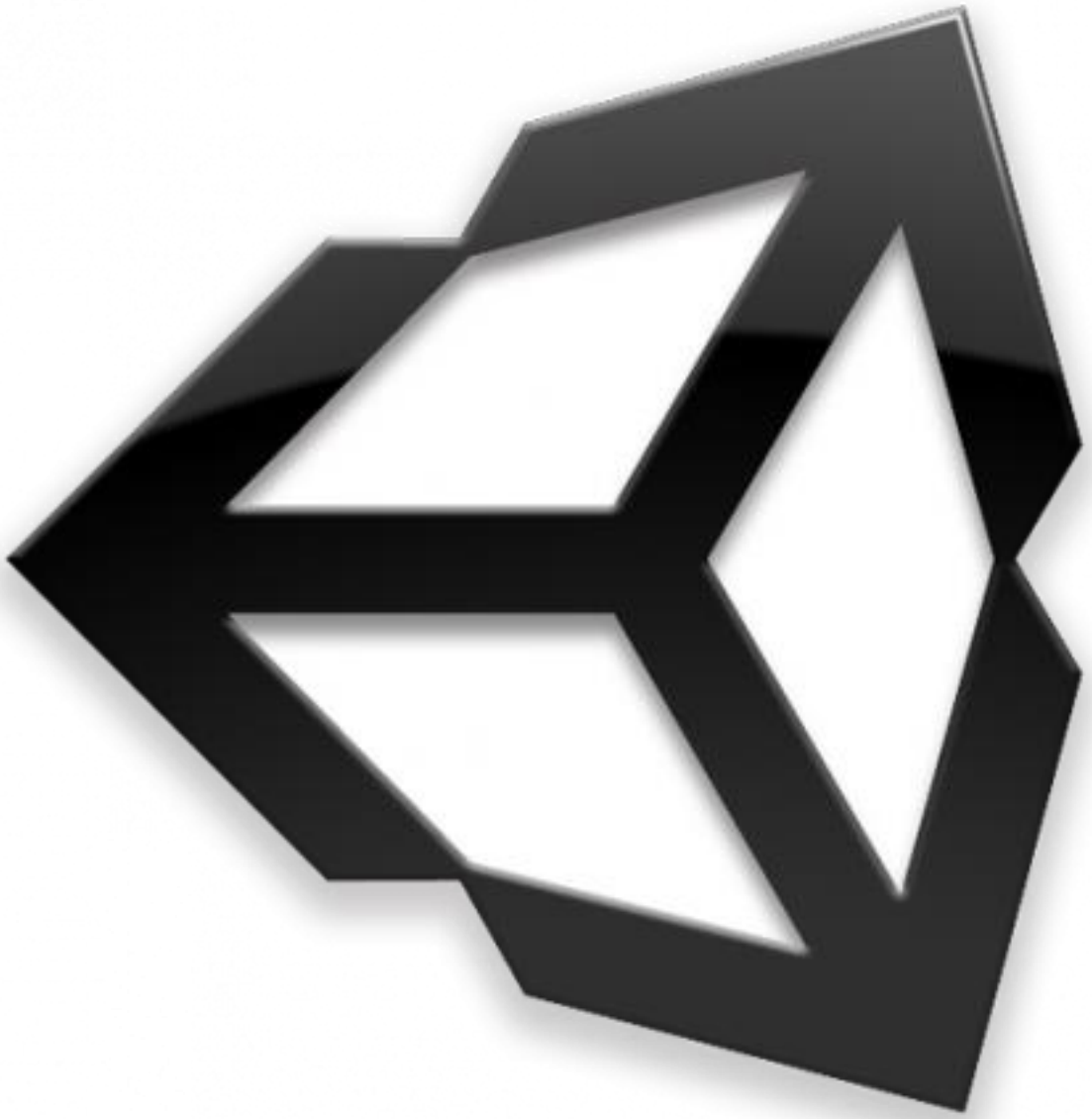


Démarrer avec Unity3D

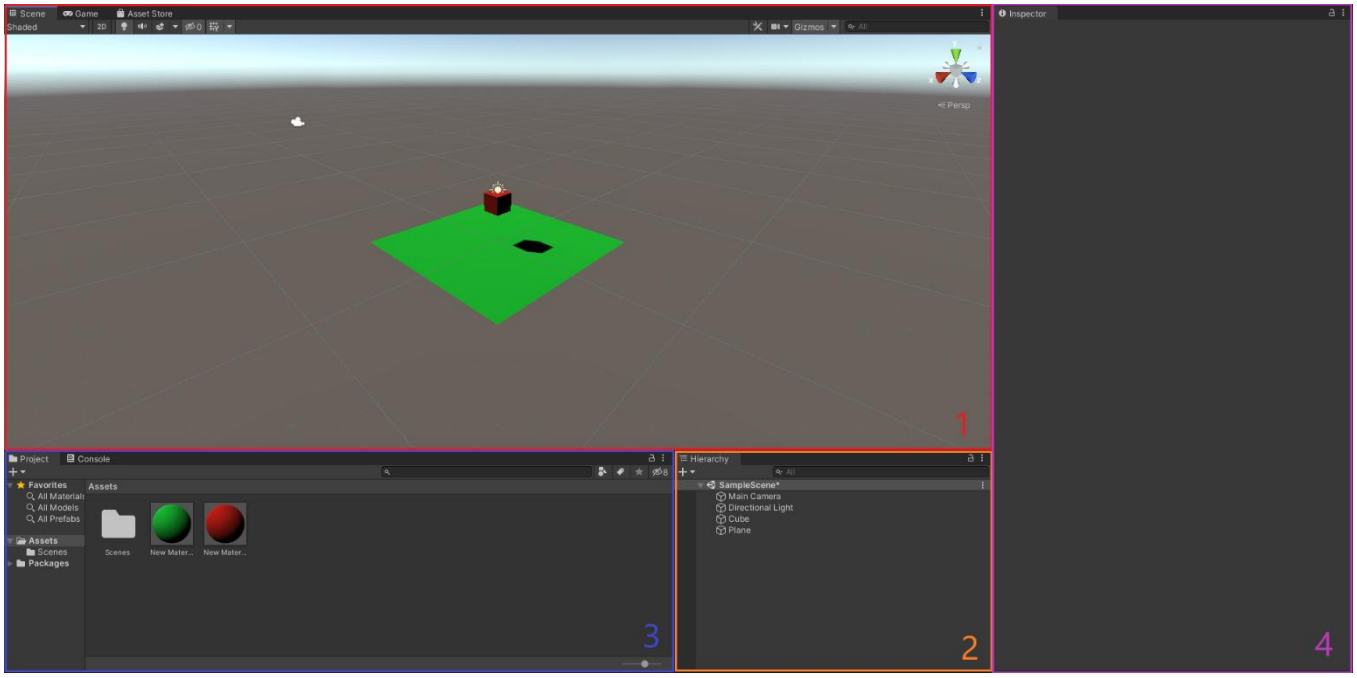
Introduction

Unity3D est un moteur de jeux vidéo qui permet de créer des applications en 2D ou 3D. Très réputé pour sa prise en main facile et rapide, il sert à faire des jeux souvent multi-plateforme (PC, Android, Xbox, ect…) assez simplement.

En plus de cela, la documentation et la communauté actives vous permettront de continuer et de vous améliorer sur ce logiciel assez facilement chez vous.

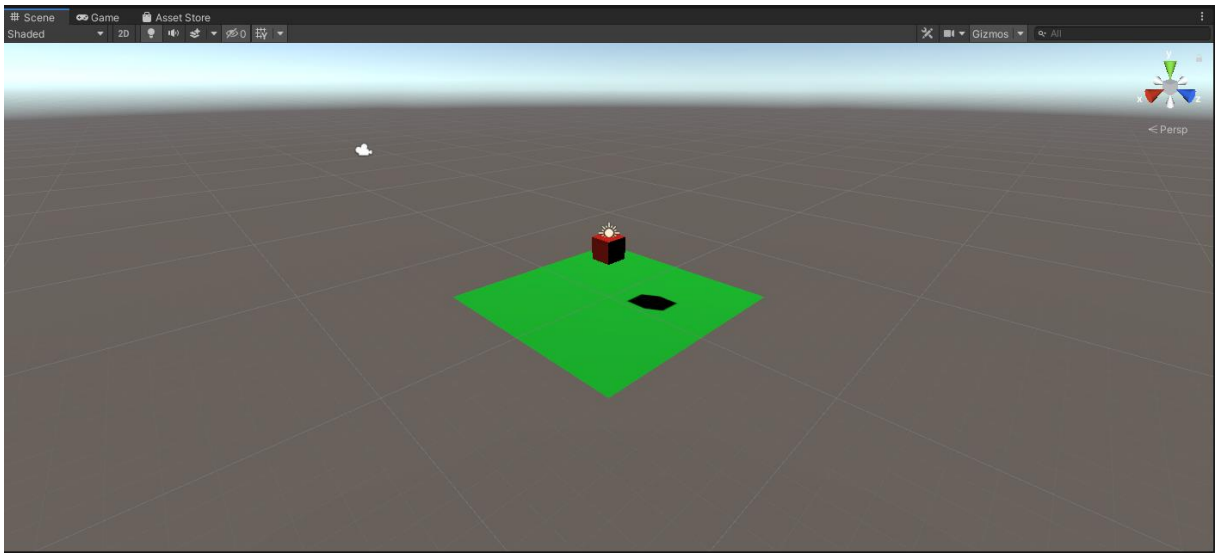
Découverte de l'interface

Pour pouvoir prendre en main plus facilement le logiciel nous allons découper en quatre parties l'interface.



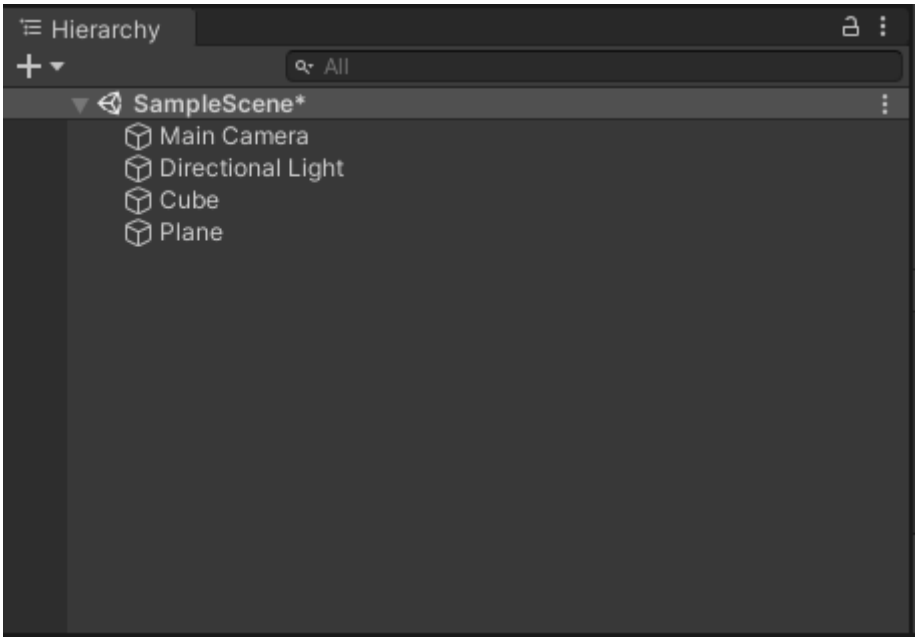
La Scène (1) :

L’interface que vous voyez au milieu de toutes autres c’est votre scène elle peut être en 2D ou en 3D. Dans celle-ci vous voyez les éléments qui compose votre scène vous pouvez interagir avec eux.



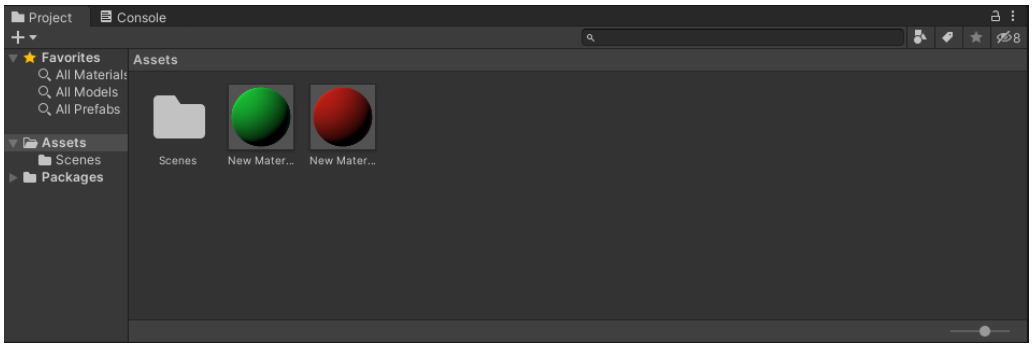
La Hiérarchie (2) :

Ici vous pouvez voir tous les objets qui sont dans votre scène ainsi que leurs sous-éléments.



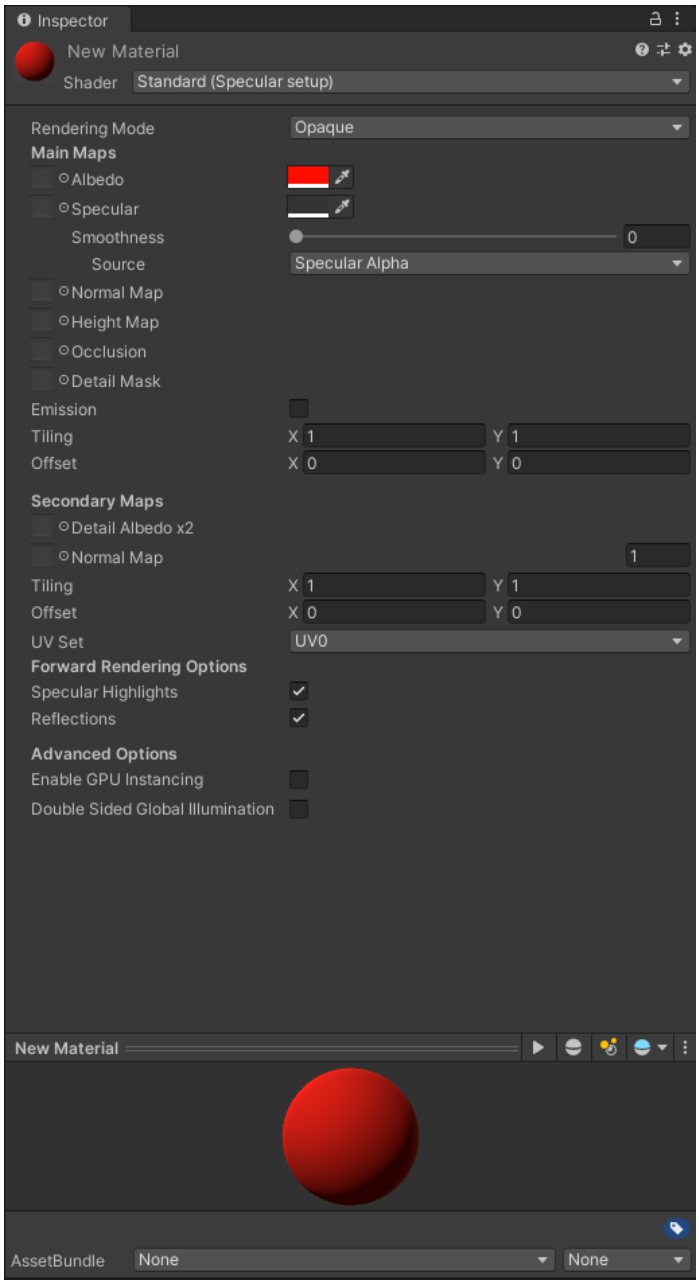
Les Assets (3) :

Cette zone est réservée à tous vos Assets. C'est-à-dire tous vos objets qu'ils soient « visuels » ceux de la scène comme ceux que vous aimeriez garder pour de futur projet ou ceux que vous téléchargé. Il y a aussi vos scènes, vos scripts et vos textures qui sont visibles depuis cette interface.



L’inspecteur (4) :

C’est la partie qui influe directement sur les composants de votre scène c’est la ou l’on peux voir les propriétés de nos objets sans avoir besoin de passer par le code.



Prise en main

Maintenant que vous connaissez mieux vos outils nous pouvons commencer à les utiliser. Dans ce tutoriel nous allons apprendre à faire bouger un cube.

Création d'objet et physique

Vous connaissez tous les effets de la gravité, dans Unity nous pouvons changer les paramètres de la physique pour la rendre plus ou moins proche de la réalité. Mais pour cela il nous faut des objets dans notre scènes.

Pour ce faire, faites un clic droit dans votre hiérarchie et sélectionnez : **3DObjects -> Cube**. Suite à cela créez un plan de la même façon.

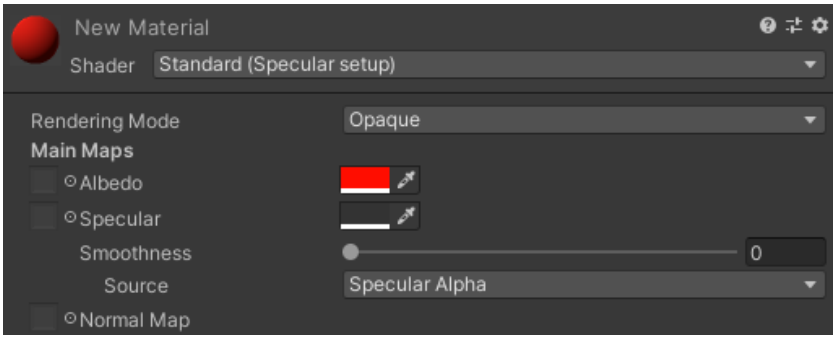
Si vous lancez la scène rien ne se passe (Pour le moment) !

Maintenant cliquez sur votre cube, ses propriétés s'affichent dans l'inspecteur. Il y a un bouton « **Add Component** », cliquez dessus et recherché « **Rigidbody** » ce component sert à gérer la physique tel que la gravité et la masse de votre objet.

Relancer la simulation, cette fois-ci le cube tombe.

Maintenant rajoutons un peu de couleur a cette scène blanche. Pour se faire c'est très simple, il vous suffit de faire un clic droit dans la partie « Assets » puis **Create -> Materials**.

Vous n'avez plus qu'a modifié la couleur de votre Material.



Rajoutez votre nouveau composent sur le cube en le glissant dans l'inspecteur quand le cube est sélectionné.

Déplacement et Code

Dans Unity il y a plusieurs langages qui peuvent être utilisé, le plus répandu est le C# (C Sharp). Nous allons l'utiliser pour faire bouger le cube que l'on vient de créer.

Pour se faire nous allons créer un nouveau script, allez dans « Assets » cliquez droit et créer un script C#. Faites un double-clique dessus il devrait s'ouvrir (dans Visual Studio) à chaque fois que vous le sauvegarderez il devrait s'actualiser dans Unity.

Lors de la création Unity vous mets directement à disposition deux méthodes ainsi qu'une classe qui contient ces deux méthodes.

Start() est une méthode qui est appelé par Unity lorsque l'objet devient actif dans votre scene. C'est-a-dire qu'elle est appelé au démarrage de votre scène pour tous vos objets « visibles ».

Update() sera une méthode qui sera appelé en boucle tant que votre programme tourne.

Maintenant que vous avez votre script et que vous connaissez les deux méthodes qui sont dedans nous pouvons commencer.

Première étape créer une variable, celle-ci servira à récupérer la propriété **RigidBody** que nous avons mis précédemment sur le cube.

```
    private Rigidbody rb;
```

Nous la mettons en « private » car nous voulons qu'elle ne soit pas récupérable par d'autre script car nous n'en avons pas besoin hors de se script.

À la suite de cela vous allez créer une variable publique de type « float » nommée « speed ».

Cette variable servira à configurée la vitesse a laquelle vous voulez bouger votre objet. Le fait qu'elle soit en public nous permettra de la modifié directement de puis l'inspecteur d'Unity sans avoir à changer sa valeur depuis le code. Si vous le souhaitez vous pouvez quand même lui mettre une valeur

par défaut cependant en C# il faut mettre un « f » après un nombre à virgule tel qu'un « float ».

```
public float speed = 0.1f;
```

Après la création de cette variable nous pouvons enfin essayer de récupérer le component « **RigidBody** » depuis le code.

Vous utiliserez la fonction **GetComponent<type>()** dans le Start() pour pouvoir le récupérer uniquement au lancement de votre programme.

Maintenant que vous avez votre « **RigidBody** » dans le code et votre vitesse nous allons faire bouger le cube.

Nous voulons que lorsque vous appuyer sur « Z » le cube avance. Pour ça nous allons ajouter dans Update() une condition pour que quand l'Input « Z » soit actif, l'action se fasse.

```
void Update()
{
    if (Input.GetKey(KeyCode.Z))
        rb.transform.Translate(Vector3.forward * speed);
```

Quand le bouton « Z » est appuyé, nous appliquons au « RigidBody » un mouvement de translation vers l'avant.

Lancer votre programme et appuyer sur Z, votre cube avance !

Vous pouvez maintenant ajouter les trois autres directions (left, right, back).

Après l'avoir fait bouger on va voir comment faire sauter votre cube avec la fonction **AddForce()**. Faire une translation sur un objet c'est sympa mais vous allez appliquer une force dessus.

```
rb.AddForce(Vector3.up * JumpForce, ForceMode.Impulse);
```

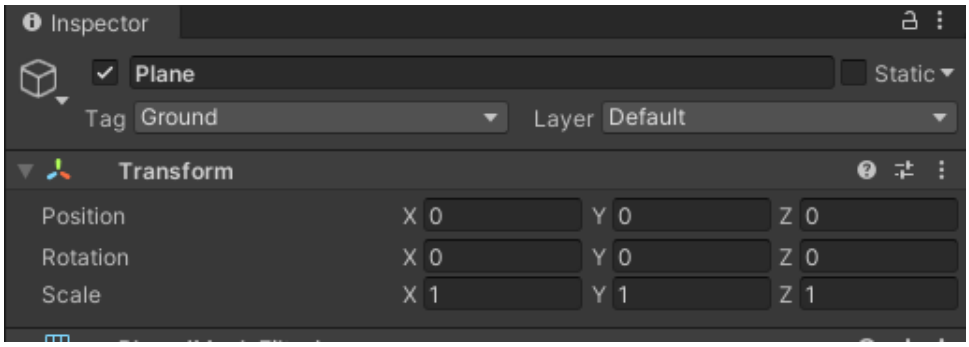
Cette fois-ci, a vous de jouer. Choisissez la touche qui va vous faire sauter ainsi que la force de votre saut représentée par la variable « JumpForce ».

Démarrer votre programme, votre cube peut facilement bouger peut-être trop facilement.

Vous pouvez sauter à l'infini, vous aimeriez sauter juste une fois ou deux après avoir quitter le sol mais pour cela il faut définir ce qu'est le sol.

Tag et Collision

Allons définir votre sol pour cela retourné dans Unity et sélectionner votre plan. Dans l'inspecteur nous allons créer un « tag » pour celui-ci.



Créer votre tag « Ground » et appliquez le sur votre objet vous servant de sol.

Nous pouvons retourner dans notre code, créez un nouveau script et dans celui-ci ajoutez une fonction avec ce prototype :

Void OnCollisionEnter(Collision collision) { }

Cette fonction ne se fait appelé que lorsque l'objet qui est associé est en collision avec un autre objet. Maintenant il faut rajouter une condition pour que quand notre objet soit en collision avec un objet avec un tag « Ground » un booléen « isGrounded » soit à true.

```
0 références
private void OnCollisionEnter(Collision collision)
{
    if (collision.tag.Equals("Ground"))
    {
        isGrounded = true;
    }
}
```

Vous devez maintenant changer votre condition de saut pour qu'elle prenne en compte le fait qu'on touche le sol (n'oubliez pas de dire au moment du saut que le sol n'est plus en collision le code ne le fait pas tous seul).

Voila maintenant que votre cube bouge et saute vous pouvez vous amusez à changer l'angle de vue, attacher votre caméra à votre cube pour une vue (FPS). Vous pouvez vous amuser à changer la valeur des variables ou bien même à faire que le cube bouge selon l'orientation de la souris.

Merci pour avoir suivi ce tutoriel, vous avez maintenant une base de connaissance un petit peu plus grande. N'hésitez pas à faire des choses plus complexes avec Unity. Nous sommes la si vous avez des retours et/ou des questions en ce qui concerne le tuto ou Unity