UNITY

Aller plus loin

14 ans et plus

Fiches techniques

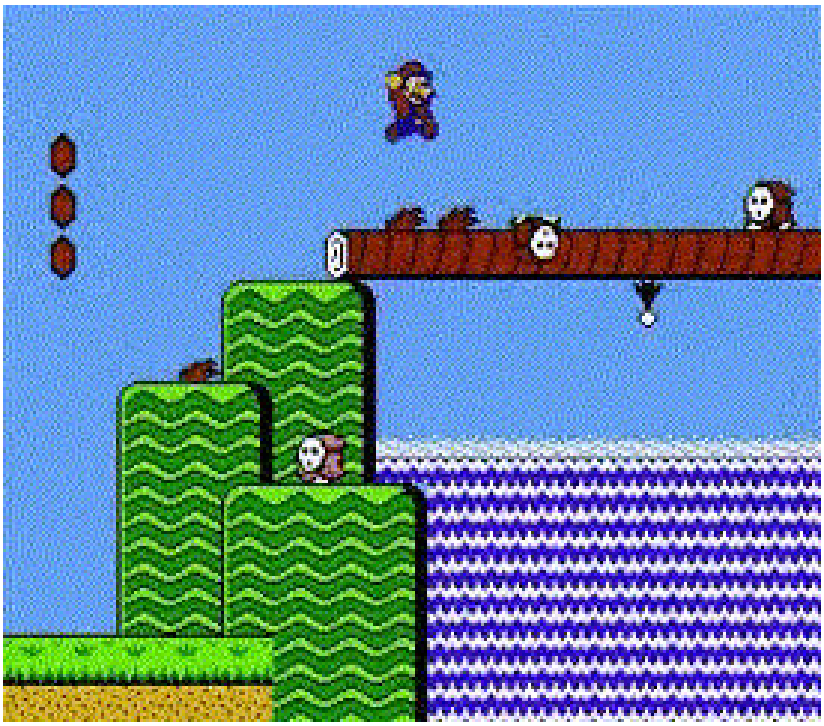
CodeNPlay

Financé par l'Union européenne
NextGenerationEU

DigiLab

2D VS 3D

Les premiers jeux vidéo étaient en 2 dimensions, c'est-à-dire que les éléments qui composent le jeu ne pouvaient se déplacer que sur 2 axes. Ceci était dû aux limitations techniques des ordinateurs et consoles de l'époque.



Par la suite, on a été capable de créer des jeux en trois dimensions avec des éléments qui se déplacent sur trois axes.

Unity est un moteur de jeu 3D qui gère donc les déplacements sur 3 axes (x, y et z). Ceci est le cas même quand vous créez un projet d'un jeu 2D. Tous vos objets auront leurs positions dans l'espace représentées par 3 valeurs x, y et z.

Les seules différences sont que le projet sera préconfiguré pour rendre plus simple certaines choses spécifiques à la 2D ou la 3D.

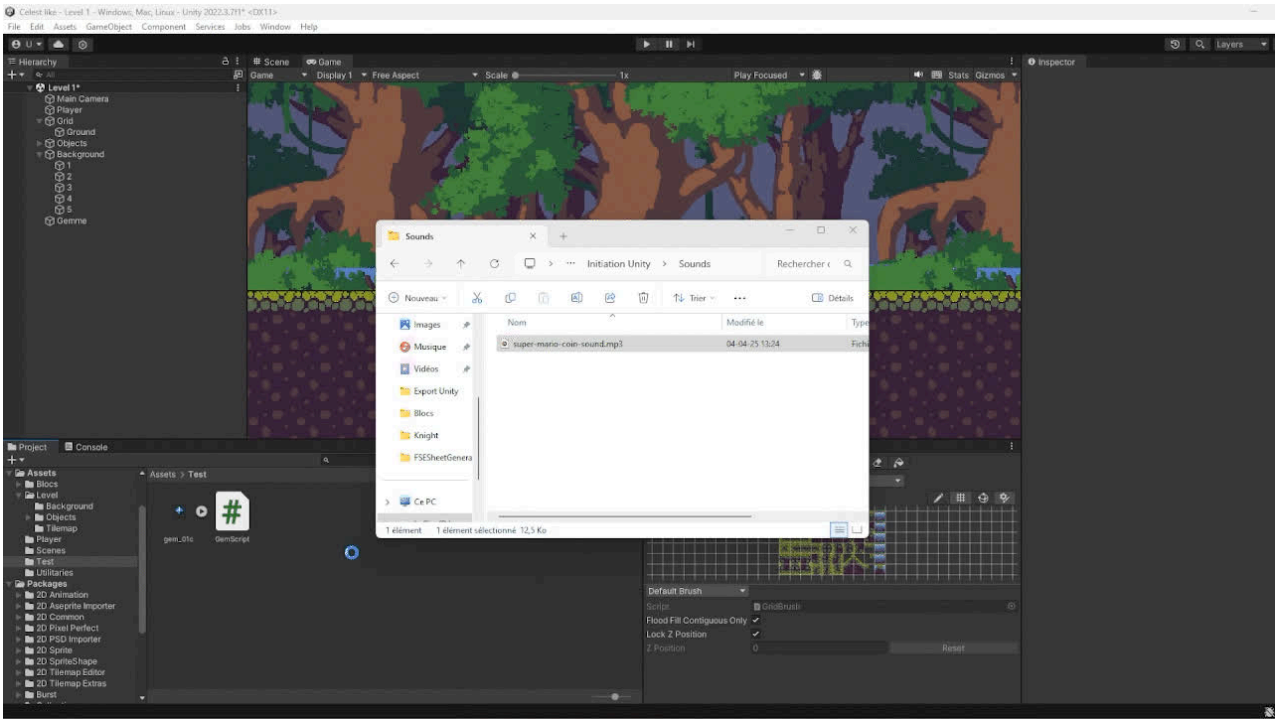
AUDIOSOURCE

A peu de chose près, tous les jeux vidéo comportent des éléments sonores.

Que ça soit des musiques qui viennent rythmer les aventures des joueur·euse·s, des bruitages qui se déclenchent en fonction des événements dans le jeu ou autre. Les sons contribuent énormément à l'expérience de jeu.

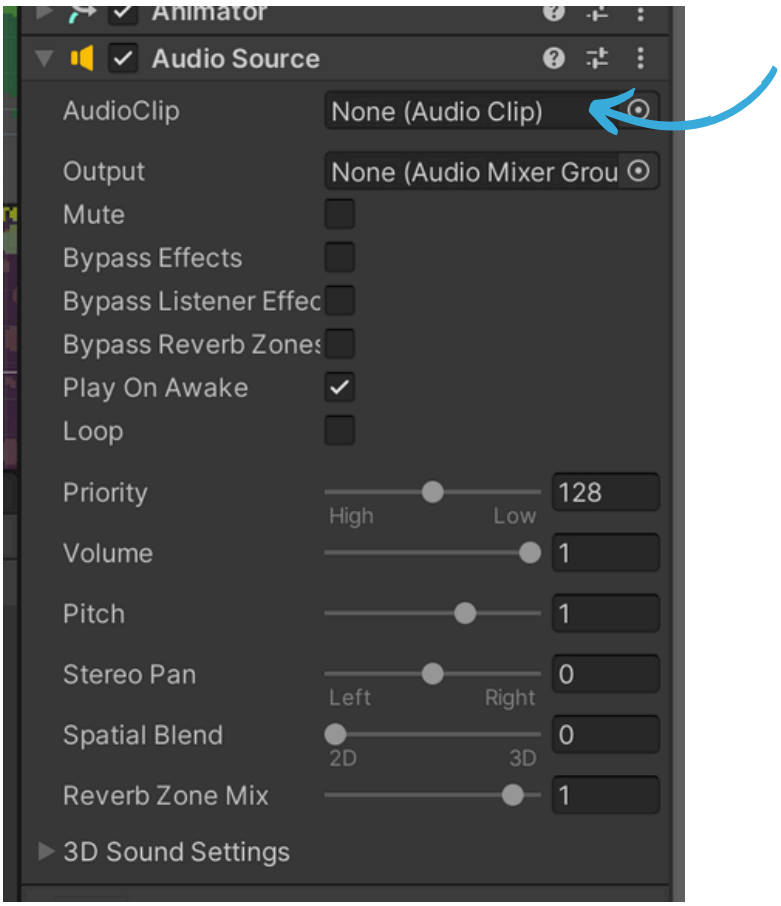
Nous allons donc voir ici comment incorporer des sons dans un jeu. Nous allons d'abord voir comment intégrer une musique de fond qui se déclenche au début du jeu. Nous verrons ensuite comment intégrer des bruitages qui se déclenchent

Commencez par importer le fichier mp3 de l'audio que vous voulez utiliser (en glissant-déposant le fichier dans l'explorateur Unity). Vous obtiendrez un Audio Clip.



Audioclip

Pour jouer un son il faut qu'un gameobject dispose du composant "AudioSource". Ce dernier va pouvoir jouer un audioclip que l'on aura sélectionné. Une fois l'Audiosource ajouté, glissez l'Audioclip dans la propriété correspondante.



Vous pouvez voir qu'il y a une série d'autres propriétés à l'AudioSource. Nous n'allons pas toutes les passer en revue mais nous allons nous attarder sur quelques unes.

- Play on Awake : cette dernière détermine si le son est joué dés le début du jeu (pour une musique il serait logique qu'elle soit cochée).
- Loop : elle défini si le son doit être rejoué quand il est fini.
- Volume : c'est le volume sonore avec lequel le son sera joué

Une fois votre AudioSource correctement paramétrée il ne vous reste plus qu'à lancer le jeu pour entendre votre son.

Si nous voulons maintenant avoir un son qui est joué à un moment donné, sous certaines conditions, il va falloir déclencher le son via un script.

Partez donc d'un GameObject comprenant une AudioSource avec l'AudioClip à jouer. Dans un premier temps décochez "Play On Awake" pour éviter que le son se joue au début du jeu.

Ensuite attachez un Script à ce GameObject. Dans ce Script créez une variable de type AdioSource et assignez l'AudioSource du GameObject à cette variable.

```
public AudioSource audioSource;
```

Maintenant pour jouer le son il vous suffira d'utiliser la méthode Play() sur la variable contenant l'AudioSource.

```
Message Unity | 0 références
private void OnTriggerEnter2D(Collider2D collision)
{
    audioSource.Play();
}
```

Ici le son est joué quand la fonction OnTriggerEnter2D() est appelée

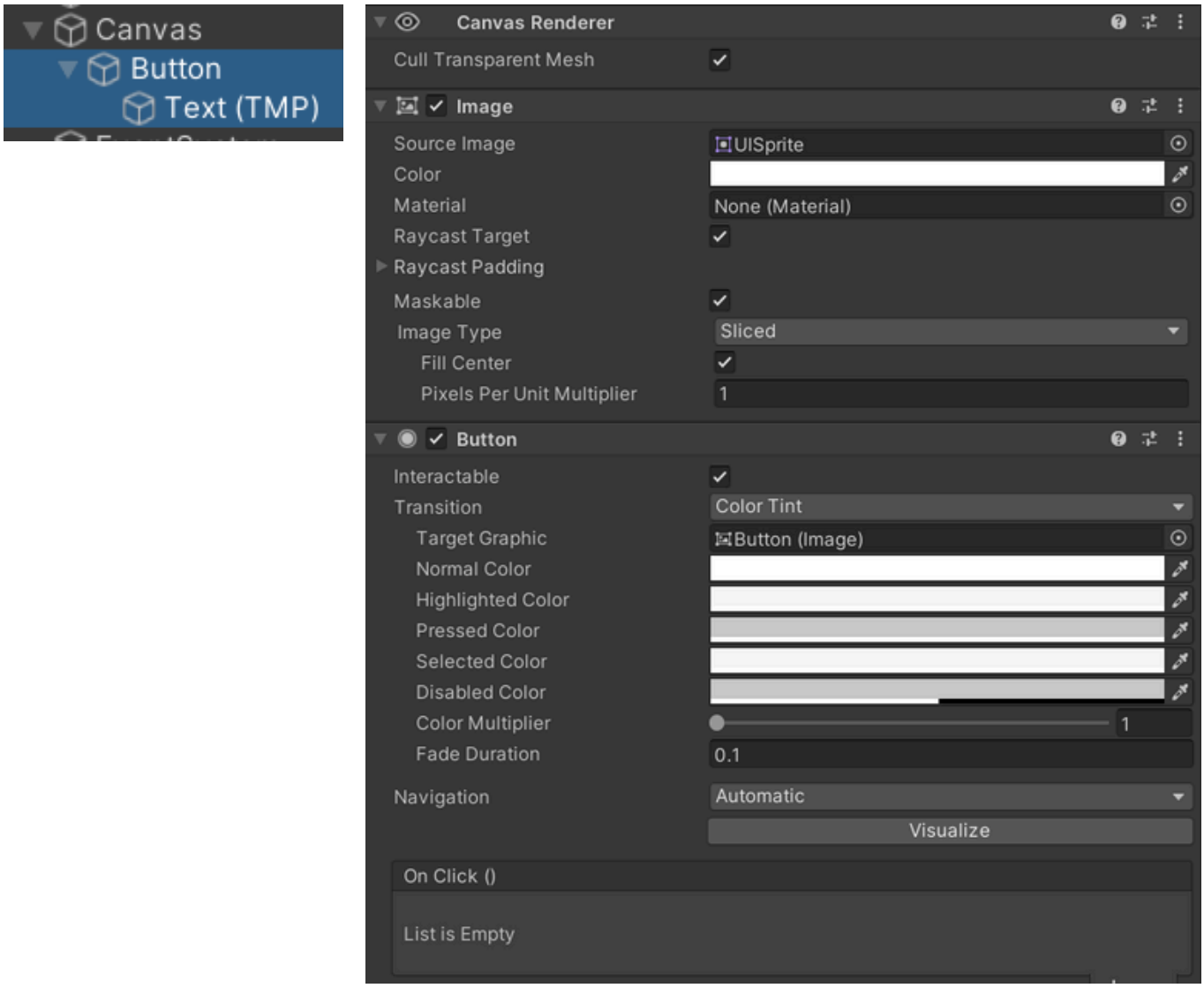
BOUTON

Quand vous créez une interface utilisateur vous pourrez avoir envie de rajouter de l'interactivité. Que le·la joueur·euse puisse interagir avec cette interface. L'un des éléments de base pour permettre cette interaction est le bouton.

En UI un bouton est un élément graphique sur lequel on va pouvoir cliquer et qui va déclencher une action. Pour créer un bouton dans Unity faites clic droit puis "UI" et "Button - TextMeshPro".

Vous aurez créé dans le Canvas un GameObject qui contient les composants suivant :

- Canvas Renderer : c'est un composant qui permet d'afficher des choses dans le canvas.
- Image : c'est un composant similaire au SpriteRenderer, il permet d'afficher une image qui sera le bouton dans l'interface.
- Button : c'est le composant qui permet de gérer l'interactivité du bouton.



Intéressons-nous tout d'abord à l'image, vous voyez qu'elle est similaire à un SpriteRenderer. Source Image est l'équivalent du Sprite et Color est identique à celui du SpriteRenderer.

Vous pouvez voir qu'un autre GameObject a été créé, il s'agit d'un TextMeshPro, il permet d'afficher du texte dans le bouton (voir section TextMEshPro). Vous n'êtes pas obligé de le gardé, si vous voulez l'enlever, sélectionnez-le et appuyez sur la touche Delete.

Intéressons-nous maintenant à l'image, vous voyez qu'elle est similaire à un SpriteRenderer. Source Image est l'équivalent du Sprite et Color est identique à celui du SpriteRenderer.

Ensuite intéressons-nous au composant Button et plus précisément au bloc On Click (). C'est ici que nous allons déterminer ce qu'il va se passer quand le bouton sera cliqué.

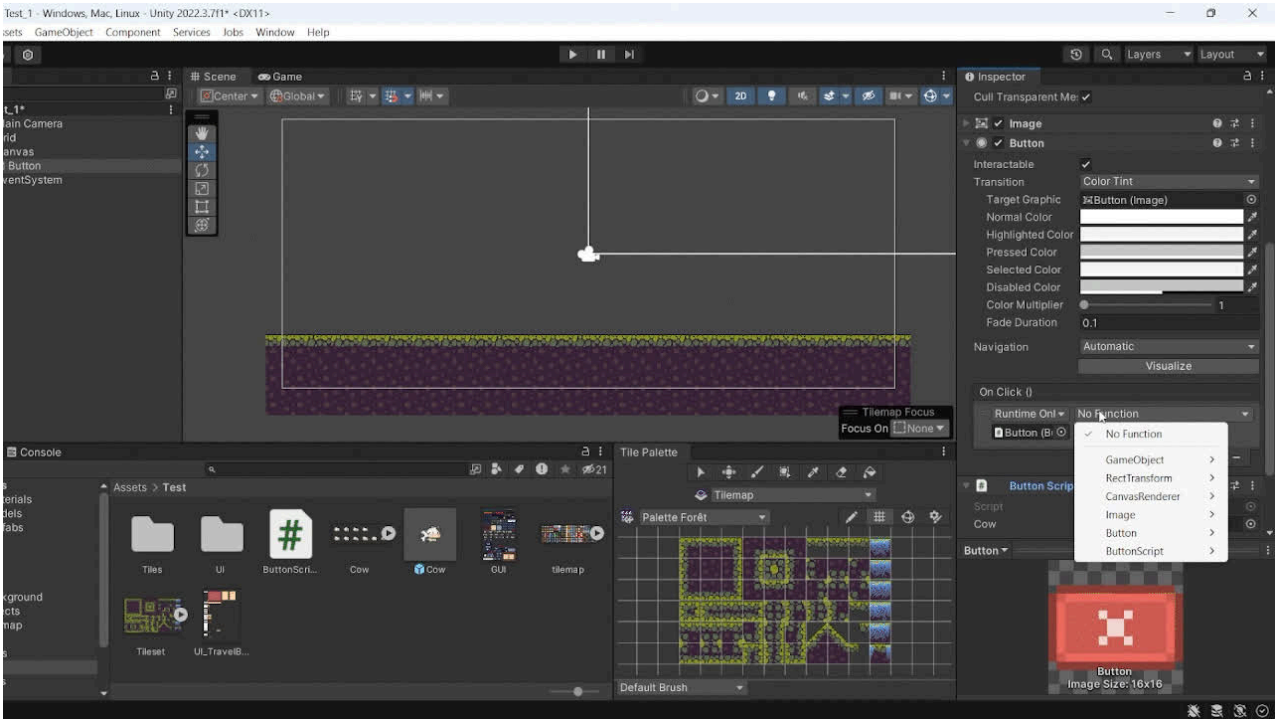
Il faut bien comprendre comment fonctionne le OnClick, ce dernier va prendre une fonction qu'on va lui donner et il va exécuter cette fonction quand le bouton sera cliqué. Cette fonction devra venir d'un Script présent dans le jeu.

Dans mon exemple je vais utiliser la fonction SpawnCow d'un Script nommé ButtonScript. Le Script ButtonScript est attaché au GameObject du bouton. Cette fonction fait simplement apparaitre un GameObject à l'apparence d'une vache. Attention, elle doit bien être Public.

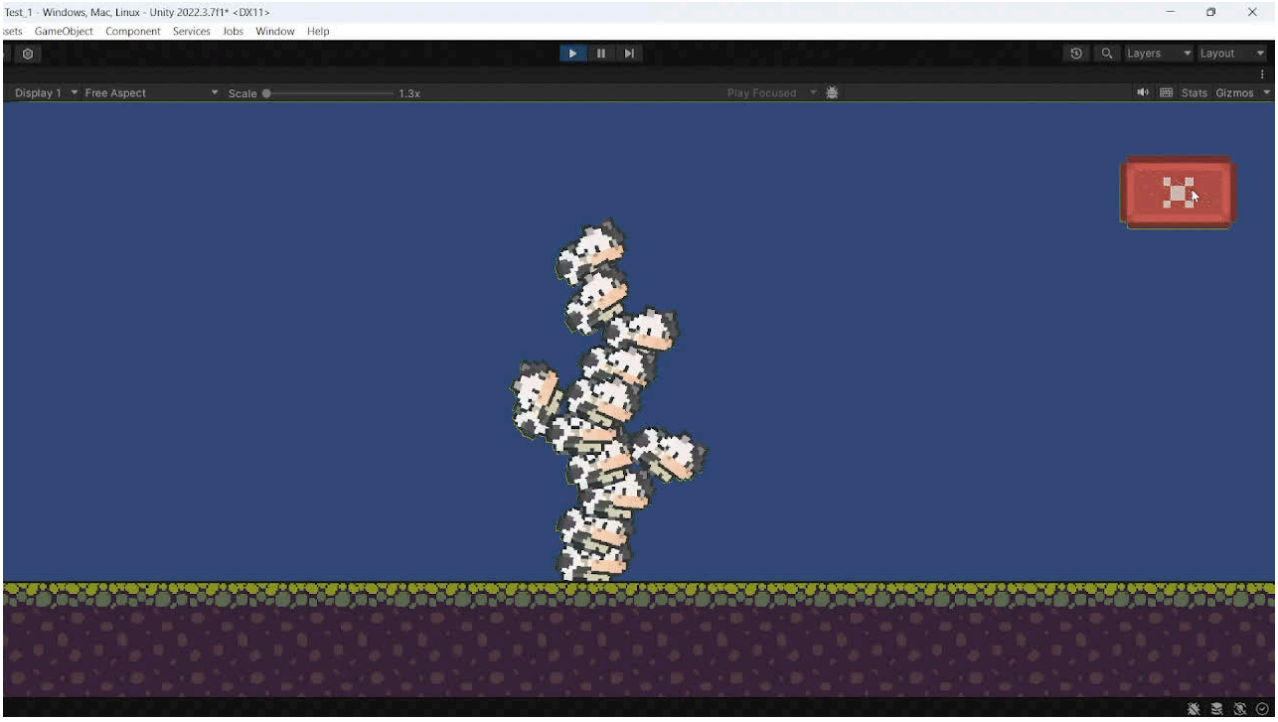
```
public class ButtonScript : MonoBehaviour
{
    public GameObject cow;

    0 références
    public void SpawnCow()
    {
        Instantiate(cow);
    }
}
```

Pour que le bouton exécute la fonction il nous suffit d'ajouter un élément dans le bloc OnClick, d'attribuer le GameObject ou le composant contenant la fonction puis de sélectionner la fonction à exécuter.

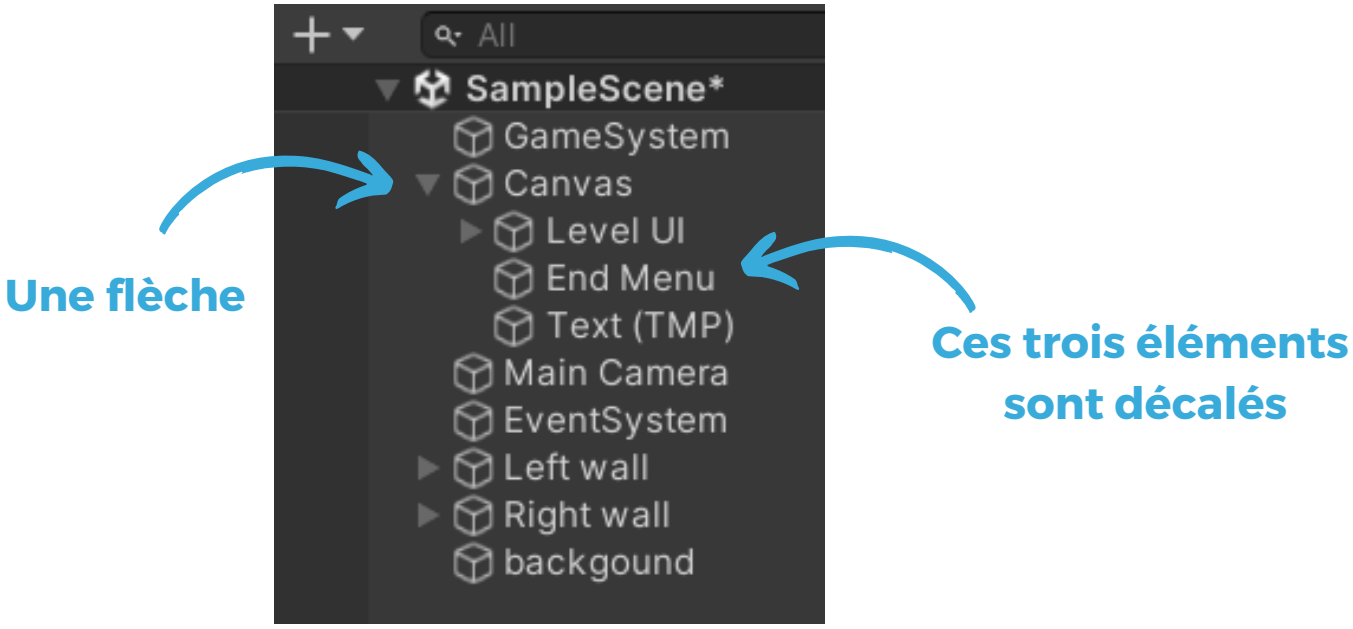


Si je lance maintenant le jeu et que je cliques sur le bouton je devrais voir apparaitre un GameObject de vache à chaque fois que ej clique.

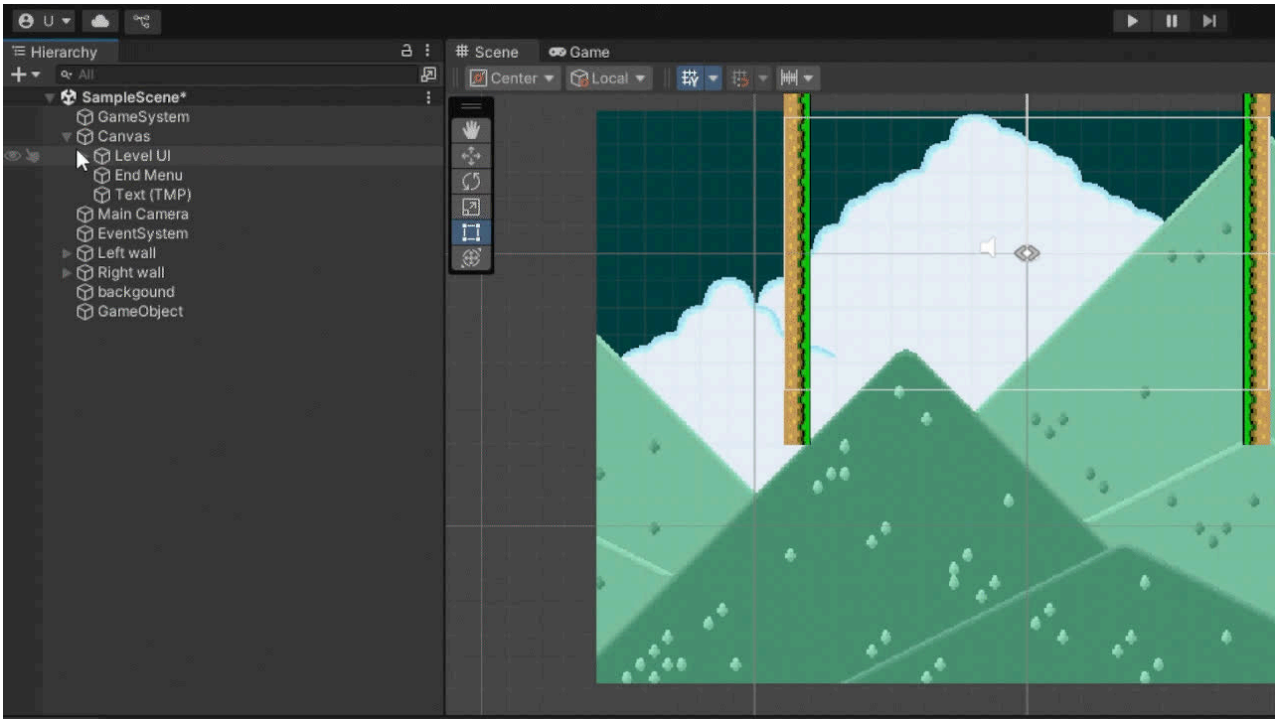


CHILDREN - PARENTS

Vous aurez peut-être déjà remarqué que tous les éléments dans la hiérarchie ne se trouvent pas à la même distance de la limite gauche de la fenêtre. Ou vous aurez pu remarquer qu'il a parfois une flèche à gauche d'un GameObject.

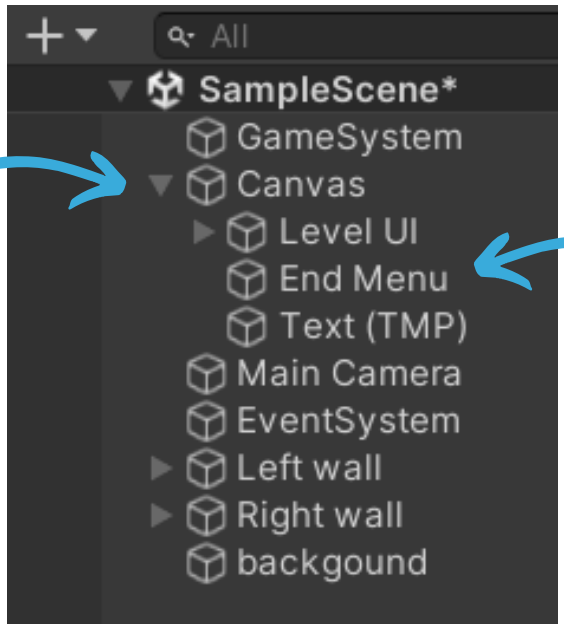


Vous pouvez cliquer sur la flèche pour faire apparaitre/masquer les éléments décalés.



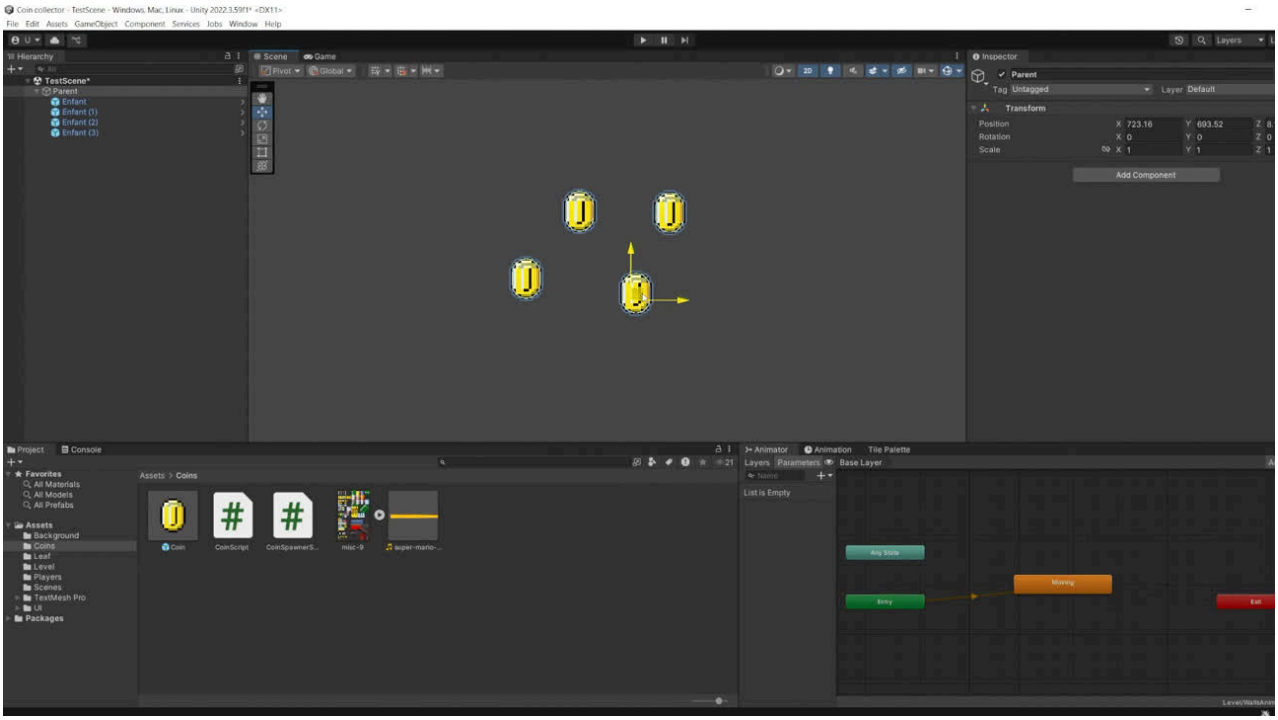
Les éléments décalés à droite ont une relation spéciale avec l'élément juste au dessus. On dit que ceux en dessous sont les Children (enfants) de celui au-dessus qui est leur Parent.

Le Canvas est le Parent des trois GameObjects en dessous



Les trois sont des enfants du parent

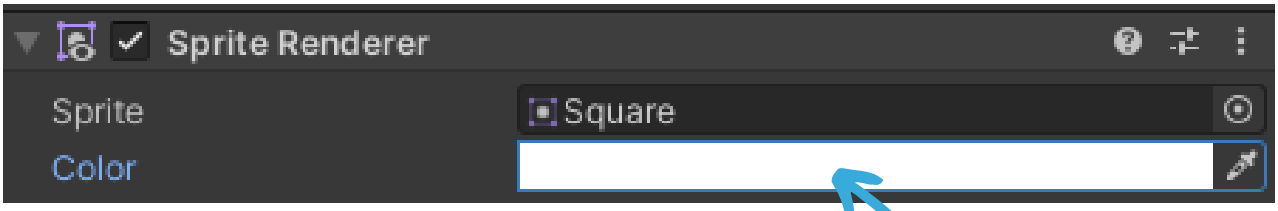
Avoir des GameObjects qui sont parents/enfants les uns des autres permet de nombreuses choses. Par exemple, si vous avez un GameObject enfant d'un autre et que vous bougez le parent, l'enfant bougera avec.



Si vous déplacer un parent et que vous regardez le transform.position d'un enfant vous verrez que celui-ci reste inchangé.

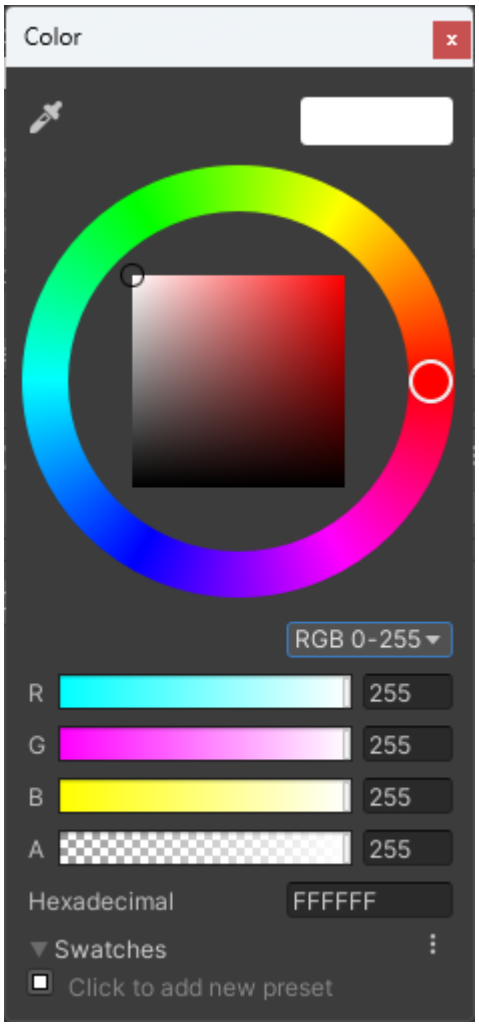
COLOR

Vous retrouverez dans de nombreux Composants des propriétés “Color” (couleur) ou qui portent un nom différent mais consistent en une case de la forme suivante :

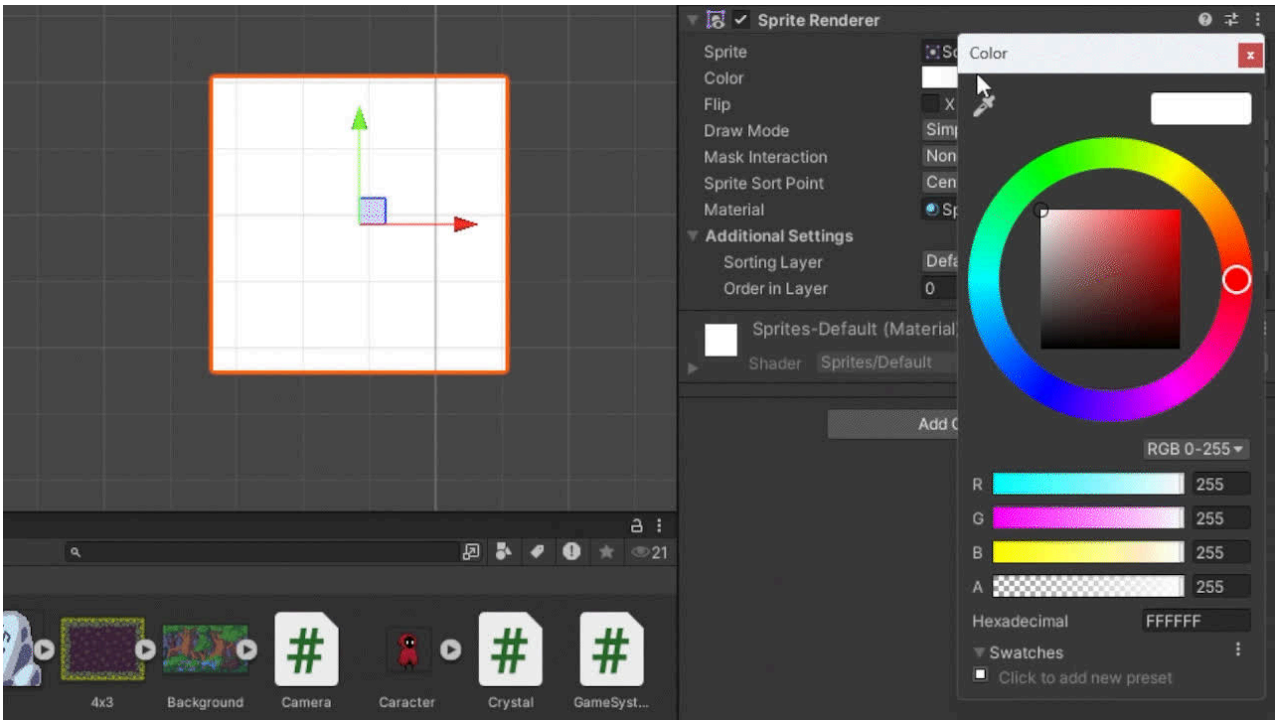


la case de sélection de couleur

Quand vous cliquez sur ces case une petite fenêtre s’ouvre vous permettant de sélectionner une couleur.

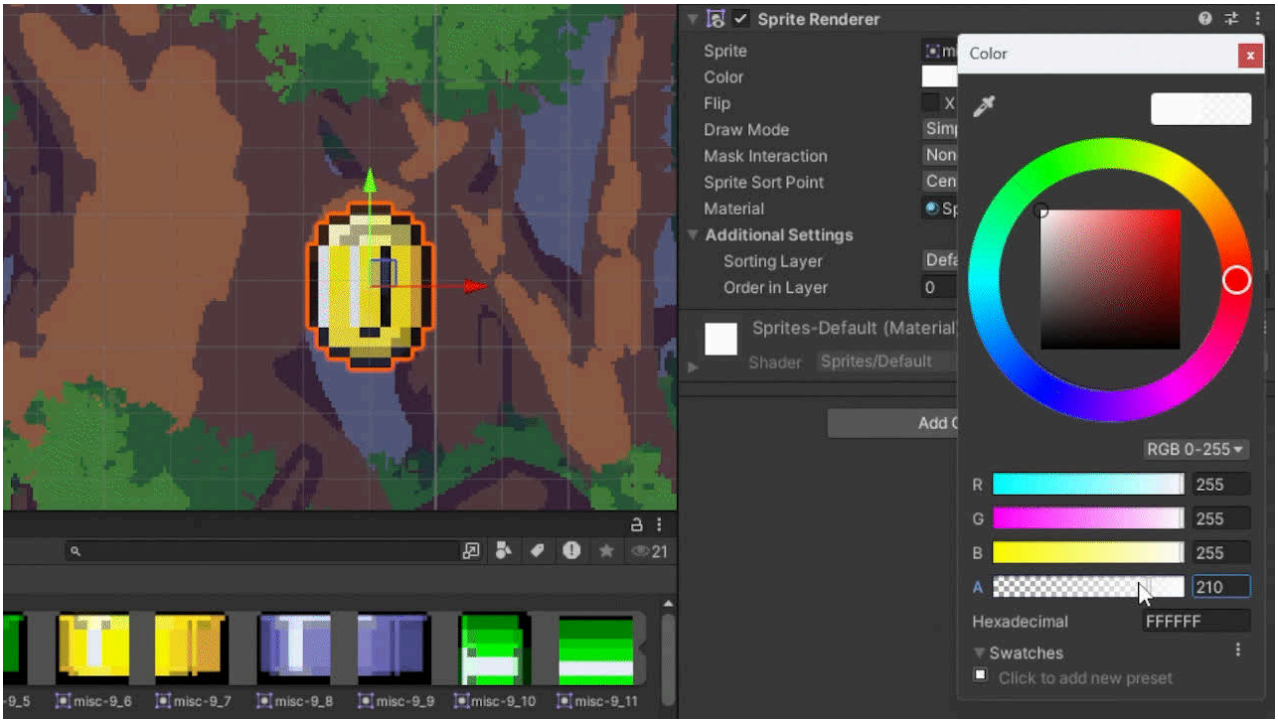


La roue autour du carré permet de choisir une couleur principale puis le carré permet de faire varier l'intensité et l'obscurité.

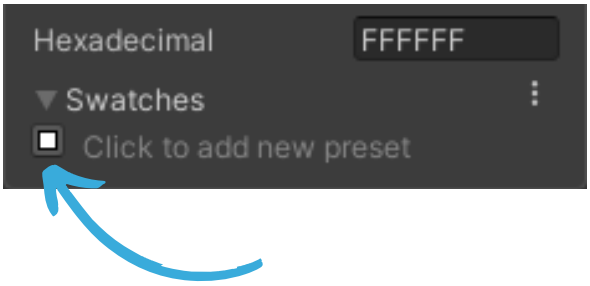


Juste en dessous du rond vous pouvez voir 4 curseurs, les trois premiers correspondent aux valeurs RGB (Red Green Blue : rouge vert bleu). Pour plus d'information sur les couleurs RGB visitez cette page.

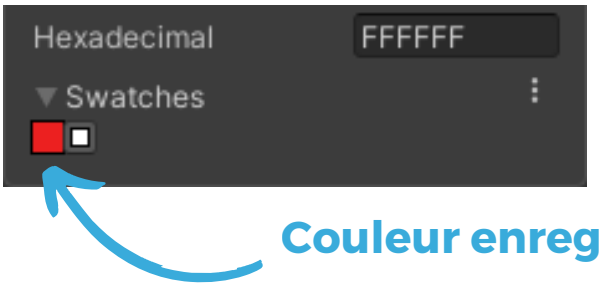
Le 4e correspond à ce que l'on appel le canal alpha, c'est une valeur qui va définir l'opacité de la couleur. Une couleur avec un alpha au maximum sera parfaitement opaque. Une couleur avec un alpha au minimum sera parfaitement transparente.



Vous pourriez vouloir enregistrer une couleur que vous avez sélectionner. Pour ce faire cliquez sur le petit carré en bas à gauche de la fenètre.



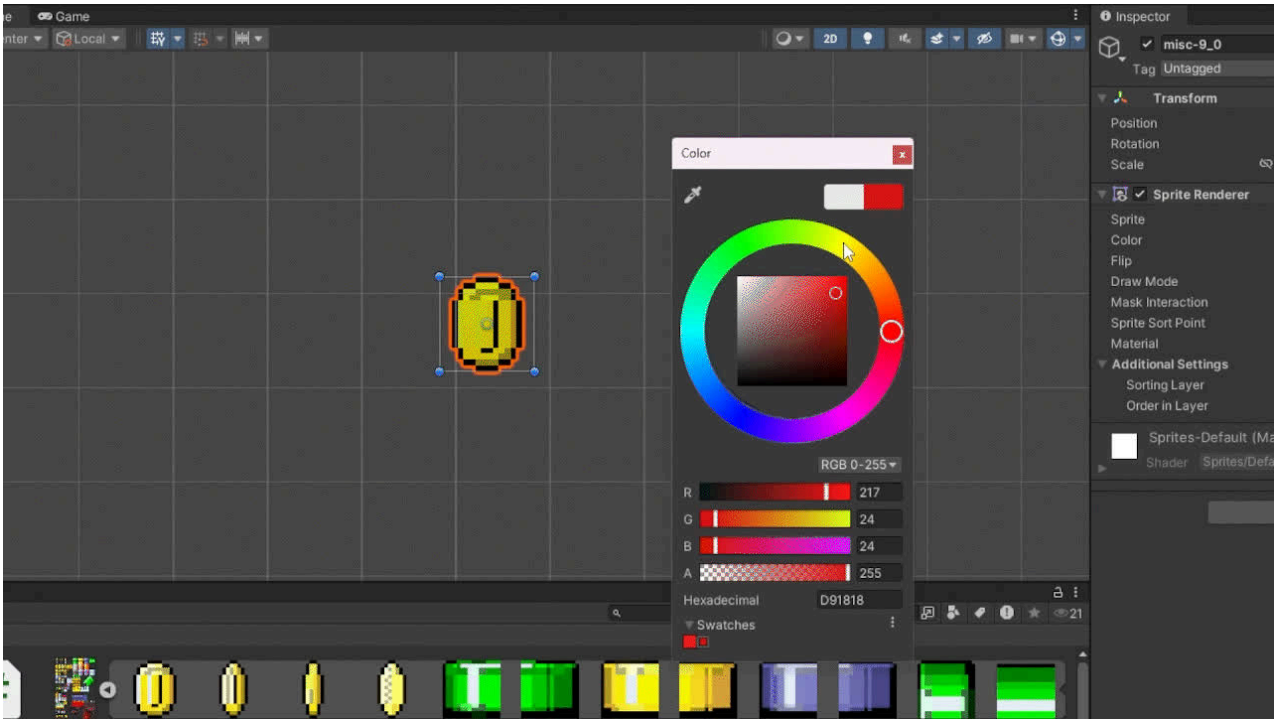
Une fois cela fait vous pourrez retrouver la couleur dans la fenêtre de sélection.



Couleur enregistrée

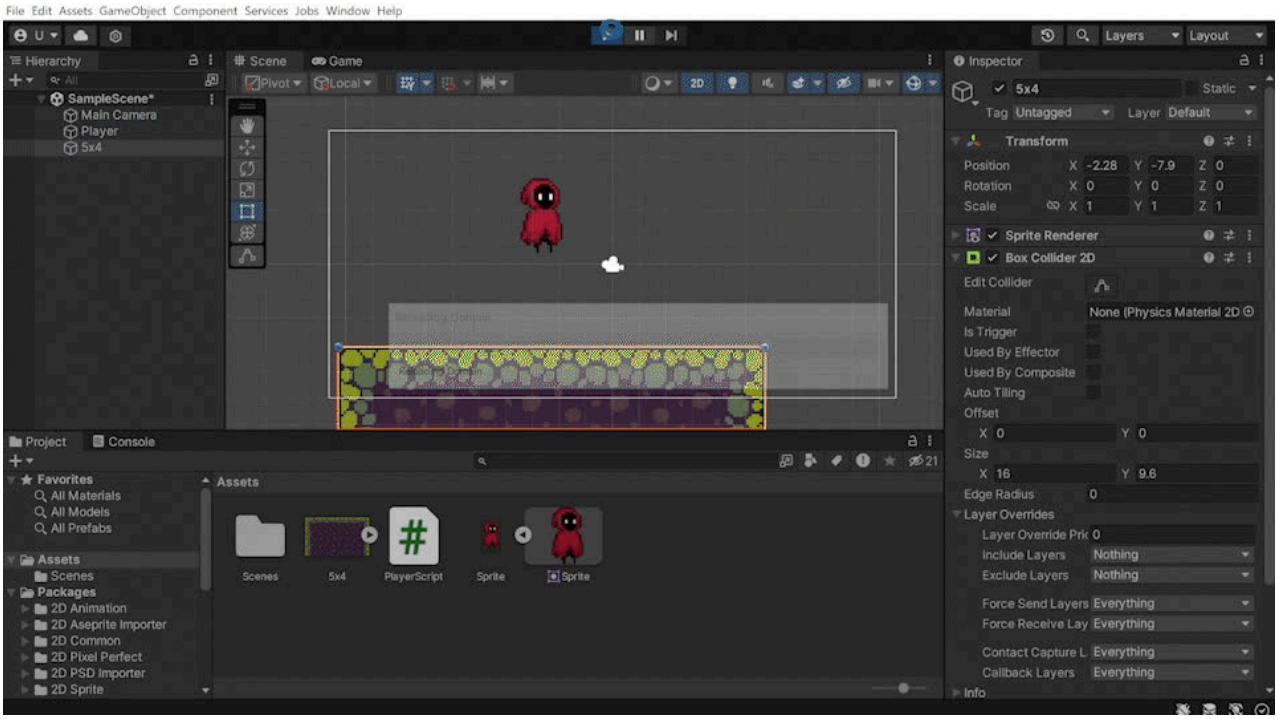
Attention : pour bon nombre de ces composants les propriétés qui prennent une couleur en paramètre ne servent pas à définir la couleur. Elles servent à modifier la couleur d'une image.

Dans l'exemple ci-dessous la propriété "Color" du Sprite Renderer change les couleurs du Sprite. Elle ne remplace pas toutes les couleurs du Sprite par la couleur sélectionnée.



COLLIDER 2D

Les Colliders sont des composants qui vont donner un volume (ou plutôt exactement une surface pour les jeux 2D) aux GameObjects. Ce volume va pouvoir entrer en collision avec les volumes d'autres Colliders.

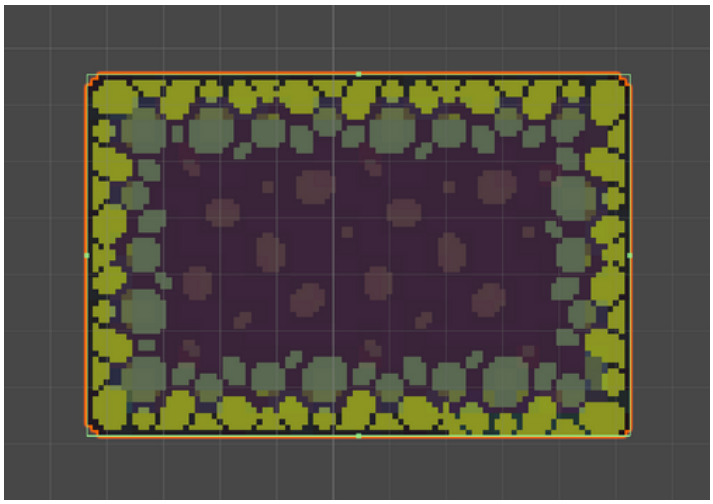


Pour qu'un GameObject entre en collision avec un autre il faut tout d'abord que chacun des deux ait un Collider et ensuite que les deux Colliders se touchent.

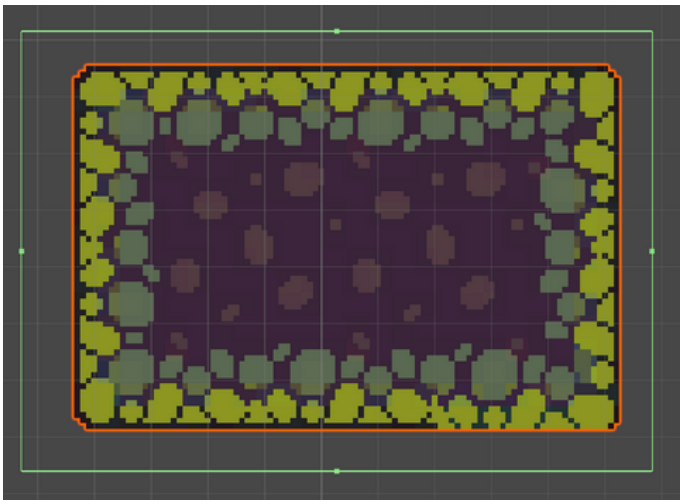
Un Collider peut déclencher un bout de programme quand il entre en collision avec un autre Collider. Pour cela voir "OnCollisionEnter et OnTriggerEnter".

Un Collider a un paramètre nommé "Is Trigger". Quand ce paramètre est coché le Collider ne rentrera pas en collision avec les autres. Mais il permettra de détecter quand un Collider rentre dans sa zone et déclenchera OnTriggerEnter.

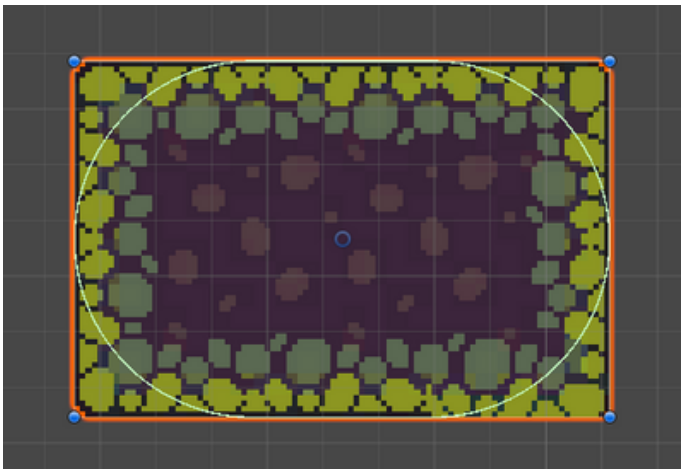
Les Colliders peuvent être de même taille et forme que le visuel des GameObjects auxquels ils sont attachés ou peut être d'une taille et forme complètement différente.



Le BoxCollider est très proche des contours du Sprite

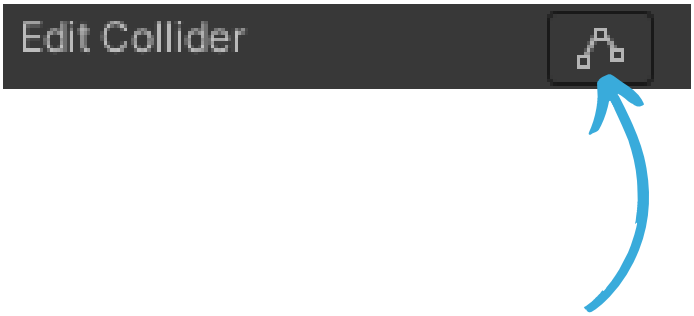


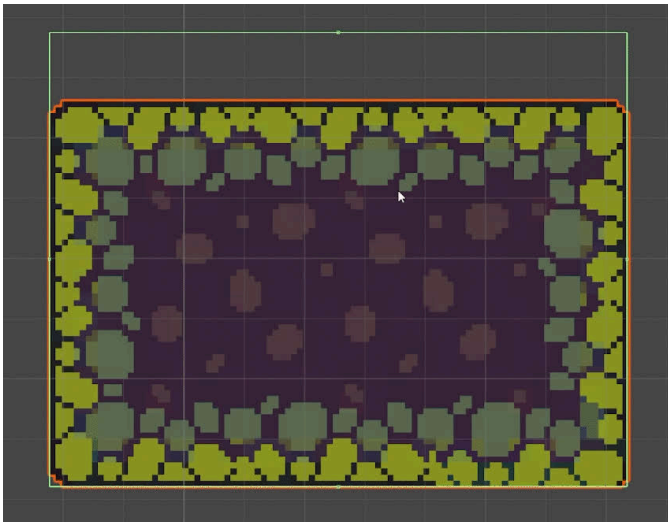
Le BoxCollider dépasse les contours du Sprite



Le CapsuleCollider n'est pas de la même forme que le Sprite

Pour changer les dimensions d'un Collider, cliquez sur la case à côté de "Edit Collider" en haut du composant dans l'inspecteur. Cliquez ensuite sur les petits carrés autour du Collider et glissez-les.





DESTROY

Il est souvent utile d'être capable de retirer des éléments du jeu pendant que celui-ci s'exécute et sous certaines conditions. Si nous avons, par exemple, un objet que le personnage joueur peut récupérer. Dans ce cas, il serait utile que l'objet disparaisse quand le personnage le récupère.

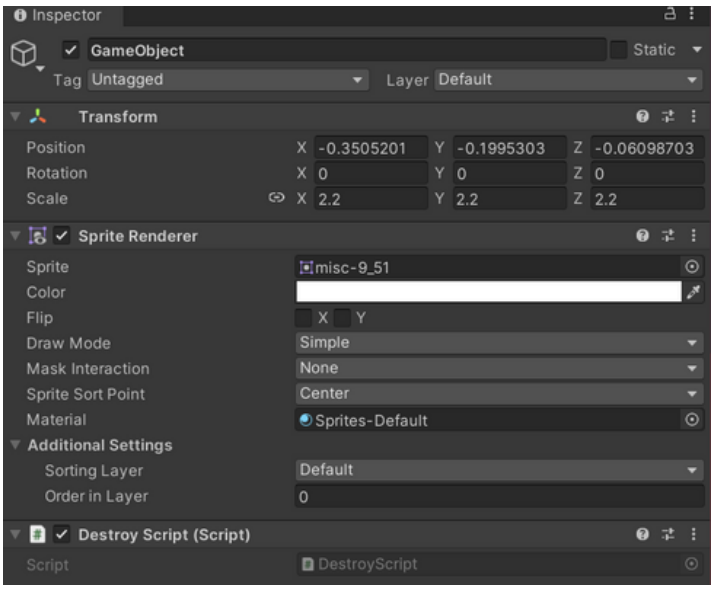
Il existe pour cela une fonction nommée Destroy(). Elle prend en paramètre le GameObject à faire disparaitre.

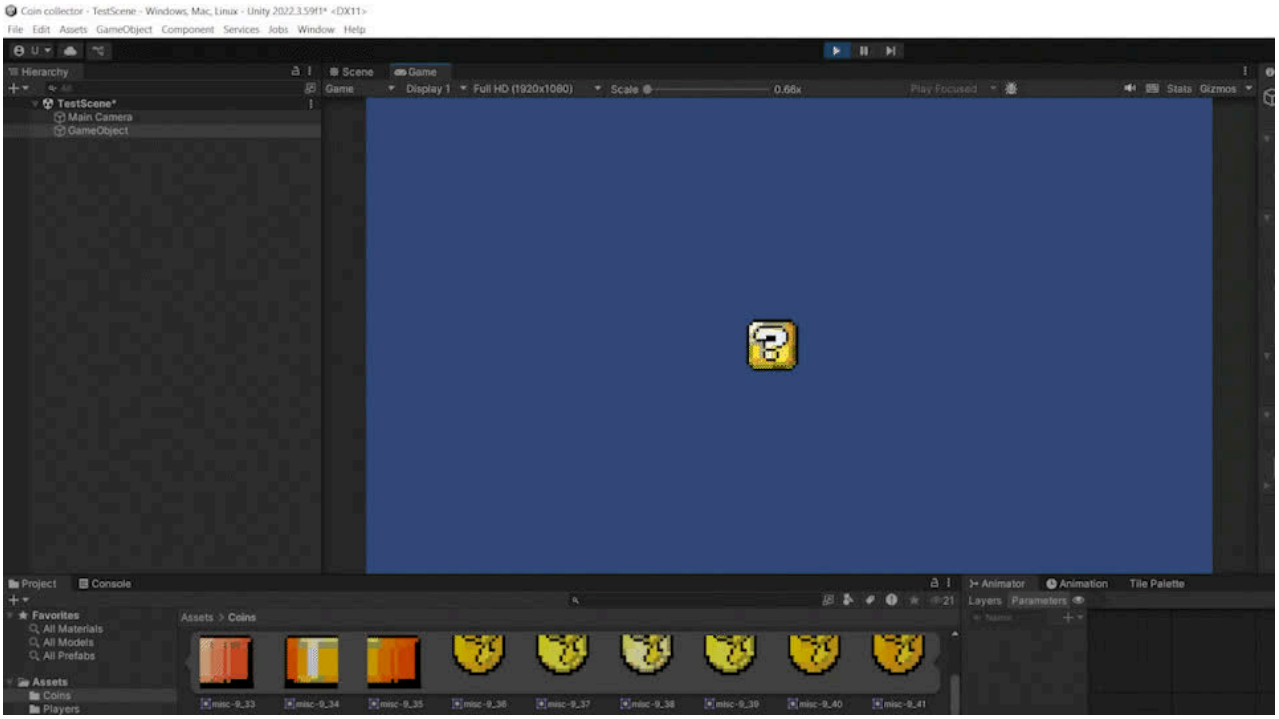
```
Destroy(gameObject);
```

Dans un script, le mot-clé gameObject correspond au GameObject auquel est attaché le script. La commande Destroy(gameObject) détruira donc le GameObject auquel est attaché le script.

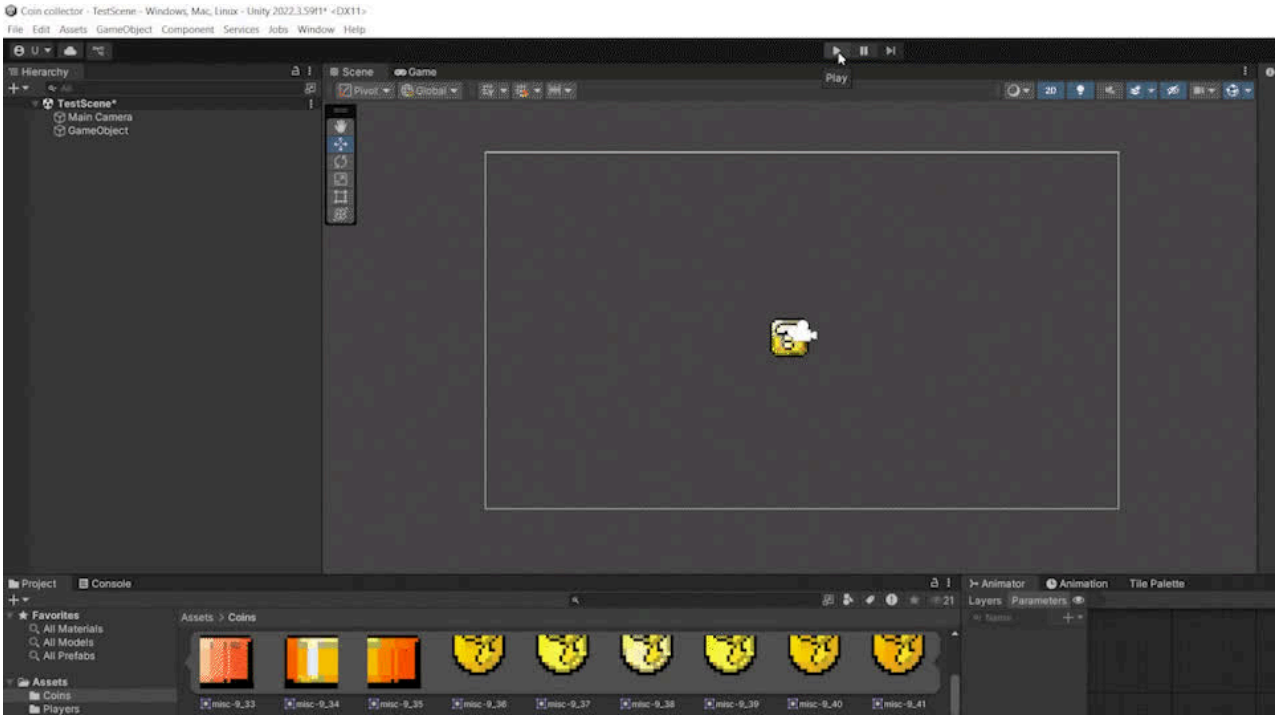
Dans l'exemple qui suit j'ai ajouté le script suivant à un GameObject avec un SpriteRenderer.

```
// Update is called once per frame
Message Unity | 0 références
void Update()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        Destroy(gameObject);
    }
}
```





Attention, le GameObject ne disparaitra pas de manière définitive. Si vous relancez le jeu il aura réapparu. Il ne disparait que dans l'exécution en cours du jeu. Si vous stoppez le jeu il réapparaitra.



GETCONPONENT

Quand vous utilisez un Composant dans un Script vous pouvez, comme vu précédemment, créer une variable du bon type et assigner le composant dans l'éditeur.

Il est cependant possible de faire l'assignation sans passer par l'éditeur.

Il suffit d'utiliser la fonction GetComponent(), cette dernière renvoie le premier Composant du type spécifié et attaché au même GameObject.

Pour l'utiliser, déclarez la variable du composant en début de programme puis appelez GetComponent et assignez son résultat à la variable.
Vous devez préciser type du composant que vous voulez récupérer entre les <> avant les parenthèses.

```
public class TestScript : MonoBehaviour
{
    public BoxCollider2D boxCollider;

    // Start is called before the first frame update
    Message Unity | 0 références
    void Start()
    {
        boxCollider = GetComponent<BoxCollider2D>();
    }
```

Type du composant

Faites attention au fait que la fonction renvoie le premier Composant qui correspond. Si le GameObject comprend plusieurs Composants du type donné vous n'en récupérerez qu'un seul.

INPUT.GETKEY

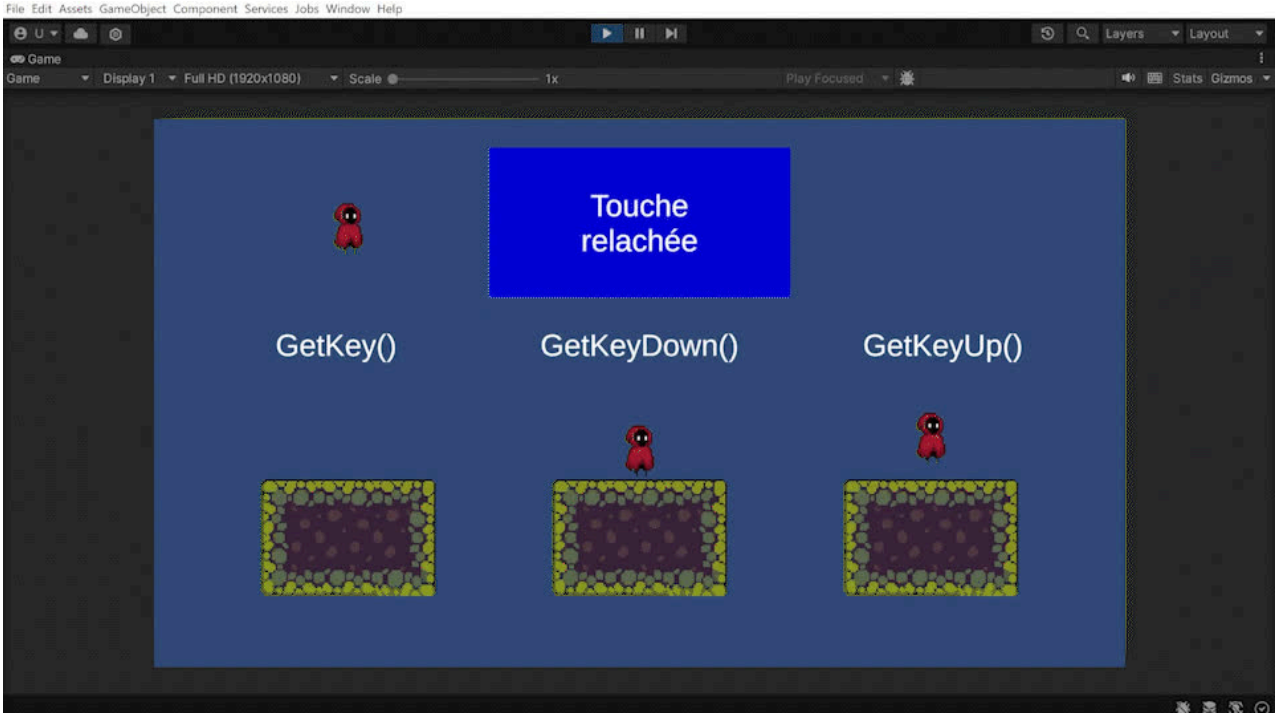
La librairie Input contient de nombreuses fonctions qui permettent de récupérer les touches pressées par l'utilisateur et autres entrées qu'ils fera avec des périphériques. Il est nécessaire de préciser la touche à inspecter via son KeyCode, un code d'identification propre à Unity. Il vous suffit pour cela de taper **Keycode.** suivit du nom de la touche.

Parmi ces fonction on en retrouve trois qui nous seront très utiles : **Input.GetKey()**, **Input.GetKeyDown()** et **Input.GetKeyUp()**.

- **Input.GetKey()** : Elle renverra True si la touche est pressée à cette frame.
- **Input.GetKeyDown()** : Elle renverra True si la touche a commencé à être pressée à cette frame et uniquement quand elle a commencé à être pressée.
- **Input.GetKeyUp()** : Elle renverra True si la touche a arrêté d'être pressée à cette frame et uniquement quand elle a arrêté d'être pressée.

Avant tout il faut bien comprendre une chose, ces fonctions vont nous renseigner sur l'état d'une touche à la frame exacte à laquelle la fonction est appelée. De ce fait si vous appuyez puis relâchez une fois les fonctions **Input.GetKeyDown()** et **Input.GetKeyDown()** ne renverront vrai qu'une seul fois tandis que **Input.GetKey()** renverra vrai pour chaque frame durant laquelle vous avez pressé la touche.

Dans l'exemple ci-dessous les trois personnages ont un Rigidbody et la velocité de celui-ci sera assignée d'un vecteur vers le haut quand la condition est remplie.



IMPORTER DES POLICES

Quand vous ajoutez des textes à l'intérieur de votre jeu (dialogues, UI, etc) vous aurez certainement envie d'utiliser une autre polices que celle de base.

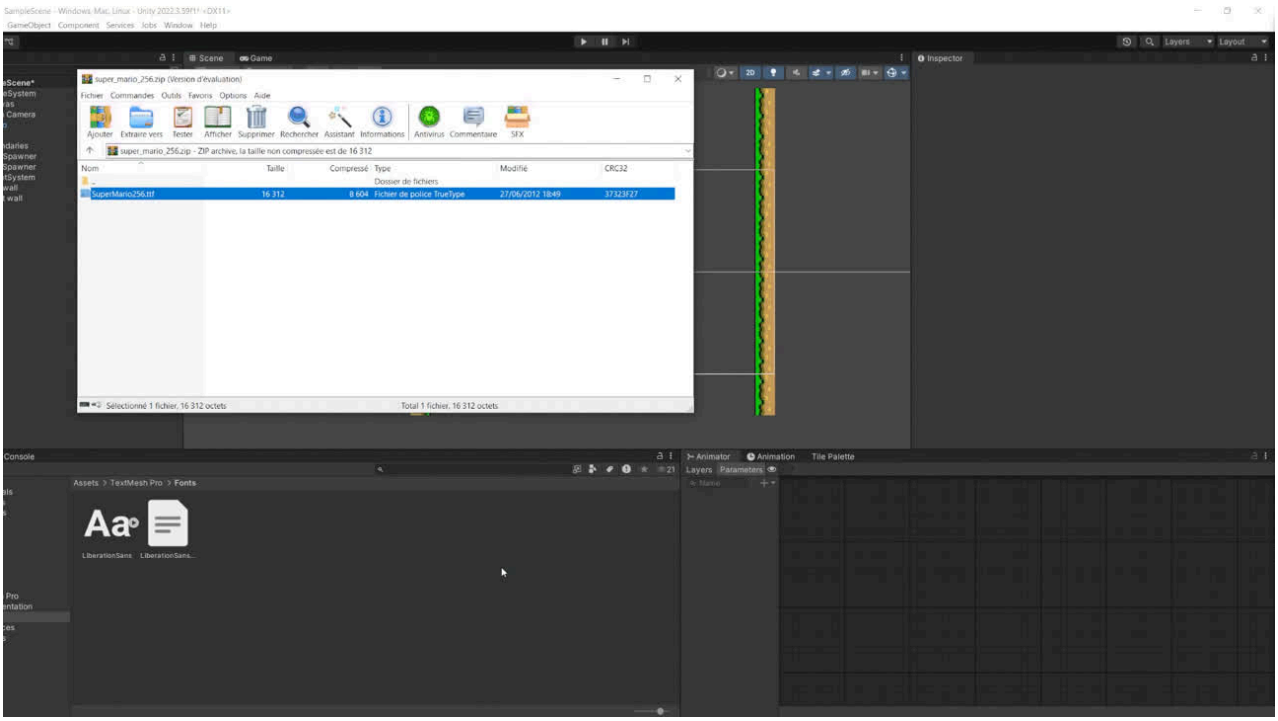
Pour ce faire vous allez devoir importer la police que vous voulez utiliser dans votre jeu. Vous ne saurez pas utiliser directement les polices disponibles sur votre ordinateur.

Pour importer une police commencez par en télécharger une, vous pouvez trouver de nombreuses polices gratuites sur les sites suivants :

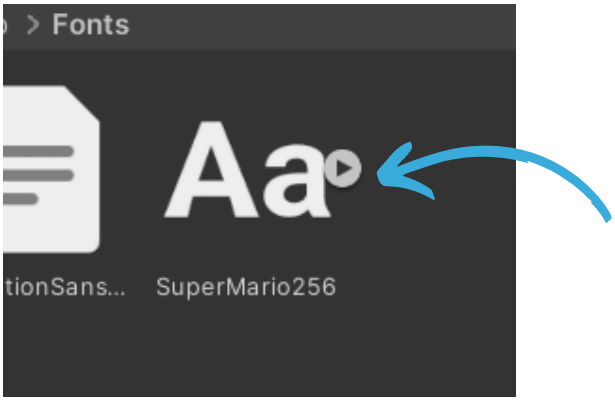
- https://www.dafont.com/fr/
- https://fonts.google.com/

Une fois votre police téléchargé, vous vous retrouverez avec un fichier en .ttf ou .otf. Dans les deux cas la manipulation est la même.
Il n'est pas rare que vous ayez à extraire

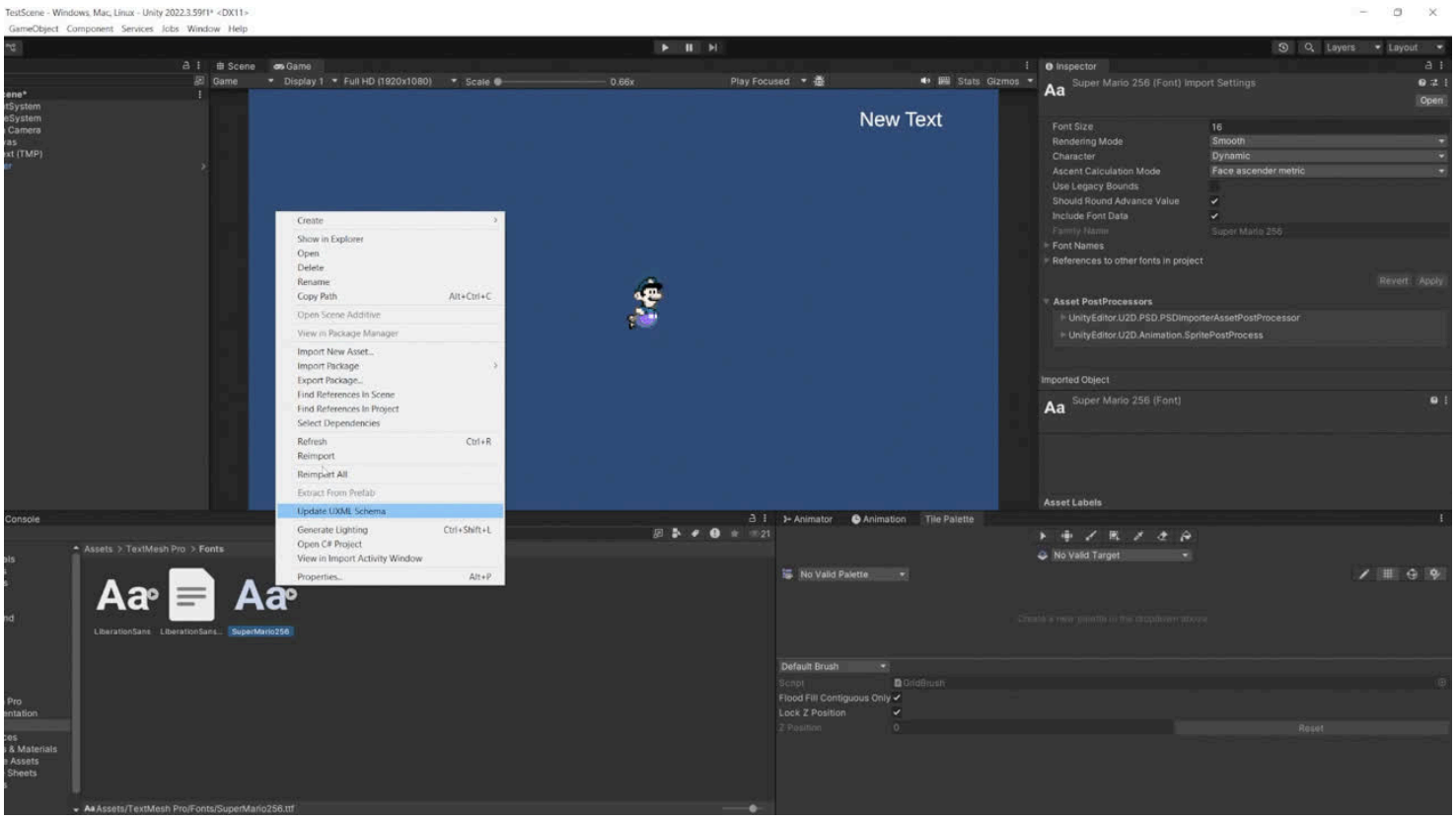
Glissez-déposez le fichier de la police dans l'explorateur Unity.



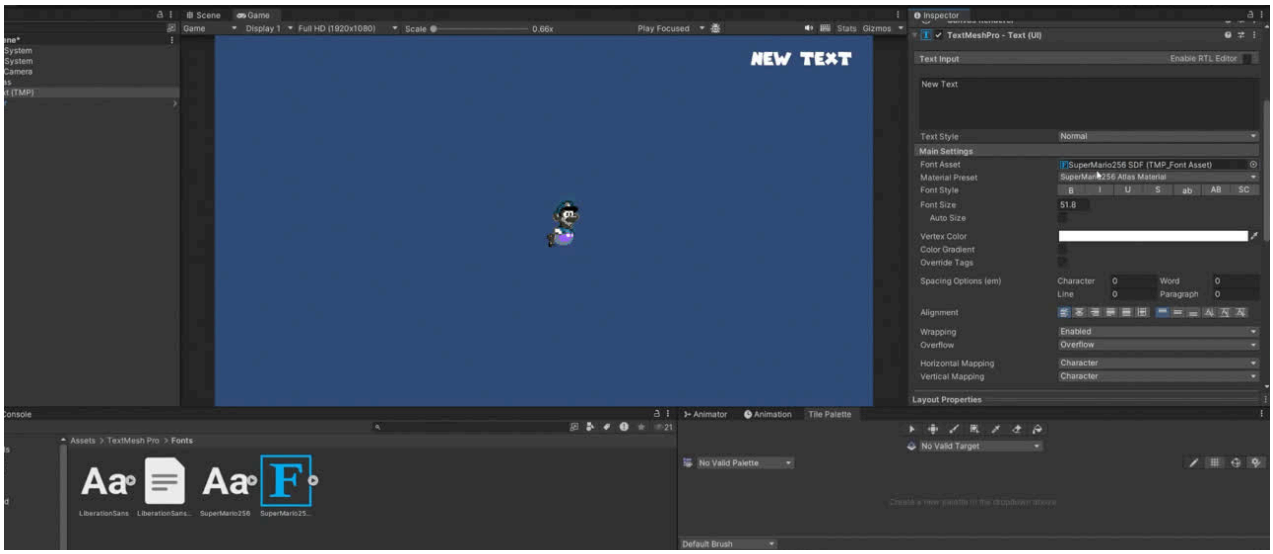
Vous verrez qu'un élément Font sera créé dans l'explorateur de fichier de Unity.



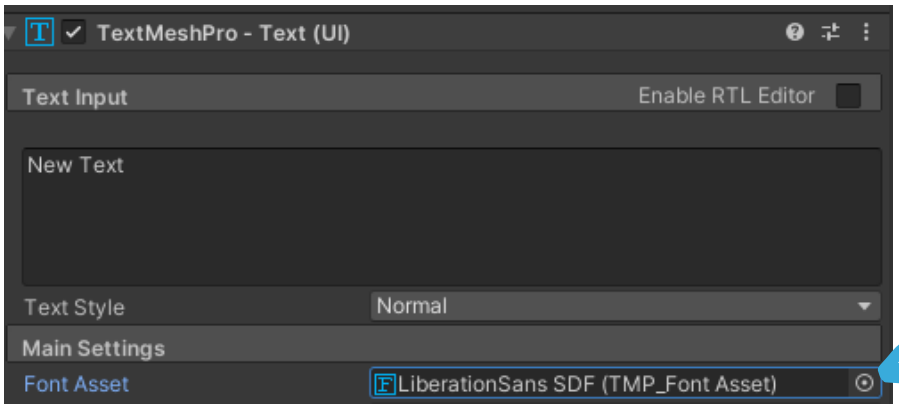
Il vous faut ensuite faire clique droit sur la Font, sélectionner Create puis TextMeshPro puis cliquer sur Font Asset



Il ne vous restera plus qu'à glisser-déposer cette Font dans la case adéquate dans tous les textes qui doivent être écrits avec cette Fonte.



Vous pouvez aussi sélectionner une police parmi celles que vous avez importé en cliquant sur le rond avec un point à droite de la case dans un TextMeshPro.



Cliquer ici

La nouvelle police dans mon exemple

INSTANTIATE

Si vous avez besoin de faire apparaitre un GameObject pendant l'exécution du jeu l'outil dont vous avez besoin est la fonction Instantiate().

Cette dernière existe sous plusieurs formes, celle à laquelle nous allons nous intéresser est Instantiate(Object original, Vector3 position, Quaternion rotation).

Elle peut sembler un peu compliqué comme ça mais elle est assez simple à utiliser. Il faut comprendre qu'elle prend trois paramètres :

- Object original : C'est le prefab que l'on va vouloir faire apparaitre dans la scène.
- Vector3 position : C'est la position à laquelle on veut faire apparaitre l'objet.
- Quaternion rotation : c'est l'orientation de l'objet dans l'espace, la direction vers laquelle il est tourné.

Pour démarrer je vous conseille d'utiliser Instantiate() depuis un GameObject qui aura pour fonction de faire apparaitre le prefab.

Par exemple si vous voulez faire apparaitre des pièces qui se déplacent, il est plus simple de créer un GameObject qui ait pour fonction de faire apparaitre ces pièces.

le gameobject qui fait apparaitre les pièces

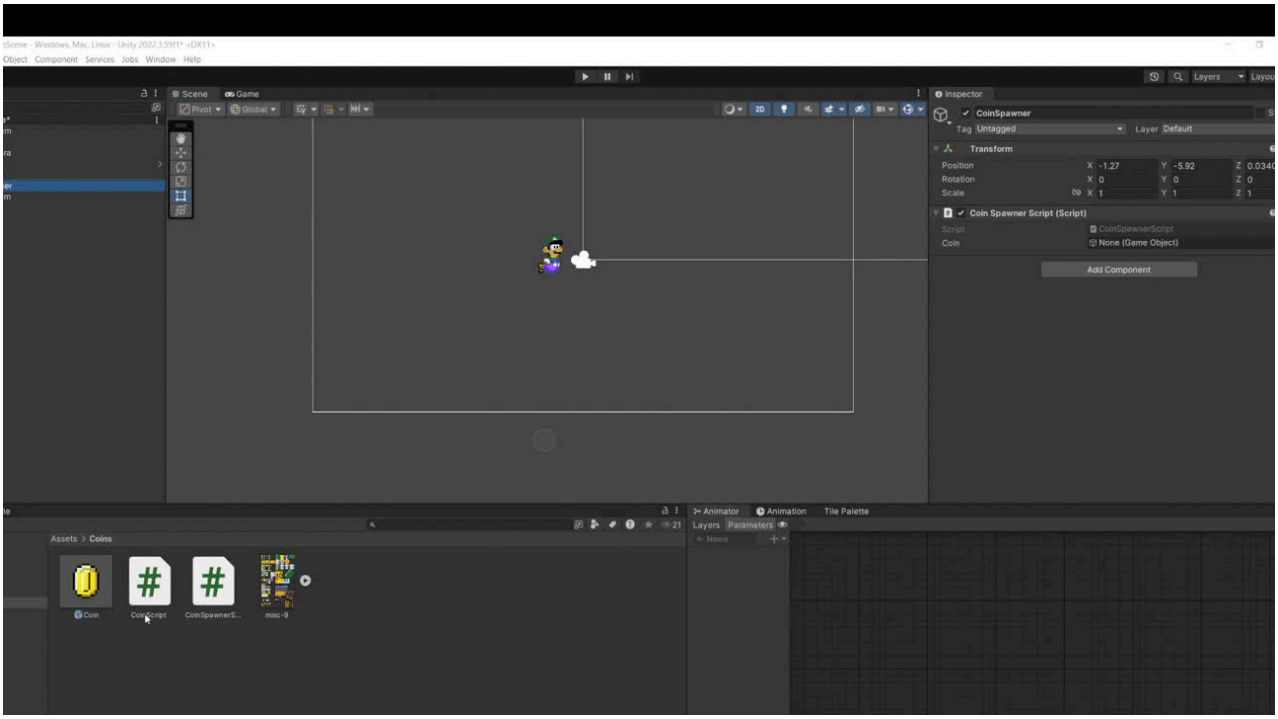
A ce GameObject (nommé CoinSpawner dans mon exemple) nous allons ajouter un script qui, toutes les secondes, fera apparaitre une pièce. Nous allons pour cela utiliser Instantiate() et lui donner en paramètre les éléments suivants :

- Une variable de type GameObject (qui contiendra le GameObject à faire apparaitre)
- Le transform.position (pour que les pièces apparaissent à la position du CoinSpawner)
- Le transform.rotation (pour que les pièces apparaissent avec la même orientation que le CoinSpawner)

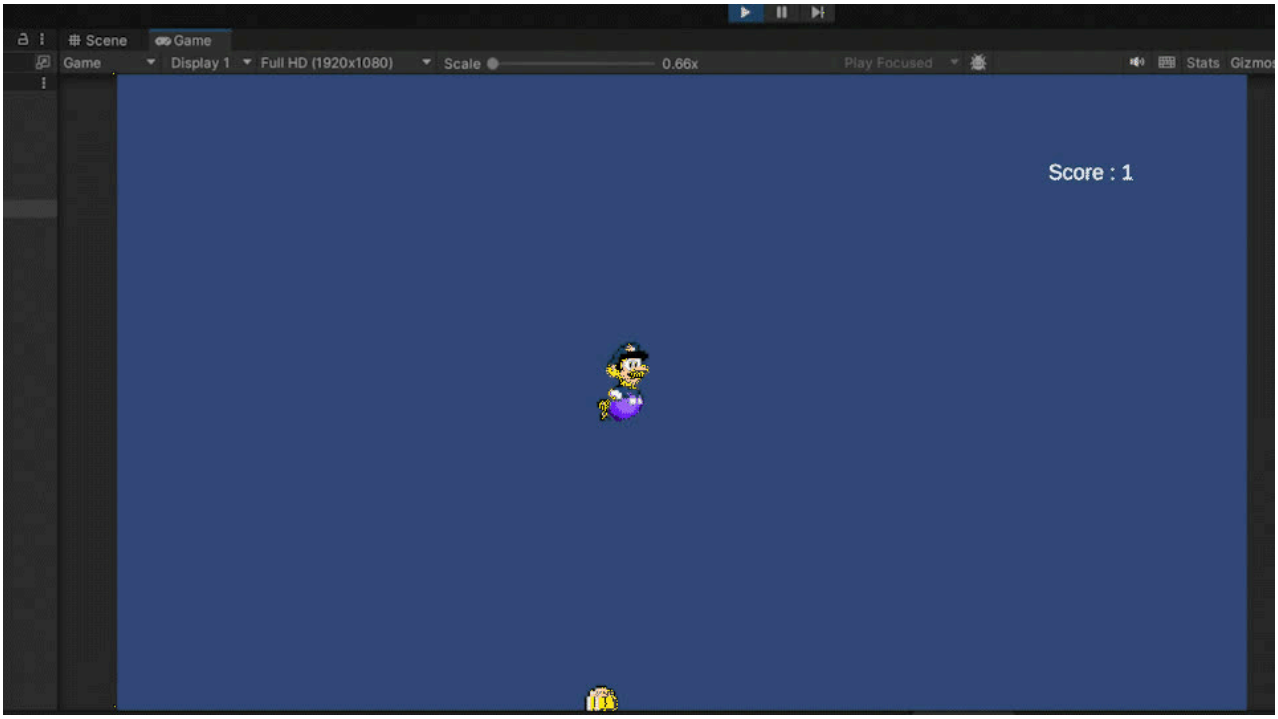
```
public class CoinSpawnerScript : MonoBehaviour
{
    public GameObject coin;
    float timeSinceLastSpawn;

    // Update is called once per frame
    Message Unity | 0 références
    void Update()
    {
        timeSinceLastSpawn += Time.deltaTime;
        if (timeSinceLastSpawn > 1f)
        {
            Instantiate(coin, transform.position, transform.rotation);
            timeSinceLastSpawn = 0;
        }
    }
}
```

Na pas oublier d'attribuer le Prefab à la variable



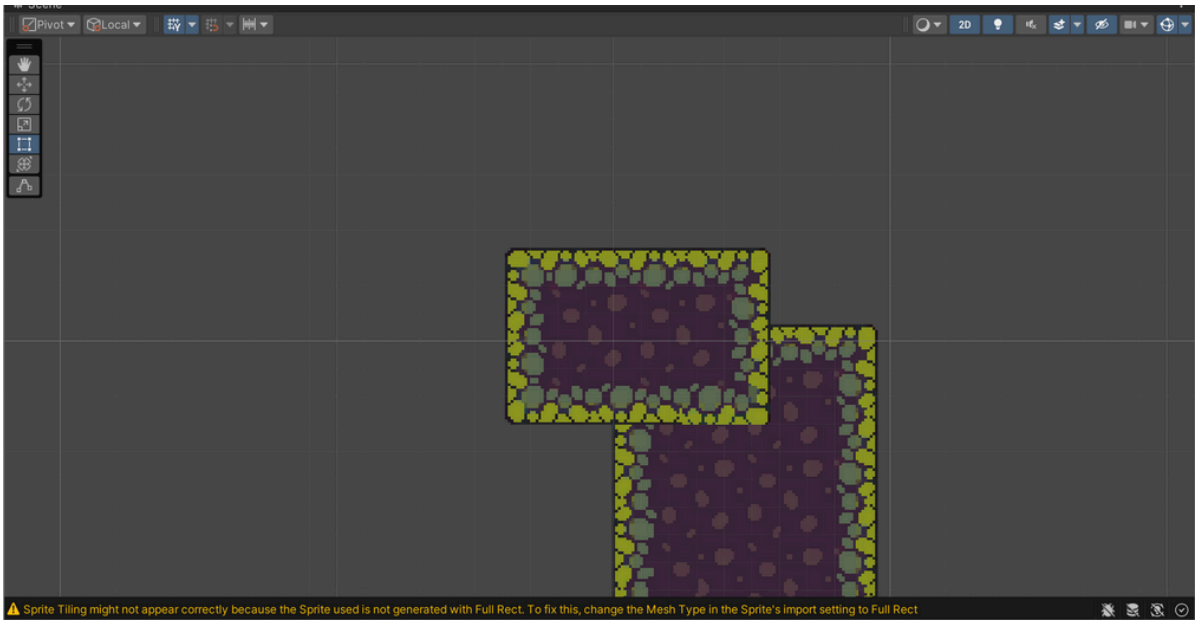
Si nous lançons le jeu nous pouvons voir que les pièces apparaissent bien.



LAYERS

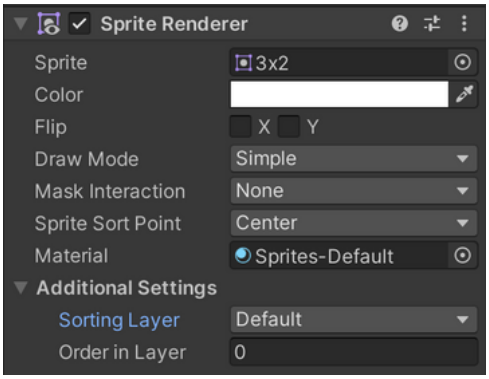
Les layers
La notion de layer (couche) peut sembler un peu bizarre mais elle est en fait assez simple. Les couches sont un outil qui permet de séparer des GameObjects en groupes et d'appliquer des règles différentes à ces groupes. Il en existe deux types qui ont des fonctionnalités différentes : les Sorting Layers et les Layers

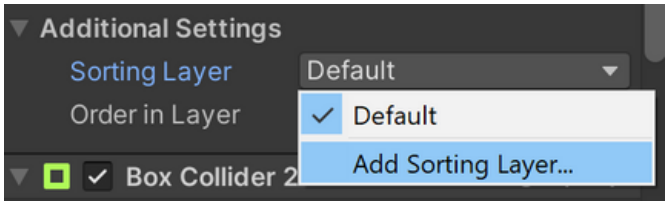
Les sorting layers
Les Sorting Layers sont un outil similaire aux Layers classiques mais qui s'applique spécifiquement aux Sprites. Elles permettent de gérer l'ordre d'affichage des Sprites. Si vous avez plusieurs Sprites superposés c'est le Sprite de la dernière couche qui apparaitra comme "devant".



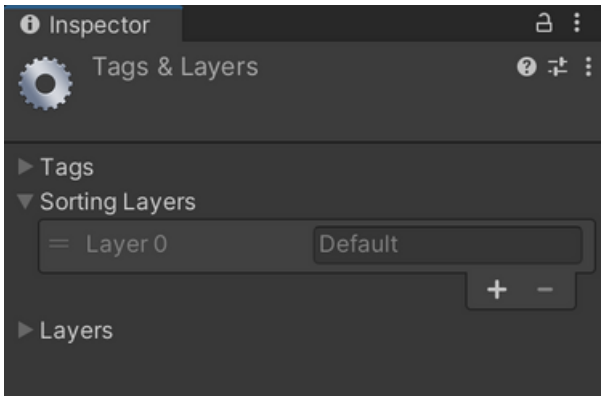
Dans l'exemple ci-dessus, le petit bloc est affiché comme si il était devant le grand bloc.

Pour changer l'ordre dans lequel les éléments apparaissent (et donc quel élément apparait devant un autre) il faut utiliser les Sorting Layers.
Pour ce faire, allez dans le composant Sprite Rendere d'un GameObject, vous verez un paramètre "Sorting Layer"

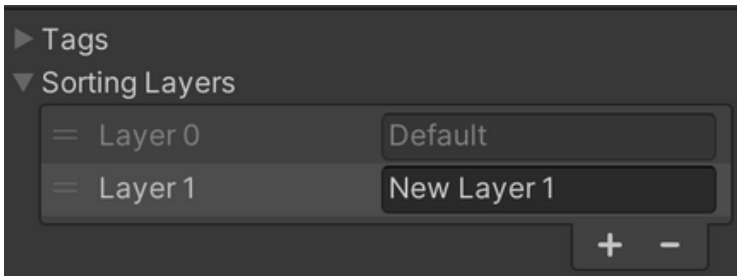




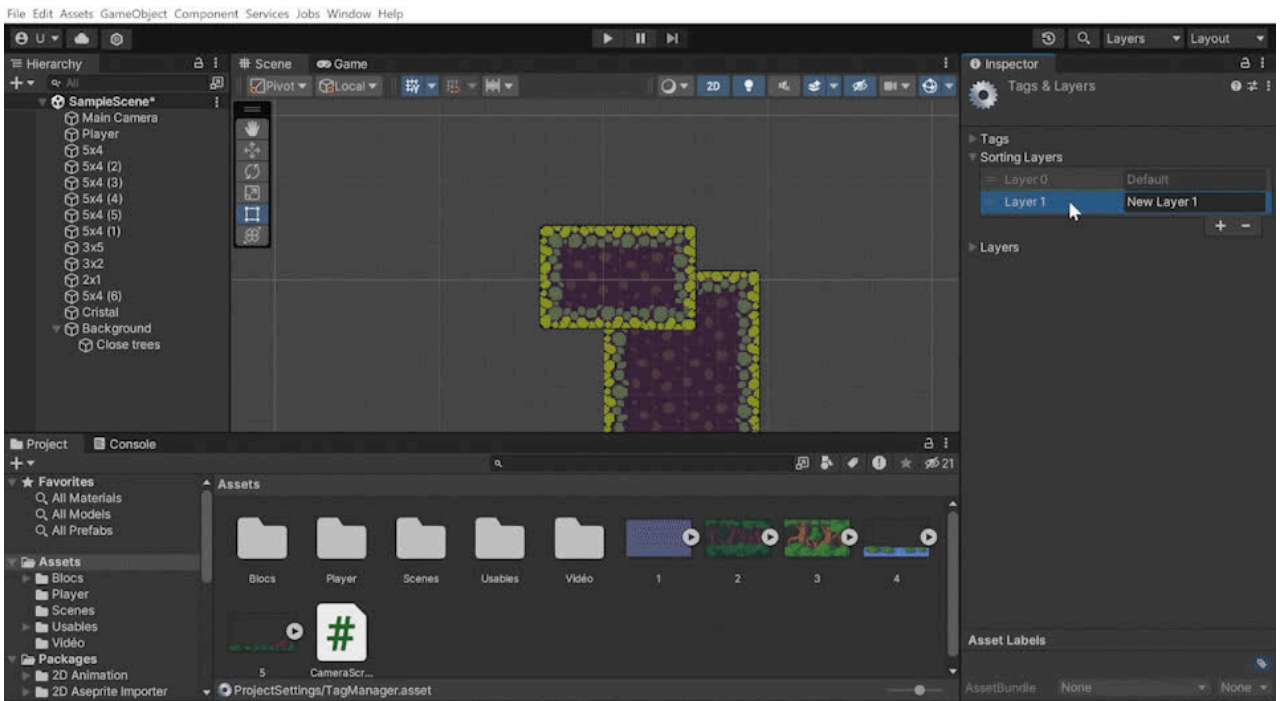
Si vous cliquez dessus vous verrez un menu déroulant avec une seule option et un bloc “Add Sorting Layer ...”, cliquez dessus.



Ce que vous voyez ici est la liste ordonnée des différentes couches de rendu pour les Sprites (elle ne contient que la couche “Default” pour le moment). Pour créer une nouvelle couche, cliquez sur le +

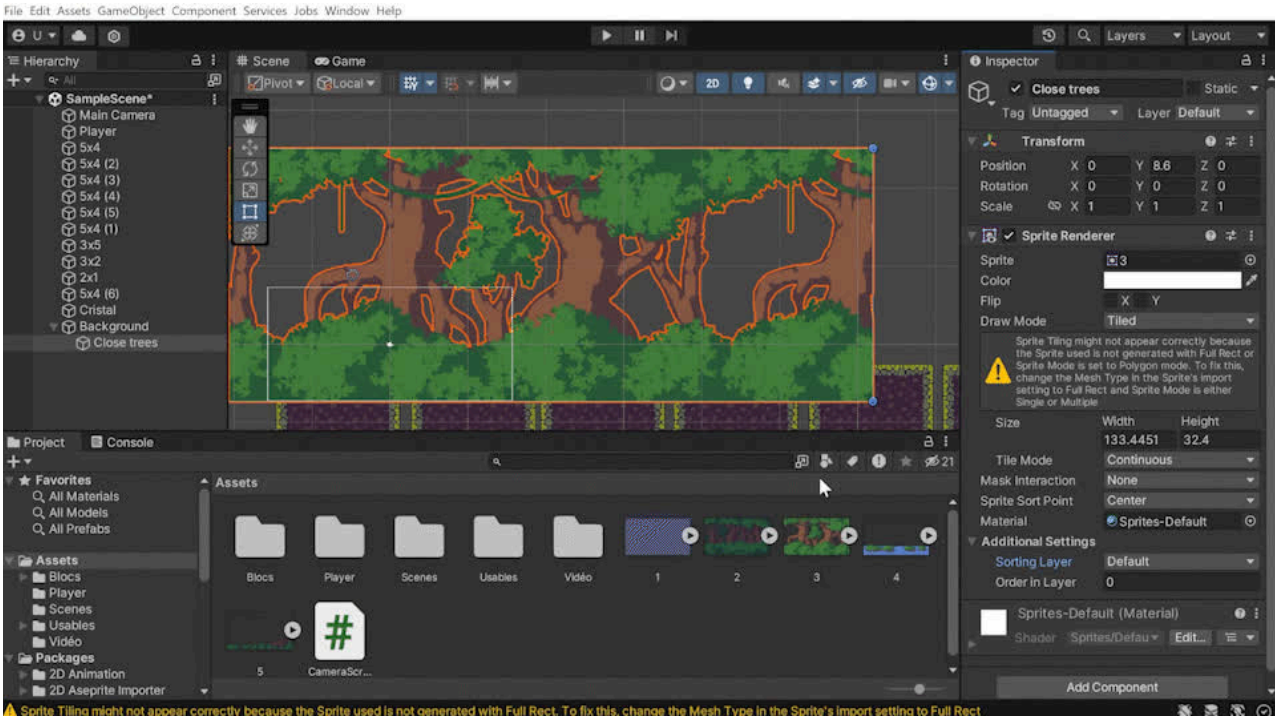


Une nouvelle couche apparaitra, vous pourrez changer le nom de cette couche et changer sa position en cliquant et glissant la couche à la position désirée.



Ce qui est important à retenir avec l'ordre des couches est que les couches les plus basses (celles avec les nombre le plus petit) seront affichées derrière celles les plus hautes (avec les nombres les plus grands).

Il ne nous reste plus qu'à retourner sur les SpriteRenderer et de changer la Sorting Layer vers notre couche tout juste créée.

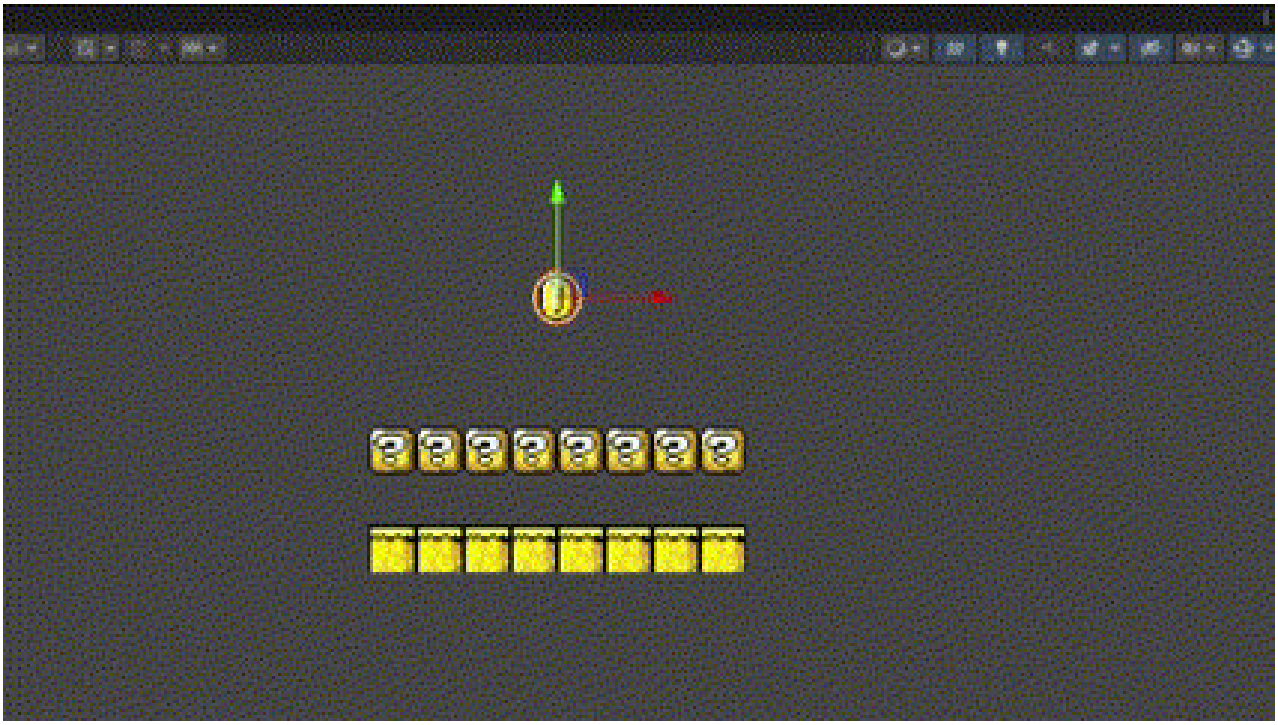


Vous aurez peut-être remarqué qu'il y a aussi un paramètre "Order In Layer" (ordre dans la couche). Ce dernier permet de définir au sein d'une couche un ordre pour les Sprites. Les éléments avec un ordre plus grand seront affichés au=dessus de ceux de la même couche mais avec un ordre plus petit.

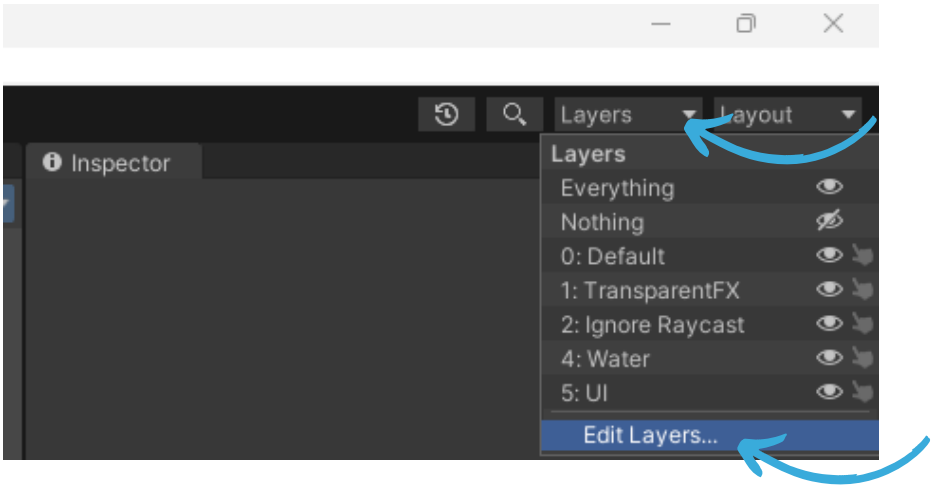
Les layers (tout court)
Les layers permettent de grouper les GameObjects et d'appliquer des logiques différentes pour chaque groupe. Elles ont de nombreuses utilités différentes. Je vais ici montrer un cas d'utilisation courant.

Il arrive souvent que nous voulions que certains GameObjects puissent entrer en collision avec certains GameObjects mais passent à travers d'autres.

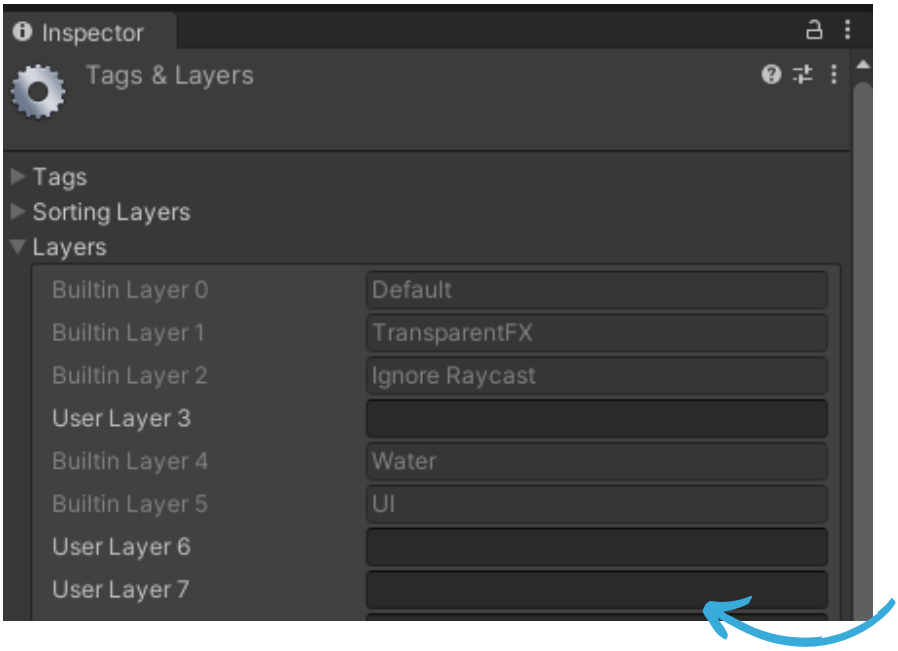
Imaginons que dans la scène ci-dessus nous ayons deux plateformes avec chacune un BoxCollider et une pièce avec un CapsuleCollider et un RigidBody. Nous voulons que la pièce tombe et passe à travers la plateforme du haut. En l'état la pièce tombe mais s'arrête à la première plateforme.



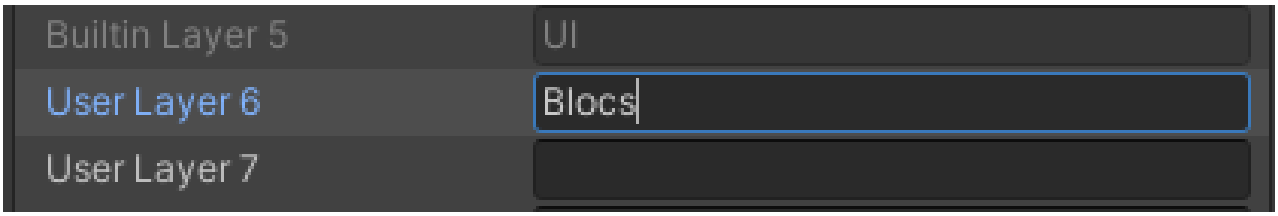
La première étape va être de créer une nouvelle Layer. Pour ce faire, cliquons sur “Layers” en haut à droite de l’éditeur puis sélectionnons “Edit Layers”.



Vous verrez alors dans l’inspecteur les différentes Layers, il y en a forcement de base 5 (Default, TransparentFX, Ignore Raycast, Water et UI).

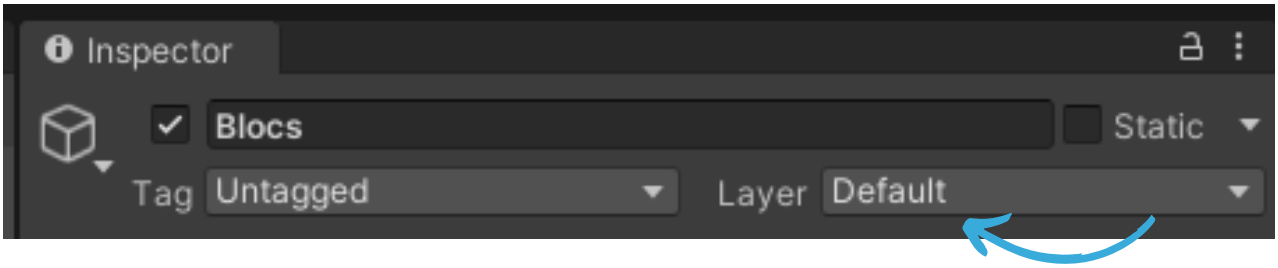


Pour en créer une nouvelle il nous suffit de cliquer sur un des espaces vierges et écrire le nom de cette nouvelle Layer.



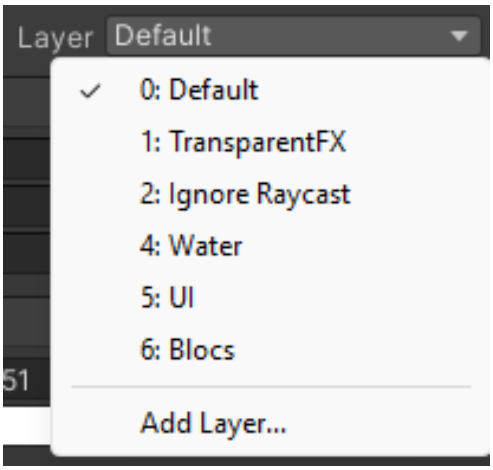
Maintenant nous allons pouvoir assigner des GameObjects à des Layers. Un fois un GameObject assigné à une Layer, toutes les règles spécifiques à cette Layer vont s'appliquer sur le GameObject.

Pour ce faire, sélectionnez le GameObject à assigner et regardez en haut à droite de l'inspecteur. Vous verrez un menu avec "Layer" devant.



Nous pouvons voir que le GameObject est déjà dans la Layer "Default" qui, comme son nom l'indique, est la couche par défaut.

Si nous cliquons sur le menu déroulant nous retrouvons les Layers qui existent pour ce projet.

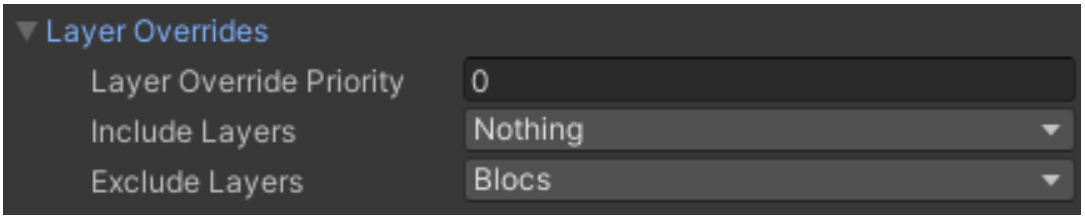


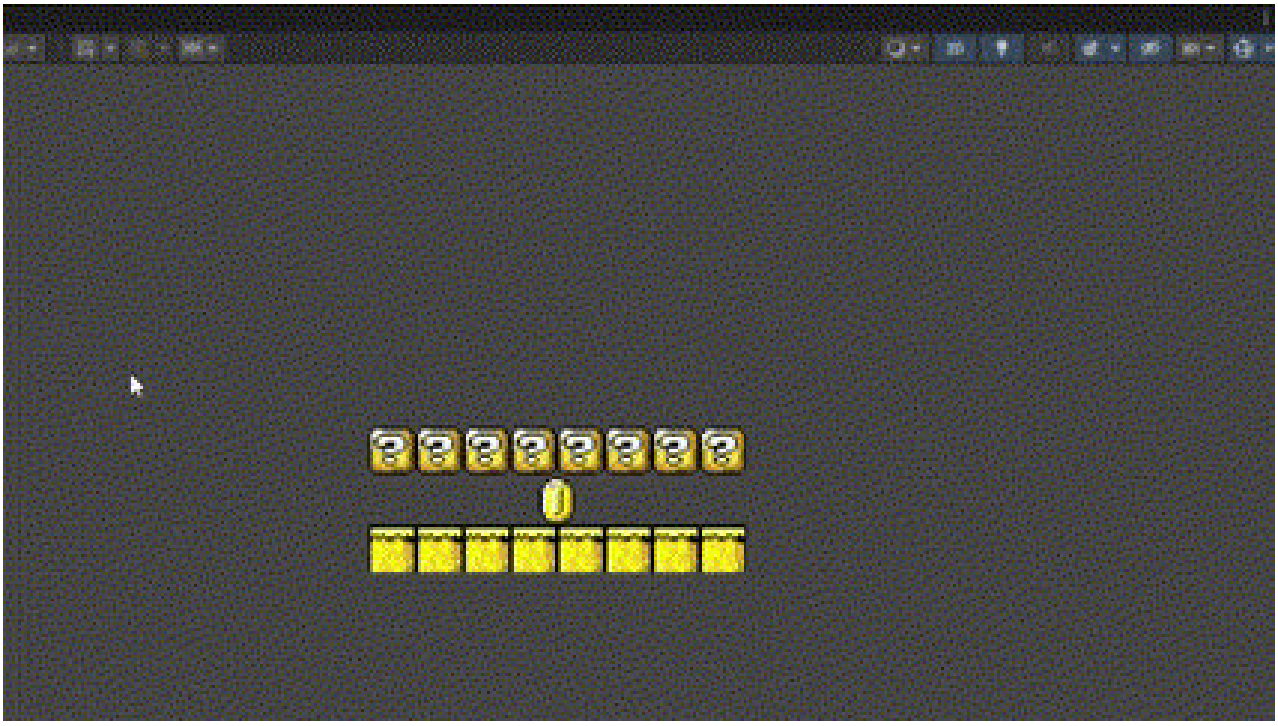
Il nous suffit donc de sélectionner la Layer à laquelle nous voulons assigner ce GameObject (dans mon exemple je vais assigner la plateforme du haut à "Blocs").

Un fois les GameObjects assignés aux bonne Layers il faut encore définir des logiques particulières à appliquer à ces Layers. Dans le cas de mon exemple je veux que la pièce ignore les collisions avec la Layer “Blocs”.

Je vais donc sélectionner la pièce et regarder dans son composant CapsuleCollider sous “Layer Overrides”.

Je vais ici modifier le paramètre “Exclude Layers”, ce dernier définit des Layers à ignorer pour les collision. Je vais donc le paramétrer pour ignorer les collisions avec la Layer “Blocs”.



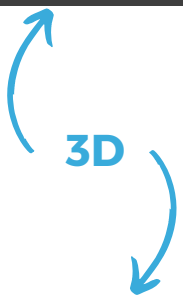


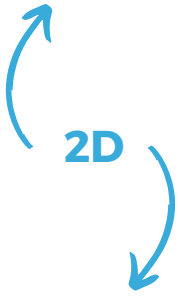
ONCOLLISION ET ONTRIGGER

Il est courant dans des jeux que des choses se produisent quand deux objets donnés se touchent. Ou encore quand un objet se trouve dans une zone donnée. Il existe pour cela deux fonctions utiles.

Les fonctions OnCollision et OnTrigger sont appelées sous certaines conditions liées au Collider attaché au même GameObject que le script.

Toutes les fonctions que nous allons voir ici existe en deux versions : une pour la 2D et une pour la 3D. Les versions 3D ne fonctionnent qu'avec les Colliders 3D (ceux sans précision dans le nom). Tandis que les version 2D ne fonctionnent qu'avec les Colliders 2D.






```
private void OnCollisionEnter(
private void OnCollisionExit(
private void OnCollisionStay(
```

```
private void OnCollisionEnter2D(
private void OnCollisionExit2D(
private void OnCollisionStay2D(
```

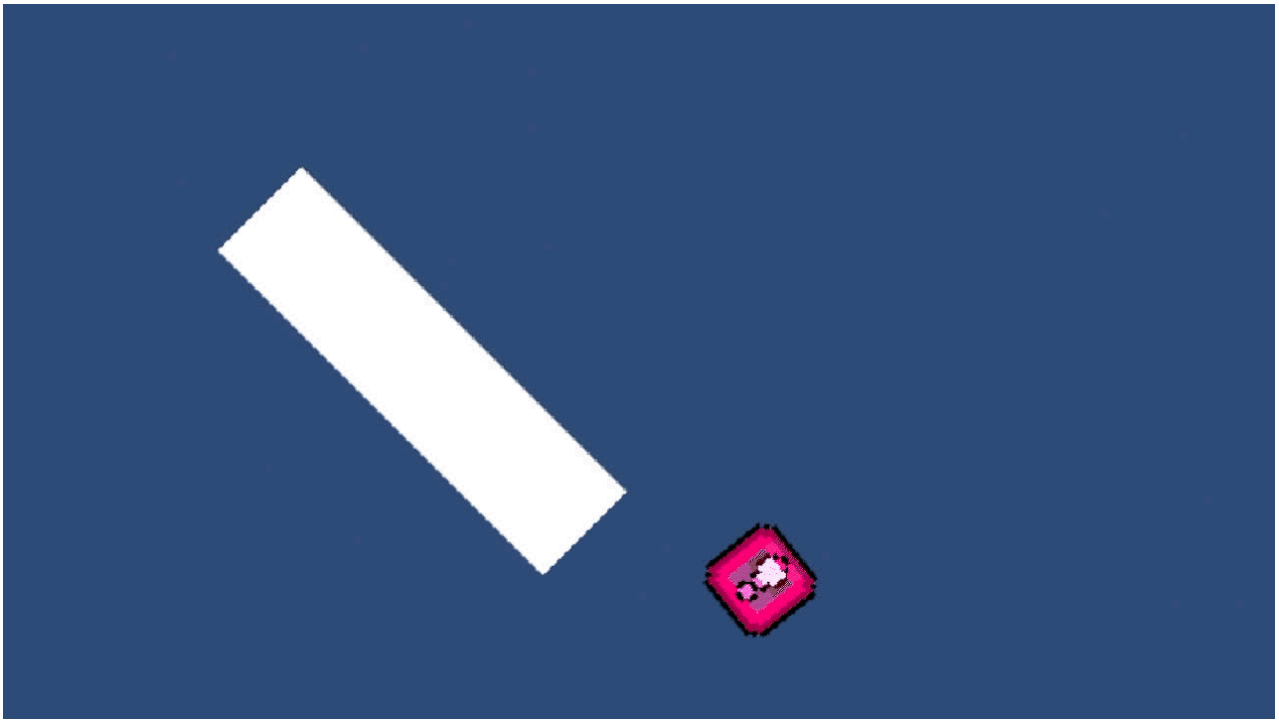
Comme vous pouvez le voir il existe à chaque fois 3 fonctions différentes, chacune d'entre elles est appelé à un moment donné lié à la collision.

Nous allons voir quels sont ces moments.

Attention : Pour toutes les fonction commençant par OnCollision... il est nécessaire que deux Colliders entrent en collision. Leur paramètre IsTrigger dont donc être décochés.
Pour toutes le fonctions Commençant par OnTrigger... il est seulement nécessaire que le collider attaché au même GameObject que le script soit avec IsTrigger coché.

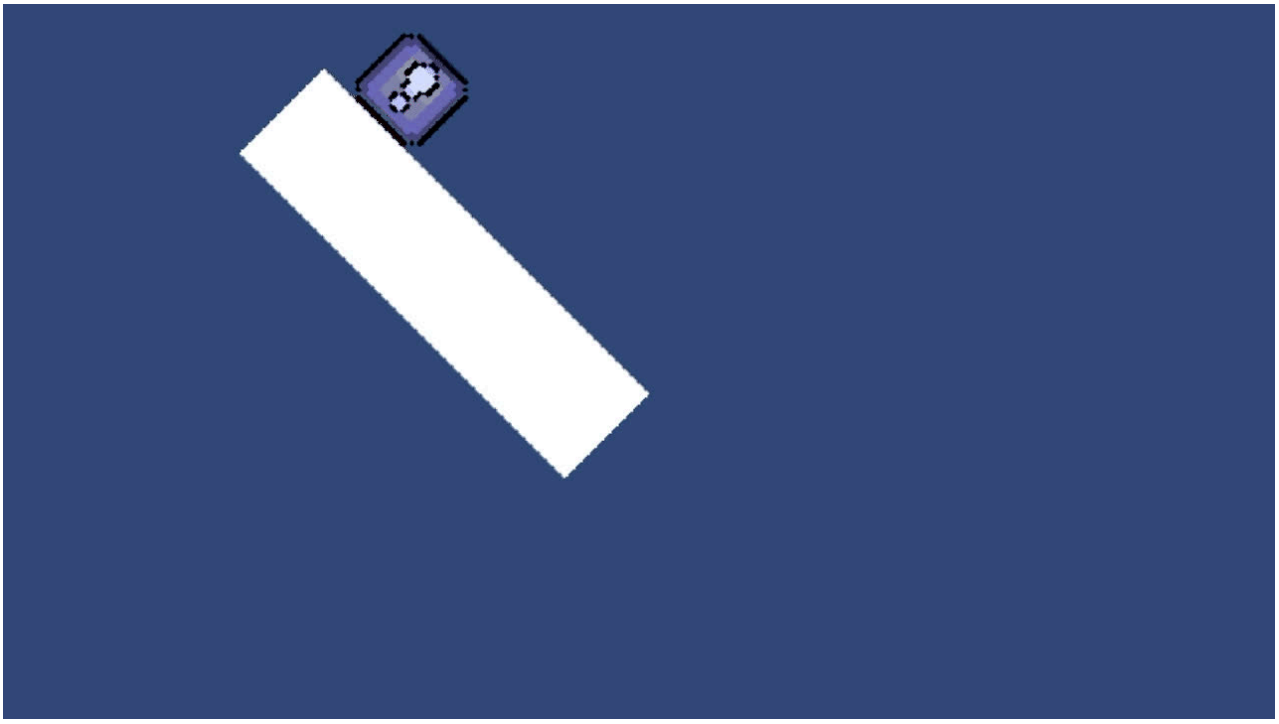
OnCollisionEnter

OnCollisionEnter et son pendant 2D sont appelés quand un Collider attaché au même GameObject que le script entre en collision avec un autre Collider.



OnCollisionExit

OnCollisionExit et son pendant 2D sont appelés quand un Collider attaché au même GameObject que le script ne touche plus un Collider qu'il touchait précédement.

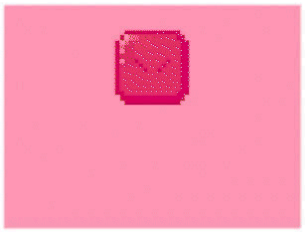


OnCollisionStay

OnCollisionStay et son pendant 2D sont appelés à chaque frame à laquelle un Collider attaché au même GameObject que le script touche un autre Collider.

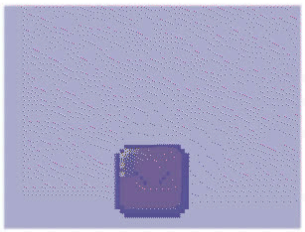
OnTriggerEnter

OnTriggerEnter est appelé quand un Collider rentre dans la zone décrite par un Collider en mode trigger attaché au même GameObject que le script.



OnTriggerExit

OnTriggerEnter est appelé quand un Collider sort de la zone décrite par un Collider en mode trigger attaché au même GameObject que le script.



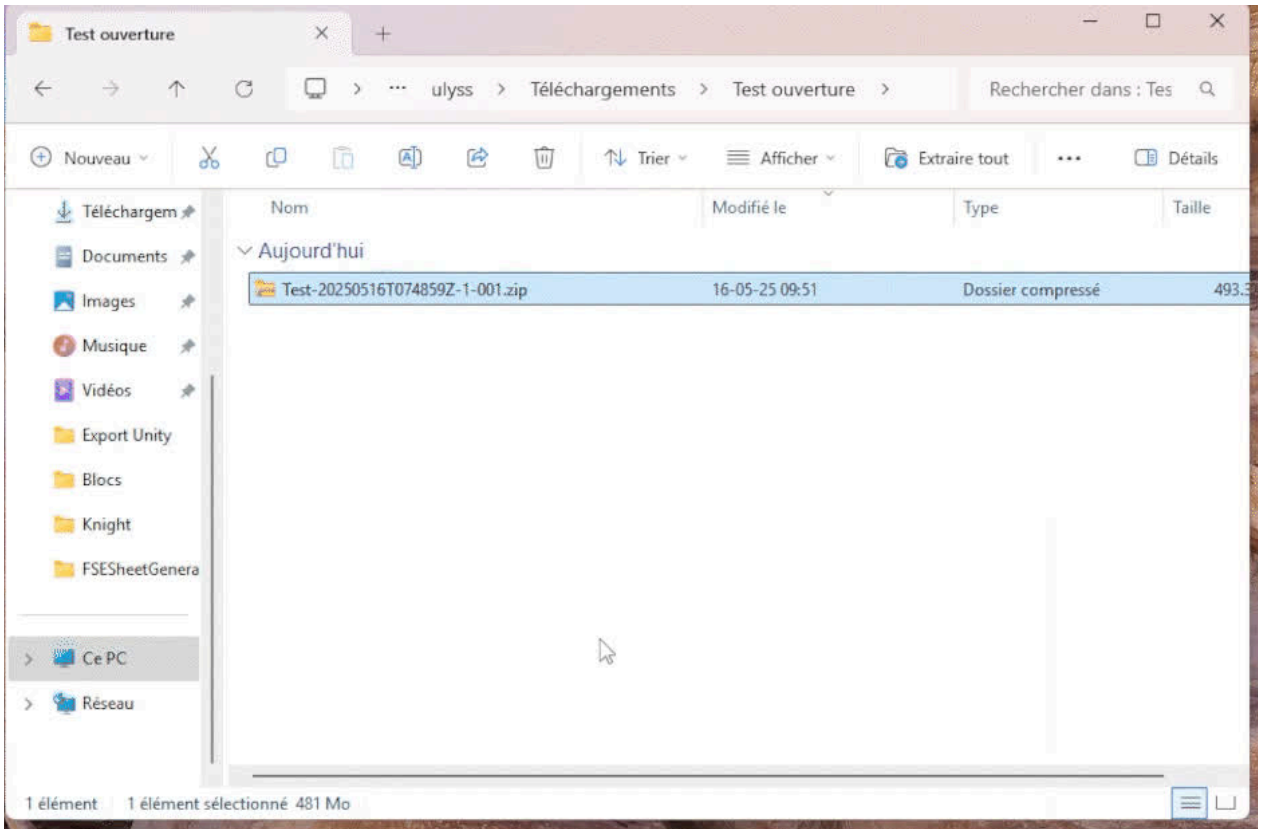
OnTriggerStay

OnTriggerEnter est appelé à chaque frame à laquelle un Collider se trouve dans la zone décrite par un Collider en mode trigger attaché au même GameObject que le script.

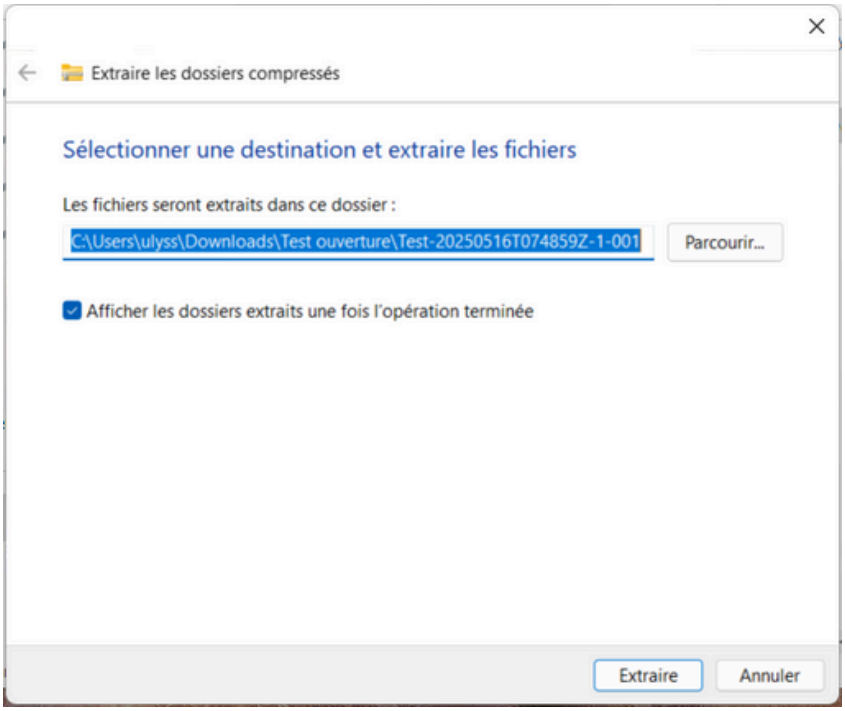
OUVRIR UN PROJET

Si vous avez téléchargé un projet Unity en ligne ou si vous transférez un projet d'un ordinateur à un autre il va vous falloir l'ouvrir dans Unity.

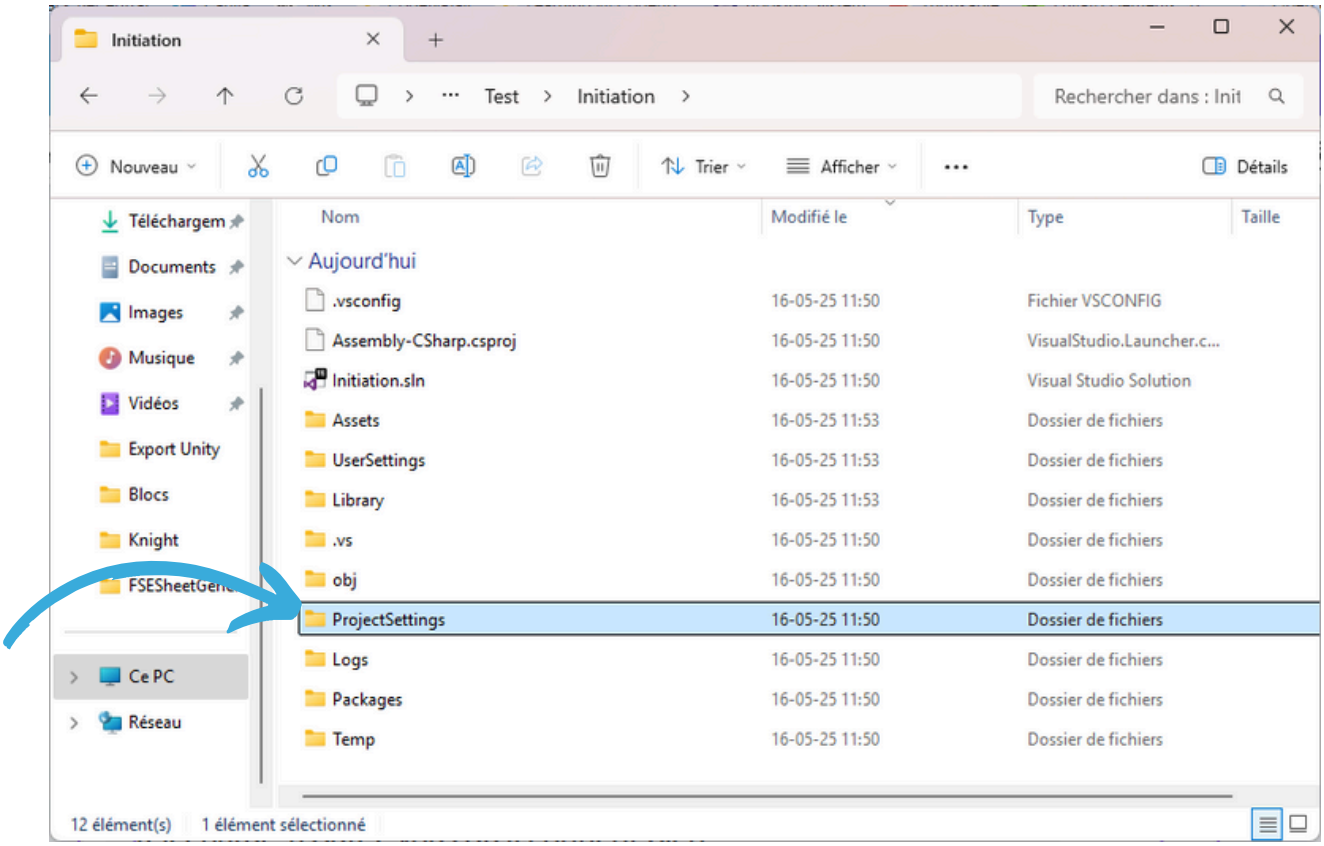
Avant toute chose, si vous avez téléchargé le projet il ne serait pas surprenant que vous ayez un fichier en .zip. Si tel est le cas il faut d'abord le dossier. Pour ce faire faites clic droit sur le .zip et sélectionnez "Extraire tout..."



Vous verrez une fenêtre s'ouvrir, elle vous permet de choisir le dossier dans lequel les fichiers seront placés. Choisissez en un à votre convenance.

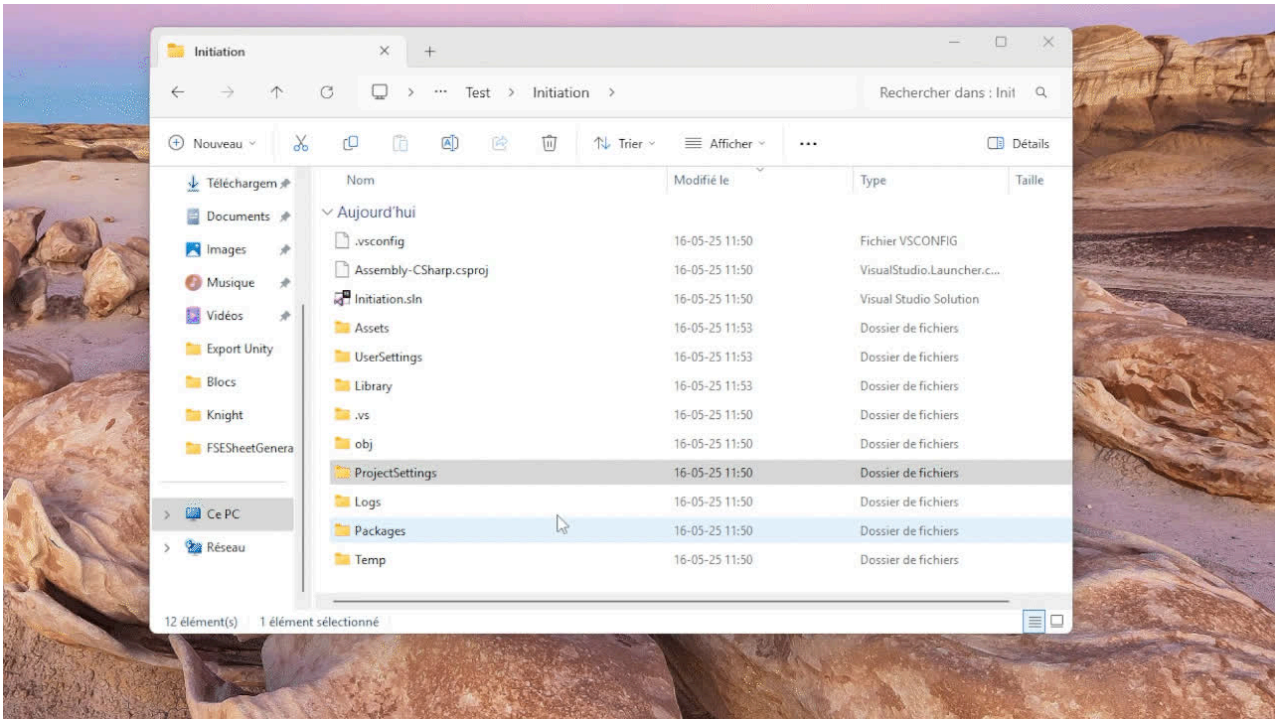


Une fois que vous avez extrait le dossier ou si vous avez pu directement le téléchargé, assurez-vous qu'il contient bien un dossier nommé "ProjectSettings".

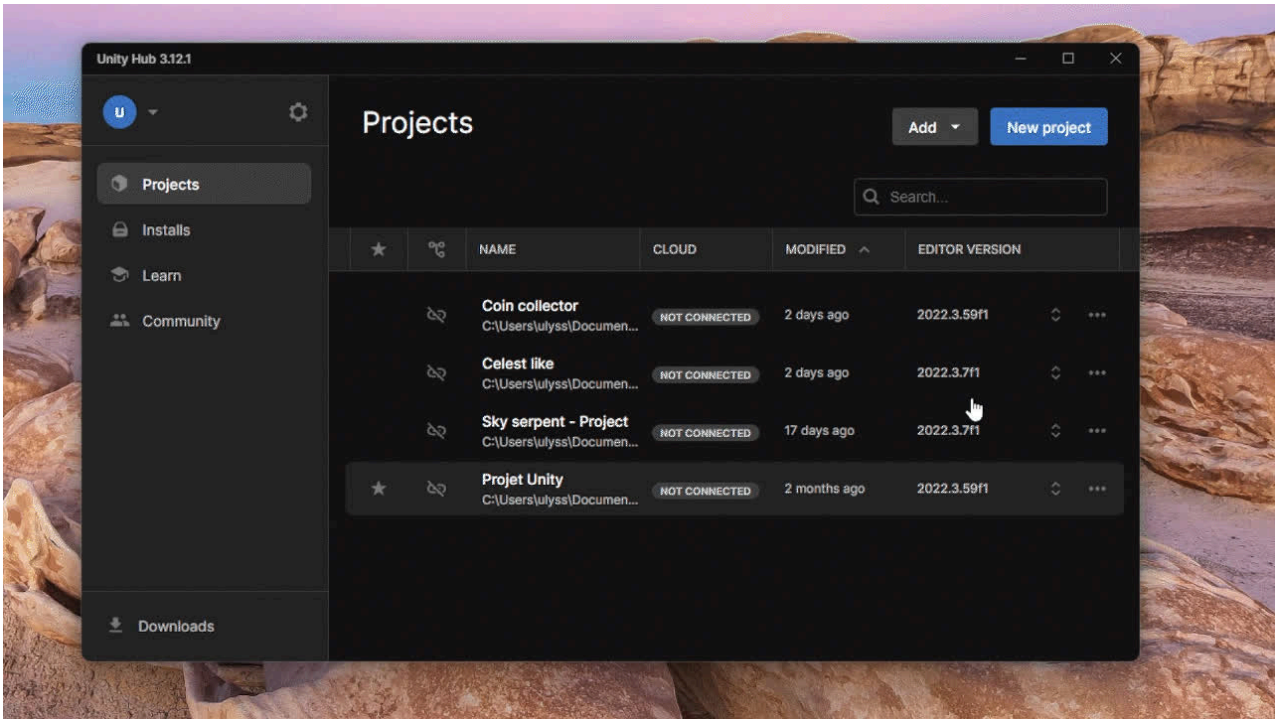


S'il ne contient pas de dossier "ProjectSettings" cherchez dans les dossiers jusqu'à trouver le bon.

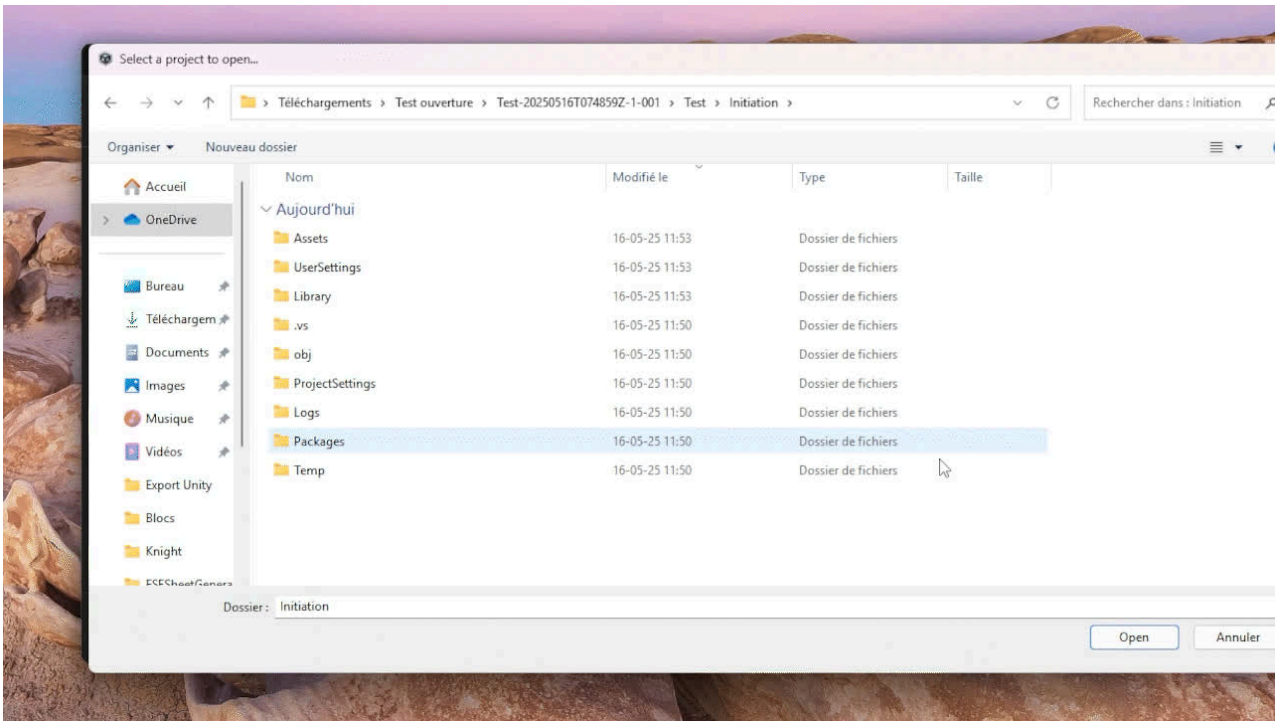
Un fois dans le bon dossier, cliquez sur l'adresse du dossier et copiez la.



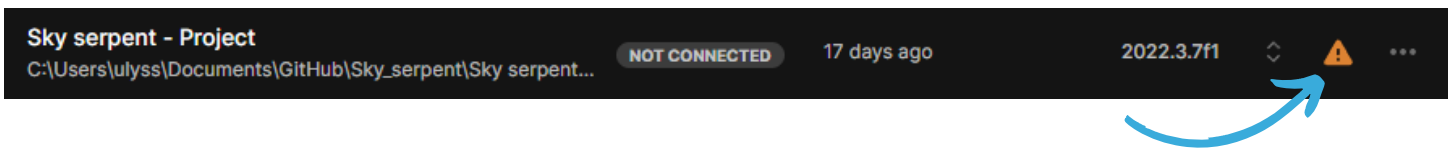
Avec l’adresse copiée ouvrez Unity Hub et cliquez sur “Add” puis “Add project from disk”.



Une fenêtre va s’ouvrir, collez l’adresse dans la barre et appuyez sur la touche Enter. Cliquez ensuite sur “Ouvrir”.



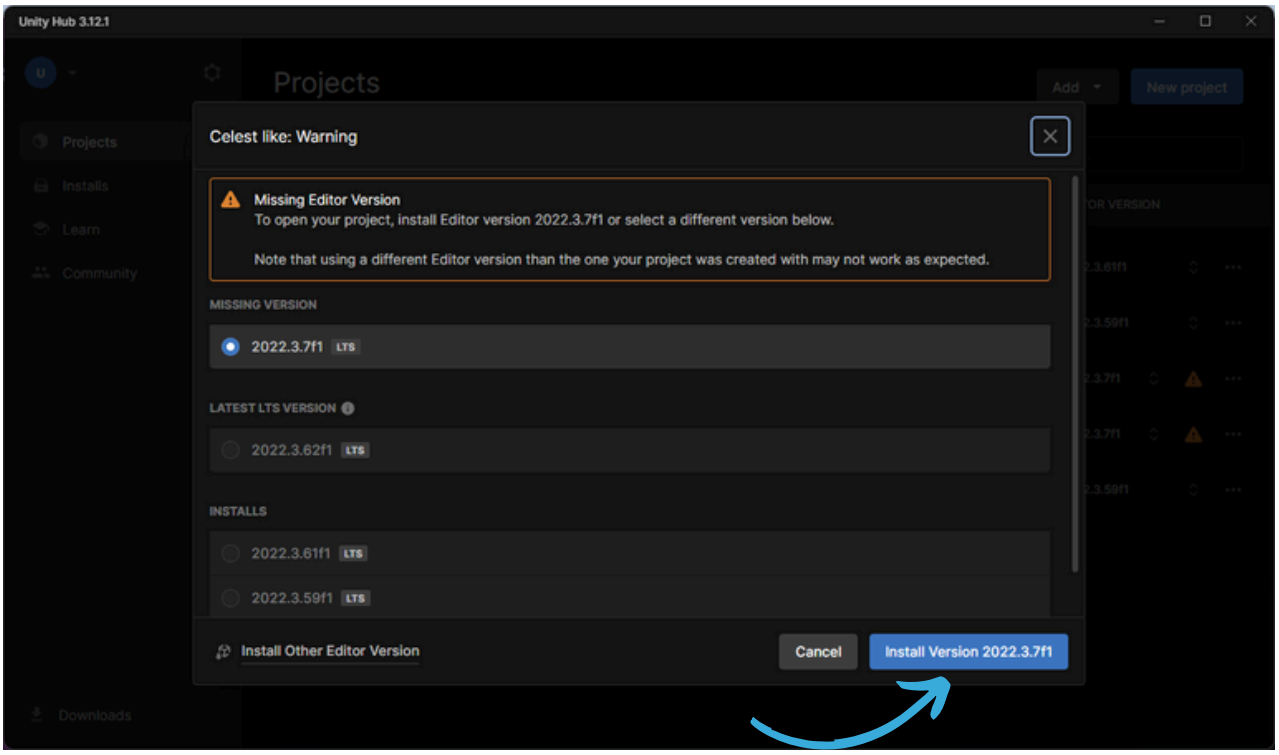
Vous pourrez voir que le projet vient de se rajouter à votre liste. Il se pourrait que vous voyiez un panneau “Warning”, si c’est la cas, lisez la prochaine page.



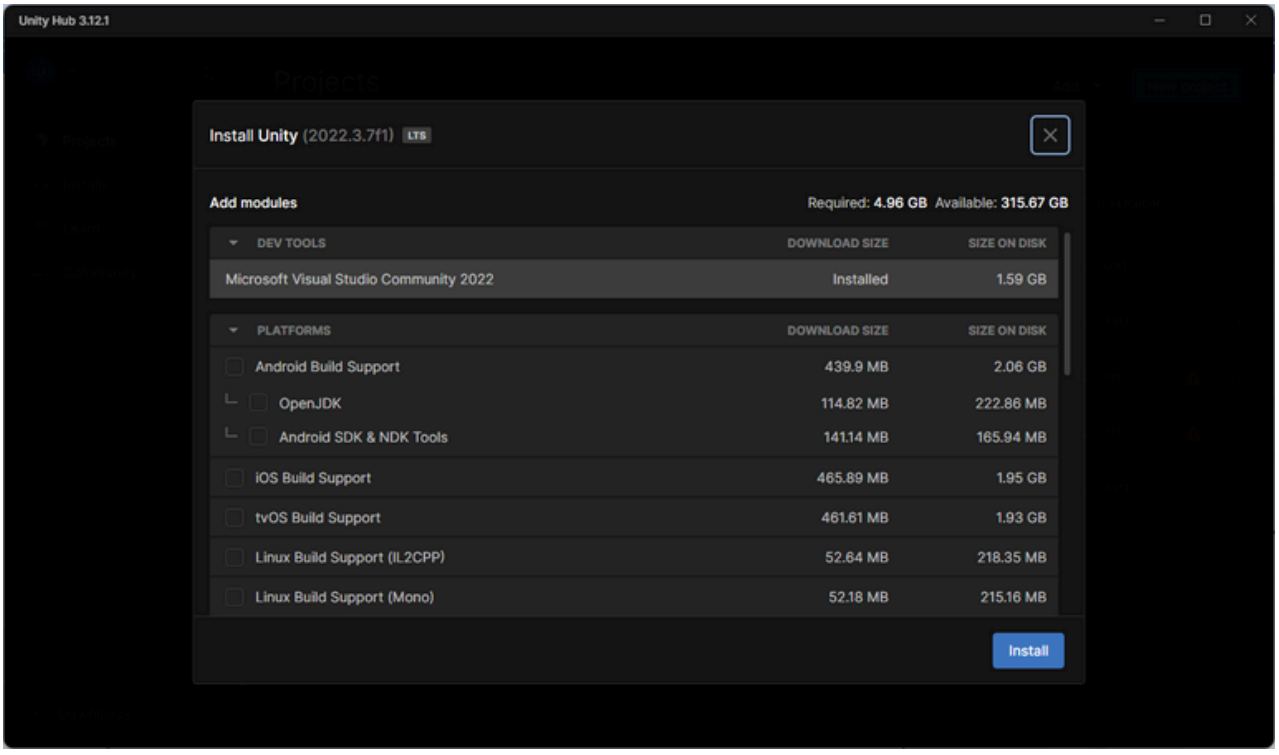
Si vous voyez un panneau “Warning” à la droite du projet cela signifie que ce projet nécessite une version de Unity que vous n’avez pas installée.

Pour régler ce problème, cliquez sur le projet comme pour l’ouvrir. Une fenêtre s’ouvrira vous demandant d’installer la version manquante.

Cliquez donc sur “Install Version ...”.

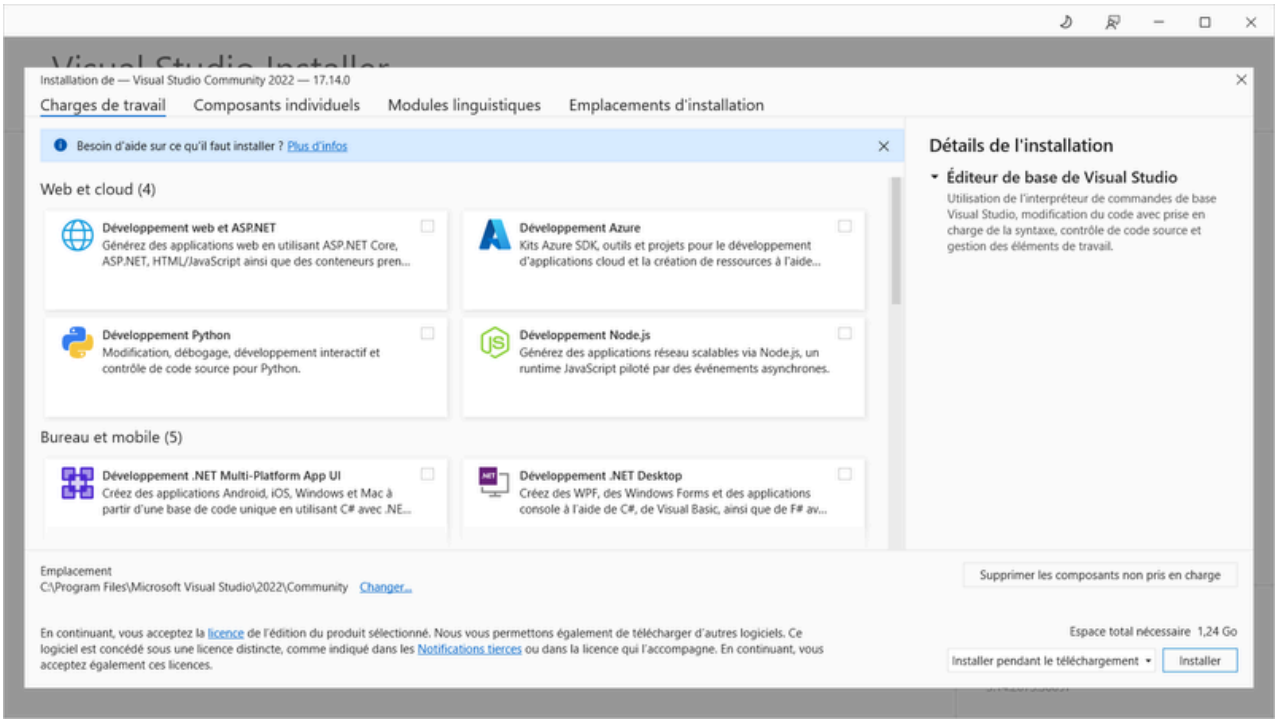


Cliquez ensuite sur “Install”.

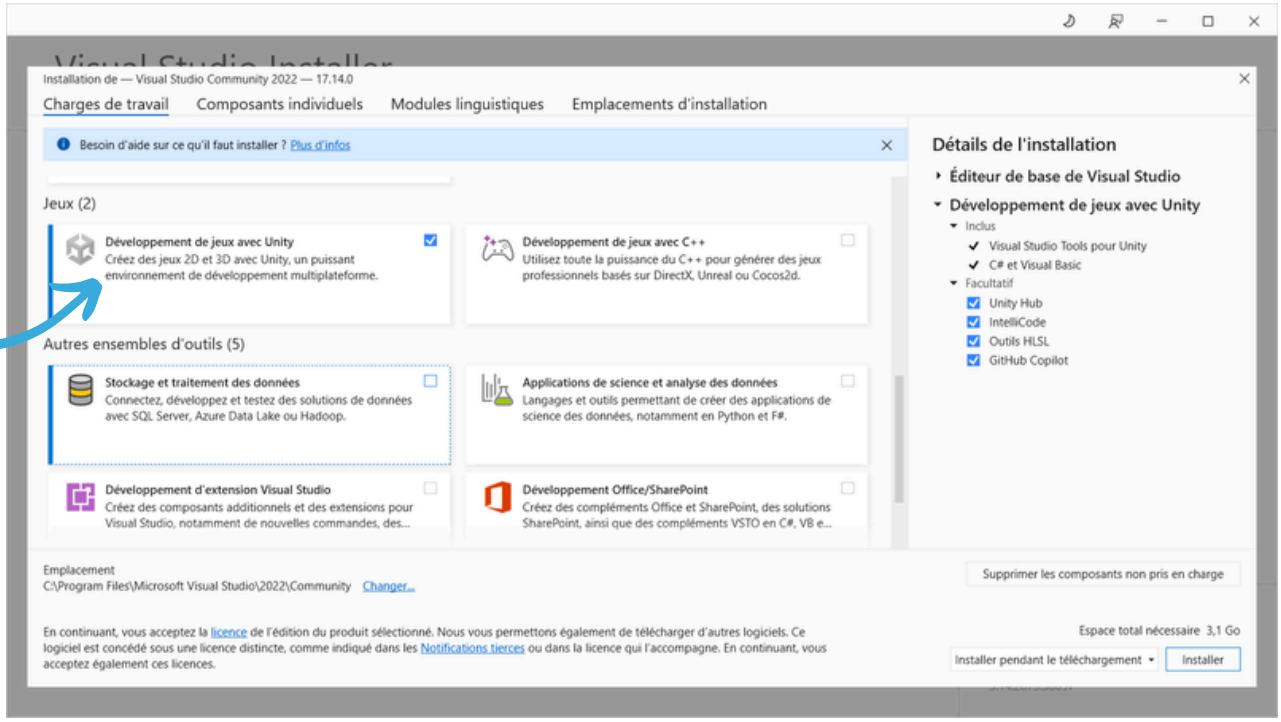


Il ne vous restera plus qu’à attendre que l’éditeurs s’installe puis de lancer votre projet.

Si vous n’avez pas encore Visual Studio sur votre ordinateur il vous sera proposé de l’installer. Durant l’installation vous arriverez sur un menu permettant de choisir des éléments supplémentaires.



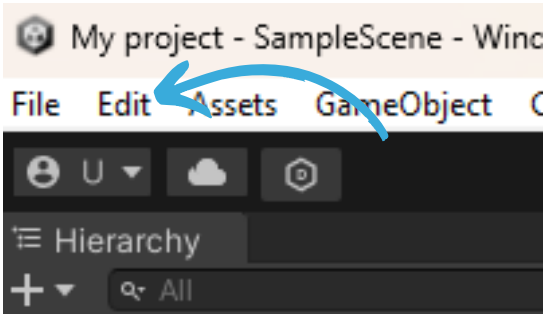
Quand vous arriverez dans ce manu, scrollez vers le bas jusqu’à voir “Développement de jeux avec Unity”. Cochez la case qui y est associée puis cliquez sur “Installer”.



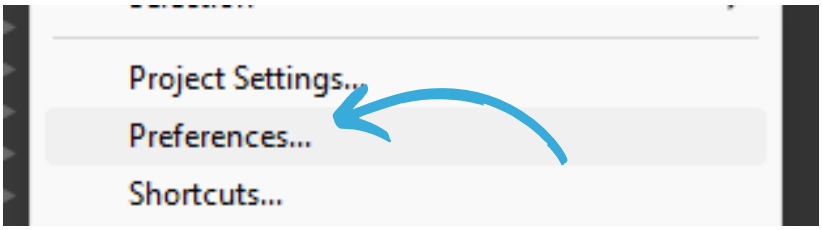
PAS D'AUTOCOMPLÉTIONS

Il peut arriver que vous n'ayez pas d'autocomplétions dans Visual Code. Pour régler ce problème il vous faut aller dans les paramètres de Unity.

Pour ce faire, cliquez sur "Edit" en haut à gauche de votre fenêtre Unity.

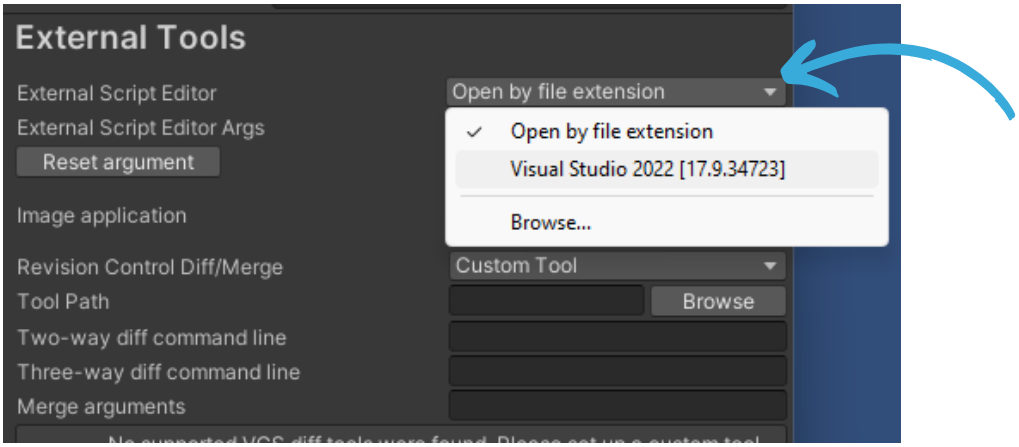


Cliquez ensuite sur "Preferences..."

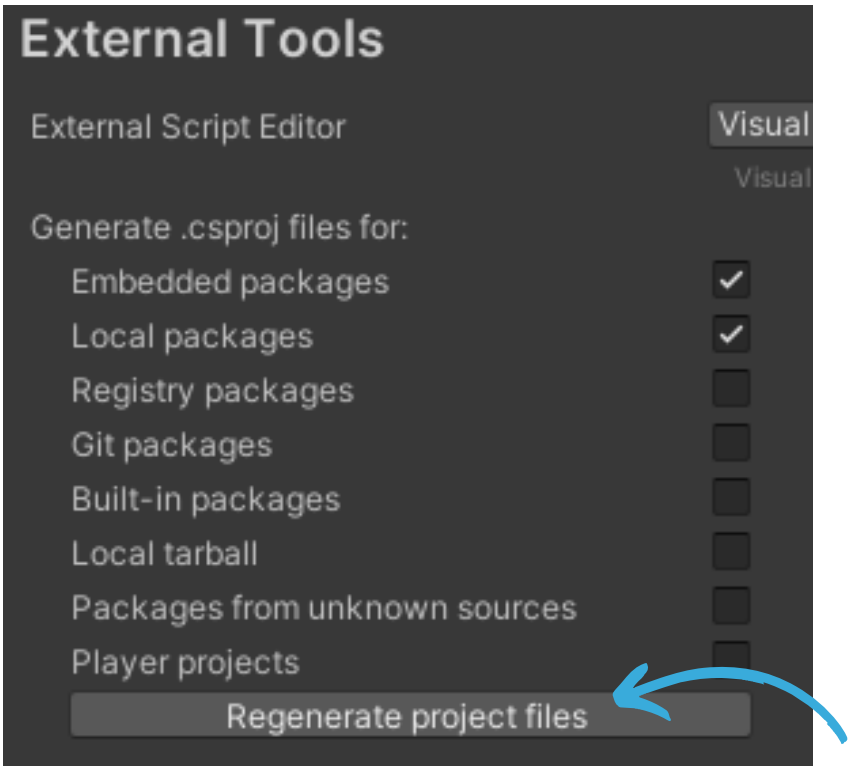


Cliquez ensuite sur "External Tools"

Cliquez sur le bloc de sélection "External Script Editor" et sélectionnez Visual Studio.



Cliquez ensuite sur “Regenerate project files”



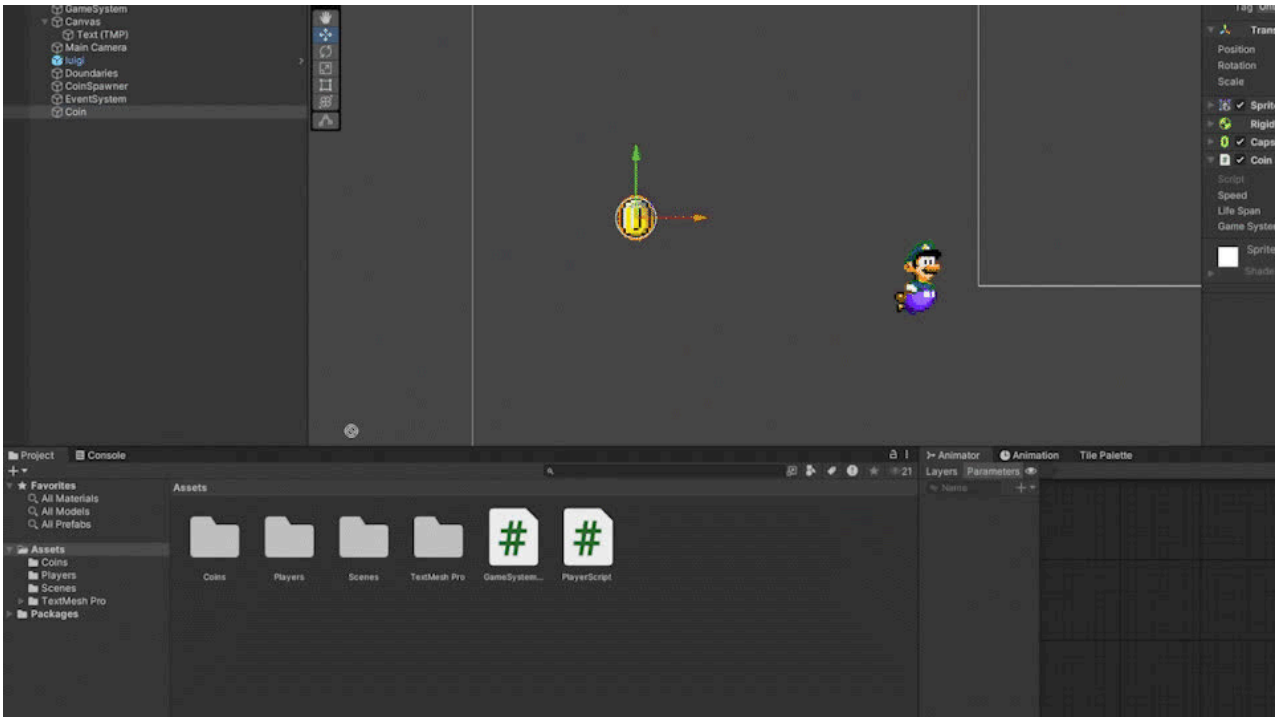
Fermez et rouvrez Visual Studio et l’autocomplétions devrait fonctionner.

PREFAB

Les Prefab (pour prefabricated ou préfabriqué en français) sont des GameObjects que vous aurez créé et enregistré pour les réutiliser plus tard.

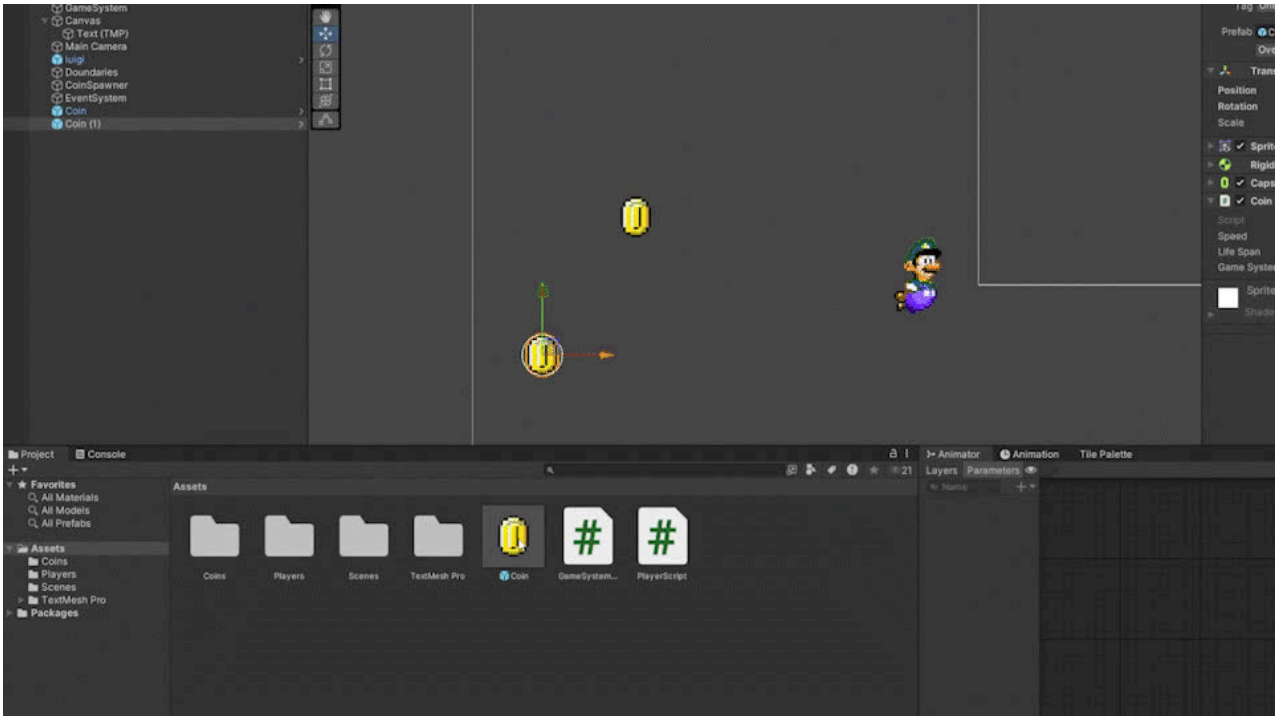
Utiliser des Prefabs comporte de nombreux avantages en terme de facilité de travail, ils sont très utiles quand vous devez avoir dans votre jeux une grande quantité de fois des GameObjects identiques.

Créer un Prefab est très simple, pour cela, glissez-déposez un GameObject dans l'explorateur de fichiers.



Vous verrez alors le Prefab dans l'explorateur de fichier. Vous pouvez aussi voir que le GameObject à partir duquel il a été créé a changé d'apparence dans la hiérarchie.

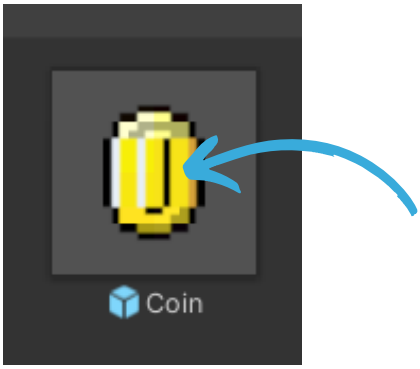
Si vous glissez-déposez le Prefab dans la scène ou dans la hiérarchie vous verrez qu'un copie du Prefab sera créée.



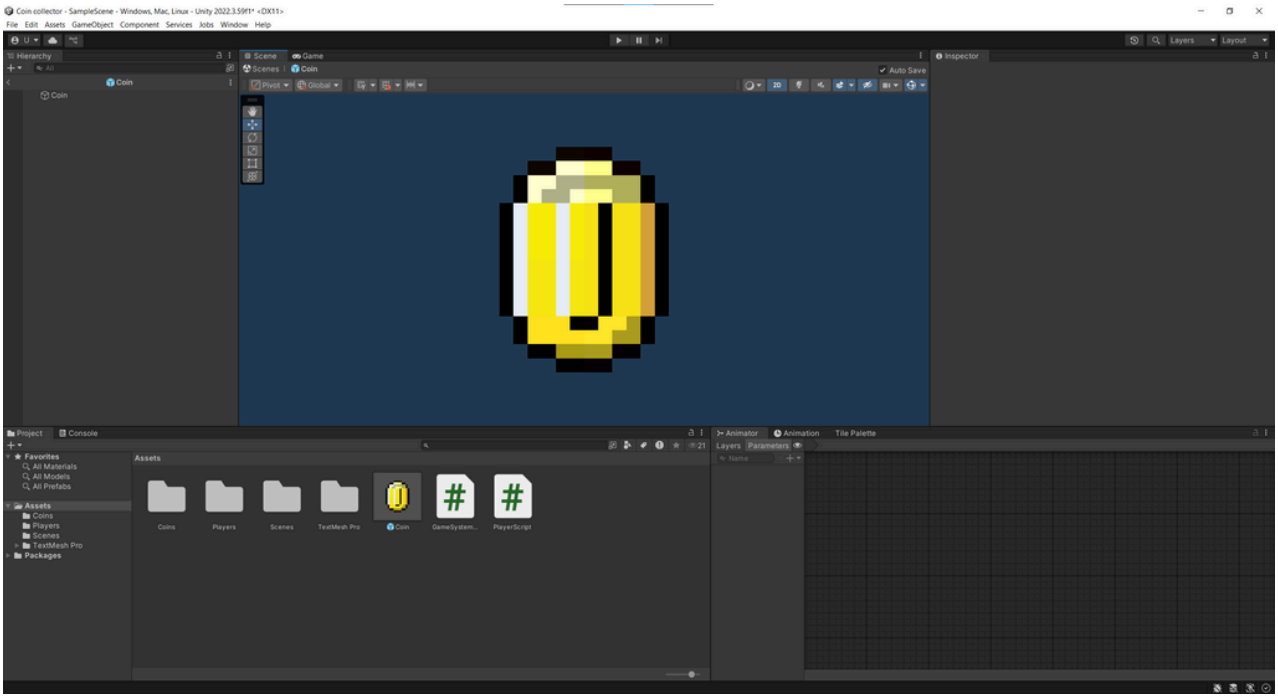
Maintenant vous pourriez vous demander quel est l'intérêt de faire de cette manière, ne serait-il pas plus simple de juste copier-coller le GameObject ?

Imaginons que vous ayez développé un niveau d'un jeu de plateforme avec des dizaines de pièces comme celle de l'exemple. Après tous vos efforts vous vous décidez de changer le Sprite des pièces. Si vous avez copié-collé les GameObjects vous serrez obligé de changer le Sprite dans chacune des pièces, ce qui risque de vous prendre un temps certain.

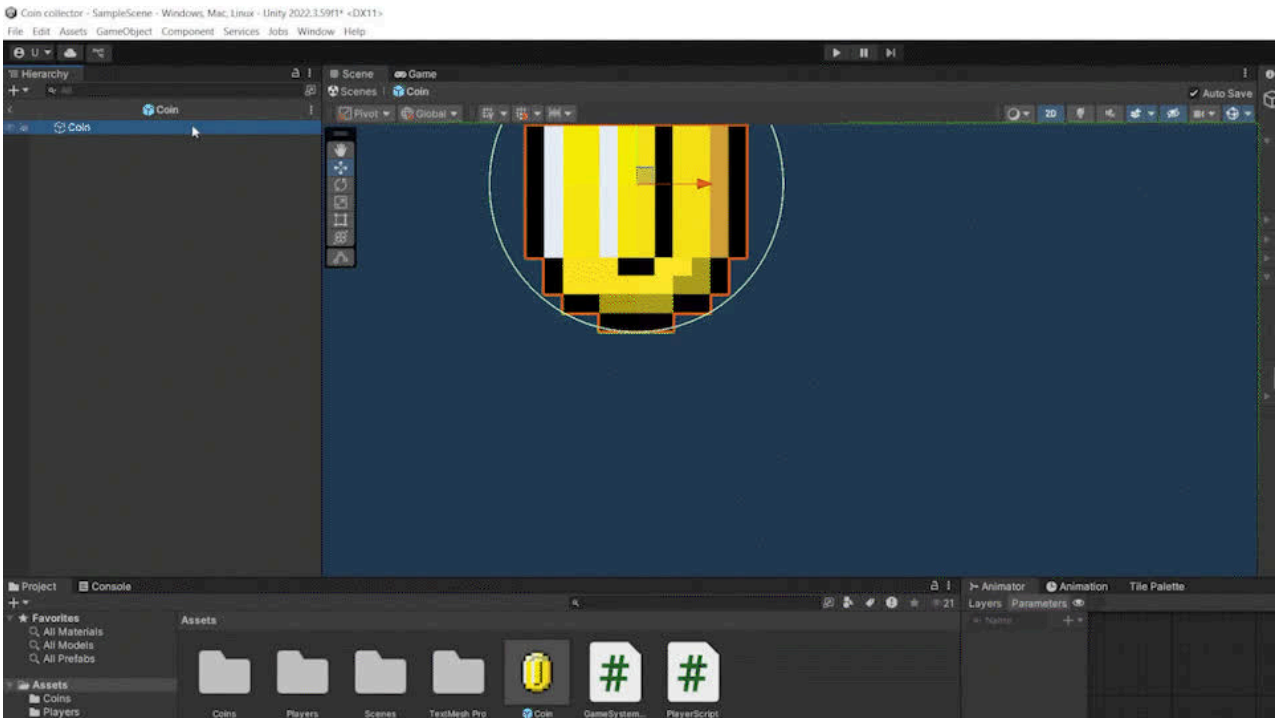
Vu que toutes les pièces sont supposément identiques il serait plus simple de toutes les changer d'un coup non ? C'est justement ce que permettent les Prefabs. Pour ce faire, doucle-cliquez sur le Prefab dans l'explorateur ou cliquez sur la flèche à droite du GameObject dans la hiérarchie.



Vous arriverez dans une scène spéciale pour modifier les Prefabs.

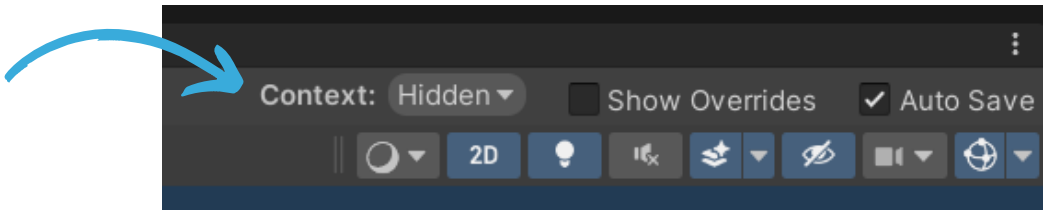


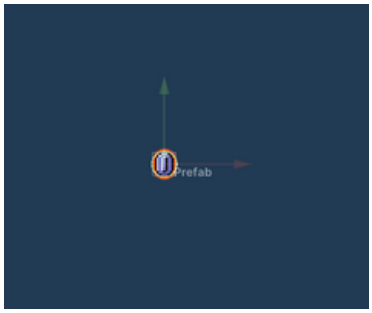
Il est possible que la scène ne soit pas centrée sur le Prefab que vous vous apprêtez à modifier, pour remédier à cela, double-cliquez sur le GameObject dans la hiérarchie.



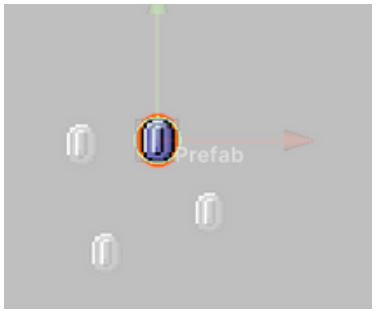
Si vous avez cliqué sur la flèche à droite il est possible que vous ne voyez pas exactement la même chose que dans l'exemple. Vous pourriez encore voir les autres éléments de la scène et vous les verrez parfois grisés.

Ceci est lié au paramètre "Context" en haut à droite de la scène. Le reste de la la scène (Context) sera soit caché (Hidden) soit grisé (Gray) soit visible (Normal).

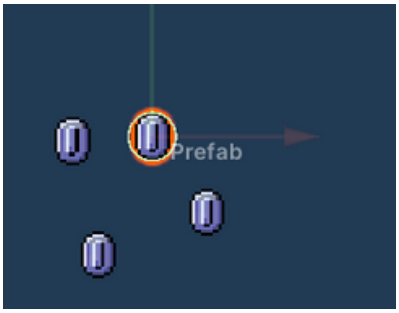




Hidden

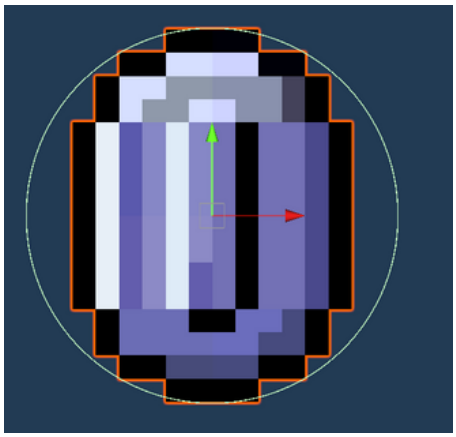


Gray

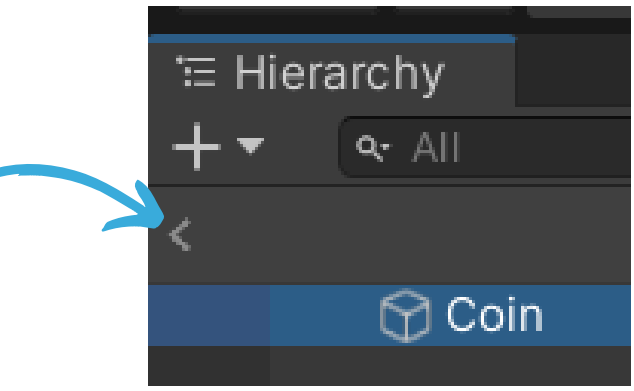


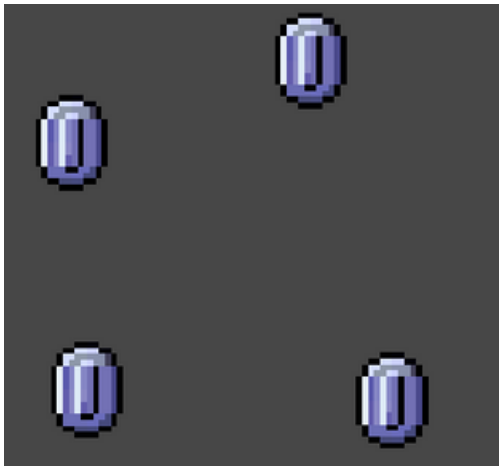
Visible

Dans cet exemple je modifie le Sprite du Prefab en changeant l'image.



Si je reviens dans la scène du jeu (en cliquant sur la flèche en haut à gauche de la hiérarchie) je peux voir que le changement de Sprite s'est répercuté sur tous les GameObjects créés sur base du Prefab.

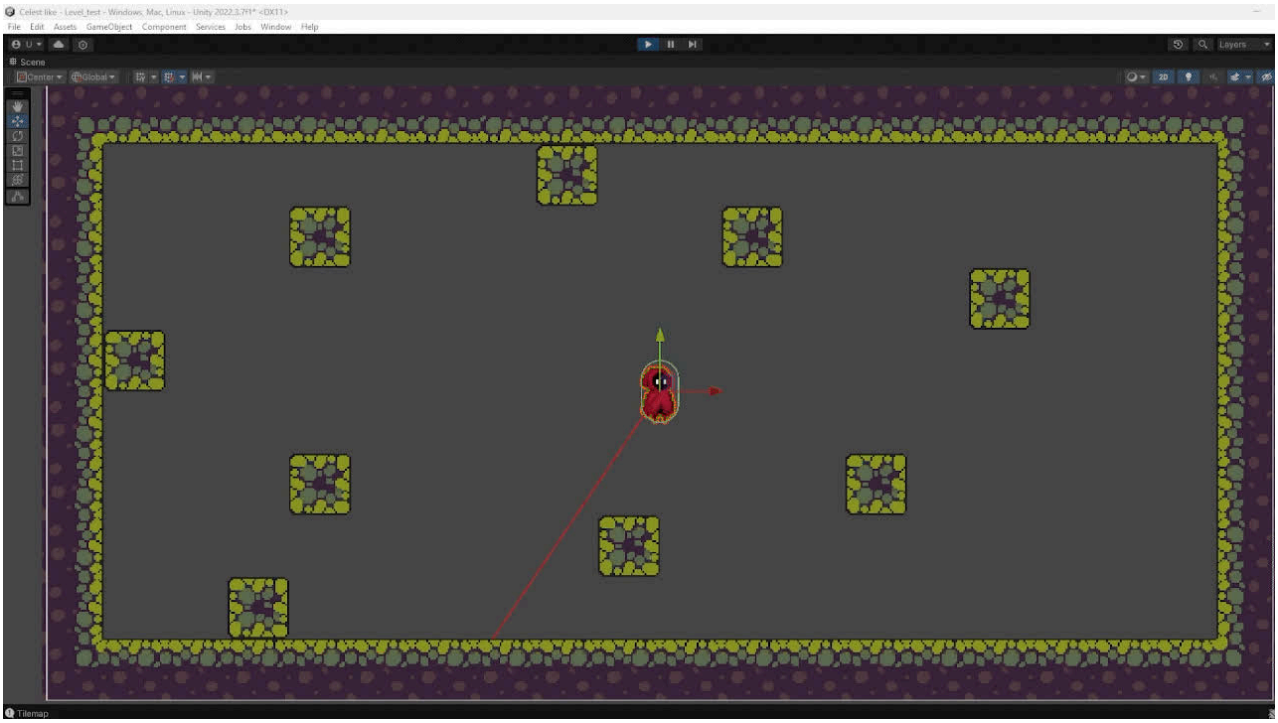




Dans cet exemple j'ai modifié le Sprite du Prefab mais vous pourriez changer n'importe quel élément du Prefab et la modification se répercutera sur tous les GameObjects créés à partir du Prefab.

RAYCAST

Les Raycasts (Tirer un rayon) sont un outils compris dans Unity que permet de “tirer un rayon laser” dans une direction depuis un point et de détecter quel objet le laser touche.



Les raycasts sont une série de fonctions qui prennent différents paramètres. Nous allons ici voir une d’entre elles mais il en existe d’autres. Leurs fonctionnements sont tous basés sur le même principe.

La version que nous allons voir est celle sous la forme “Physics2D.Raycast()” avec les paramètres suivants :

- Vector2 origin : c’est le point d’origine du rayon.
- Vector2 direction : c’est la direction vers laquelle le rayon ca être tiré.
- Float distance : c’est la distance maximale du rayon.
- LayerMask layerMask c’est l’ensemble des layers qui vont pouvoir être détectées par le rayon (voir fiche “Layers”. Attention, si vous tirez un rayon depuis un GameObject qui comporte un Collider. Le Collider ne doit pas être sur une des Layers du LayerMask.

La fonction renvoie un RaycastHit2D (touche d’envoie de rayon) qui contient des informations sur l’objet touché, le point auquel il a touché, l’angle, etc.

Quand vous utilisez la fonction Raycast, un rayon est tiré du point d'origine dans la direction donnée. Si le rayon touche un Collider qui se trouve sur une des couches du LayerMask avant d'arriver à la distance alors elle renvoie un Hit.

```
RaycastHit2D hit = Physics2D.Raycast(transform.position, Vector3.down, 1.5f, levelMask);
```

Le hit va ensuite pouvoir être utilisé dans une condition. Il équivaudra à un **true** si le rayon a touché quelque chose.

```
if (Input.GetKeyDown(KeyCode.UpArrow) && hit)
{
    myRigidBody.velocity += Vector2.up * jumpPower;
}
```

Dans l'exemple ci-dessus on a un bout de code pour permettre de sauter quand on appuie sur la flèche du haut. Quand on appuie et que le rayon a touché alors la vitesse est augmentée d'un vecteur vers le haut.

Le Raycast permet de s'assurer qu'il y a bien une plateforme sous le personnage.

Comme dit plus haut, le Hit contient des informations sur l'impact du Raycast. Il permet entre autres de connaitre :

- Le collider qu'il a touché (collider)
- Le point d'impact (point)
- La distance parcouru par le rayon lors de l'impact (distance)

On peut accéder à ces informations avec le Hit suivi d'un point puis du nom de la propriété à laquelle on veut accéder. Dans la liste au-dessus les mots entre parenthèses sont les noms des propriétés.

```
print(hit.distance);
```

Ceci afficher la distance à l'impact

RIGIDBODY 2D

Le RigidBody prend en charge tout un tas de comportements propres à un corps qui respecte les lois de la physique (gravité, frottement, etc).

Nous allons nous intéresser à quelque uns des paramètres d'un RigidBody pour voir ce qu'ils font et comment s'en servir.

Pour commencer nous avons le **BodyType.** Il peut être de plusieurs types (Cinematic, Kinematic et Static).
Un **Cinematic** est un corps qui va être complètement simulé par le moteur physique du moteur de jeu.
Un **Kinematic** est un corps particulier qui sert quand on fait de l'animation un peu poussée (nous ne l'utiliserons pas).
Un **Static** est un corps qui ne sera pas simulé par le moteur physique mais qui pourra interagir avec les RigidBody d'autres GameObjects (il sert aux éléments de l'environnement qui ne bougent pas).

Le **Material** est un objet qui va définir des propriétés physiques pour le matériau. En 2D il ne va définir que le coefficient de friction. Changer sa valeur va permette d'augmenter ou diminuer les frottements entre deux objets.

La **Mass** correspond à la masse dans les interactions entre deux corps. Si un corps avec un fable masse entre en collision avec un corps avec une grande masse, la vitesse du corps massif va peu changer tandis que celle du corps peu massif va fort changer.

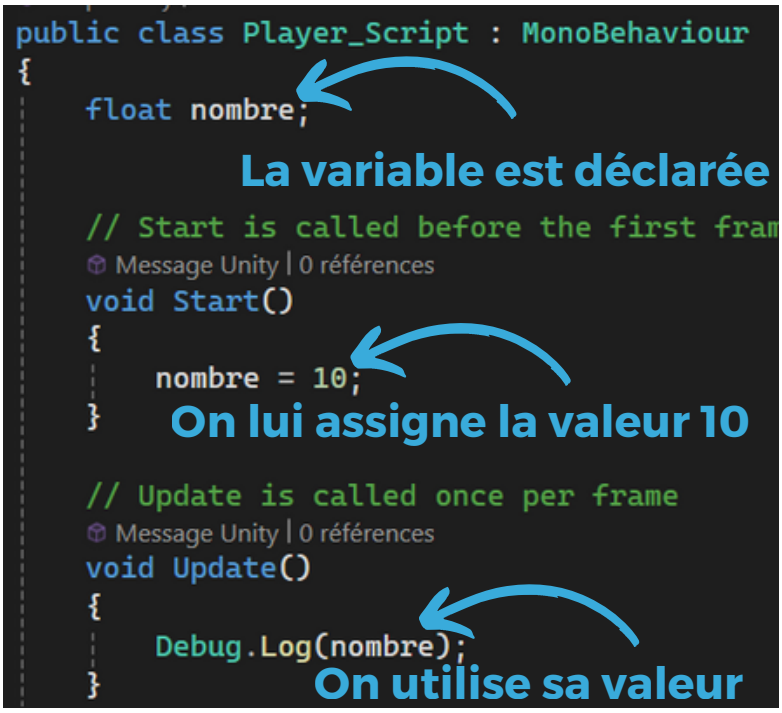
La **Gravity Scale** correspond à un facteur qui va influencer l'effet de la gravité sur l'objet. Plus il est grand plus l'effet de la gravité sera grand.

Les **Constaints** vont permettre de contraindre les mouvements du corps. Freeze position X ou Y vont bloquer les mouvements sur un axe donné. Si vous cochez Freeze Y, le GameObject ne pourra pas se déplacer à la verticale. Si vous cochez Freeze Rotation Z le corps ne pourra plus tourner autour de l'axe Z.

SCOPE (PARTIE 1)

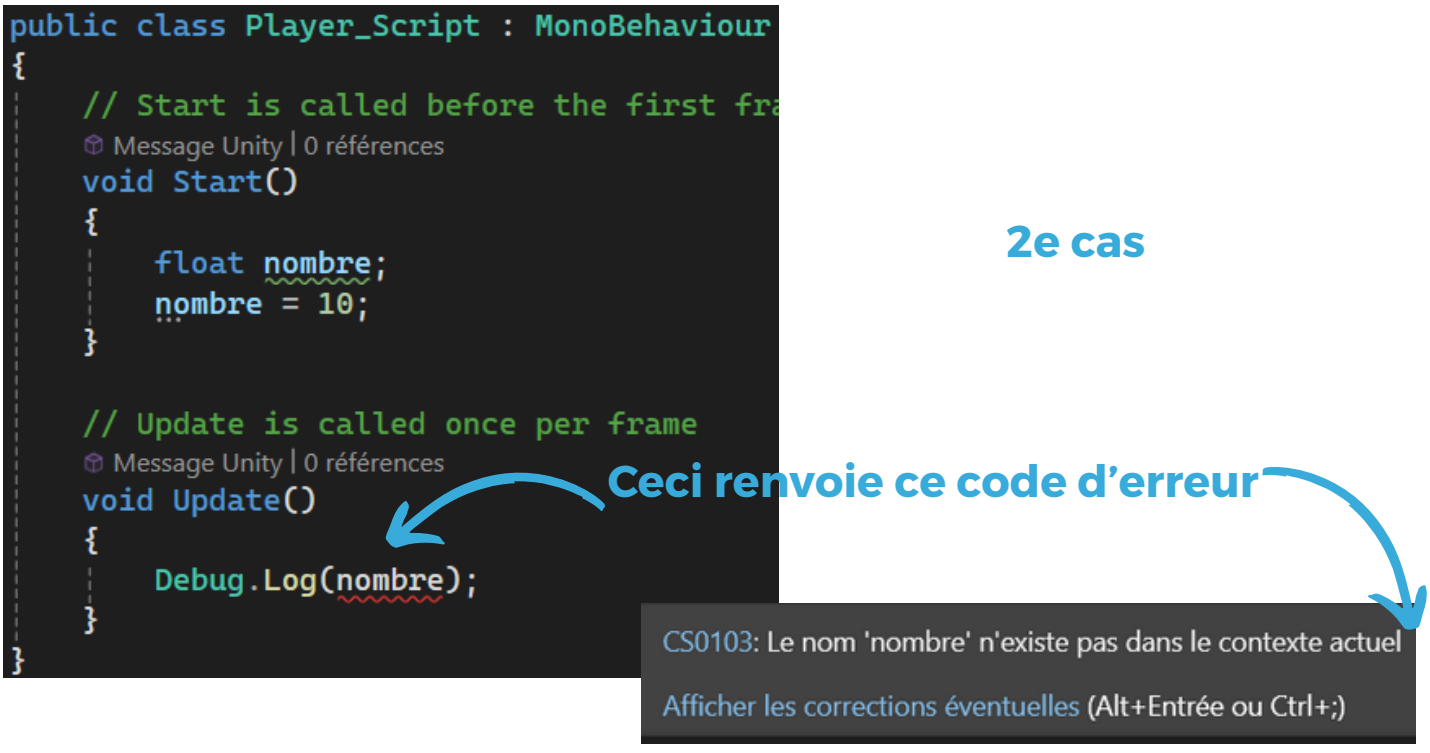
Nous allons voir ici une notion un peu obscure mais qui risque de vous poser problème tôt ou tard. Lorsque vous créez une variable, vous pourrez accéder à sa valeur plus tard dans votre programme. Mais vous pourriez tomber sur des situations dans lesquelles vous n'arrivez pas à accéder à une variable que vous aurez préalablement créée.

Quand vous créez une variable dans une class d'un script elle sera généralement accessible dans l'entièreté du script.



1er cas

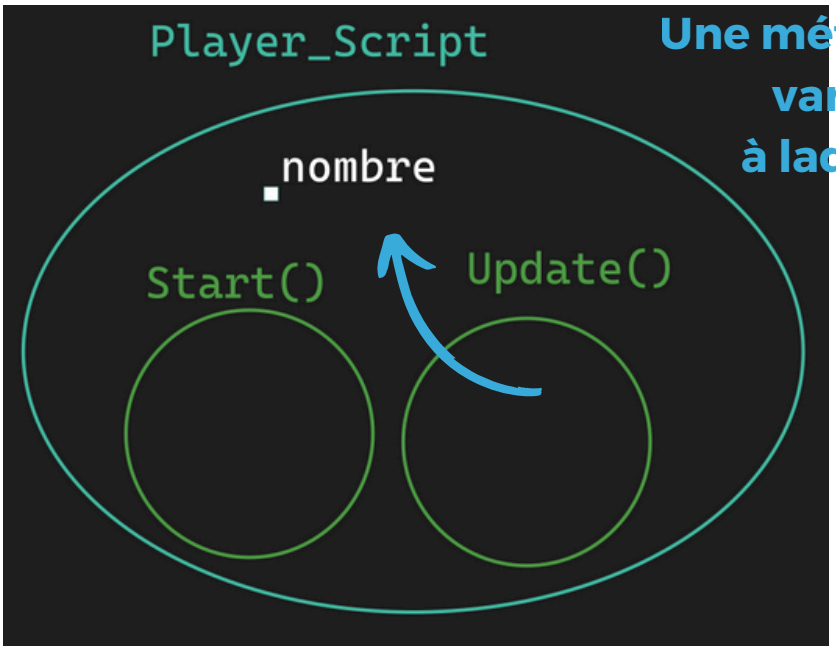
En revanche, si vous déclarez une variable dans une méthode (start, update ou autre) elle ne sera accessible que dans cette méthode.



2e cas

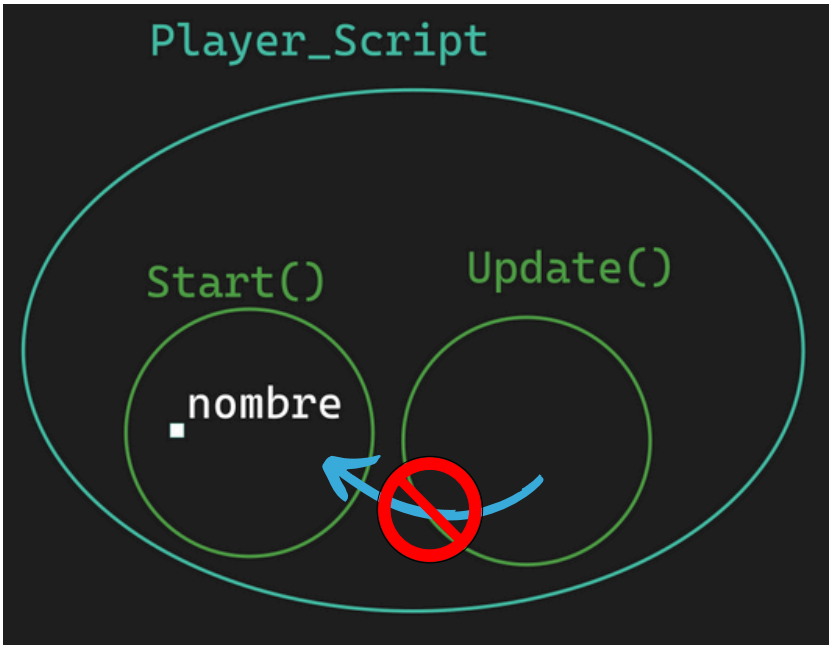
L'erreur nous indique que la variable n'existe pas dans le contexte actuel. Ceci s'explique par le fait que la variable a été déclarée dans la méthode start. Elle n'est donc accessible que dans cette même méthode.

La "zone" dans laquelle une variable est accessible est appelée son scope (prononcez "scaupe"). Il peut être visualisé comme les ensembles en mathématique :



Une méthode peut accéder aux variables de la classe à laquelle elle fait partie

1er cas



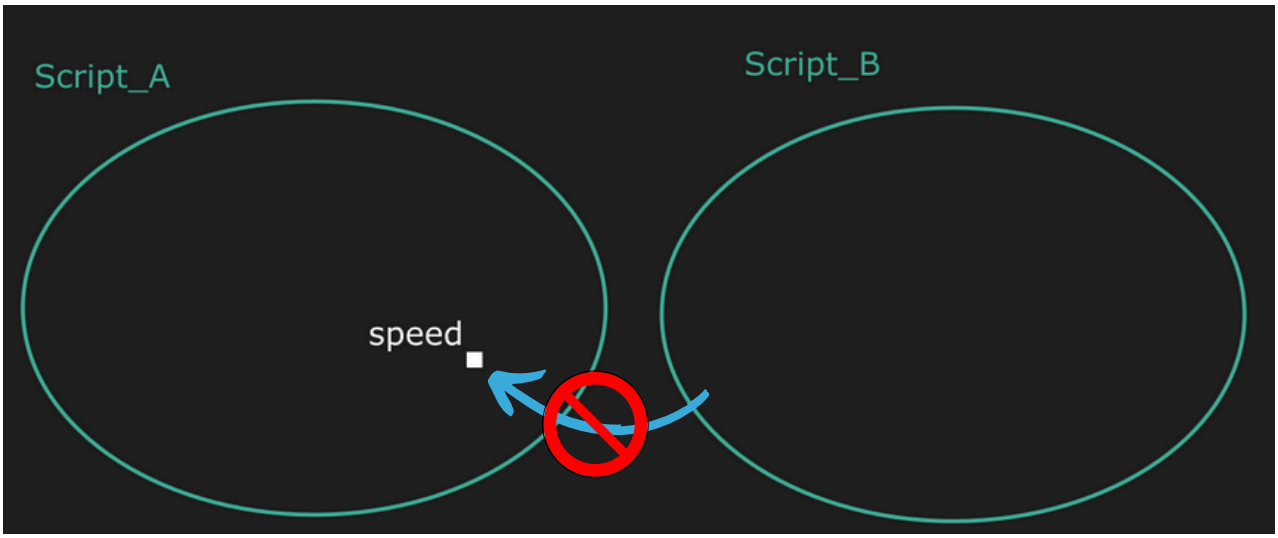
2e cas

Une méthode ne peut pas accéder à une variables d'une autre méthode

SCOPE (PARTIE 2)

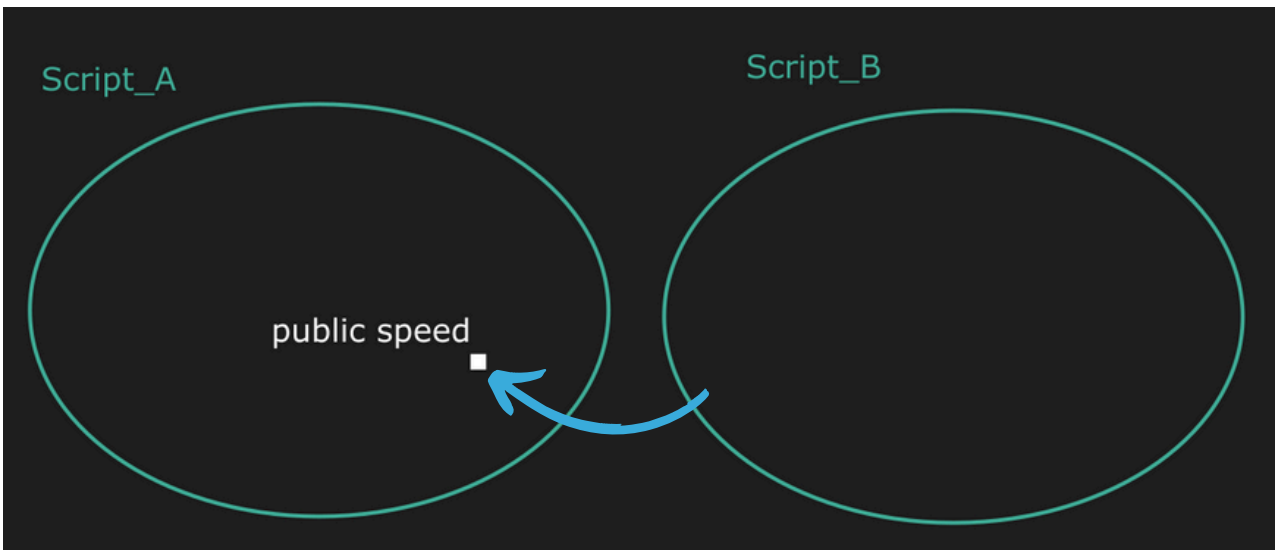
Nous allons maintenant aller plus loin avec cette idée de scopes.

Les variables créées dans un script ne sont, de base, accessibles qu'au sein de ce script. Un script A ne peut accéder à une variable contenue dans un script B. ceci est vrai pour une variable de base, mais il est possible de rendre une variable accessible à l'extérieur d'un script.



Pour cela, rien de plus simple, il suffit de rajouter le mot-clé "public" devant la déclaration de la variable.

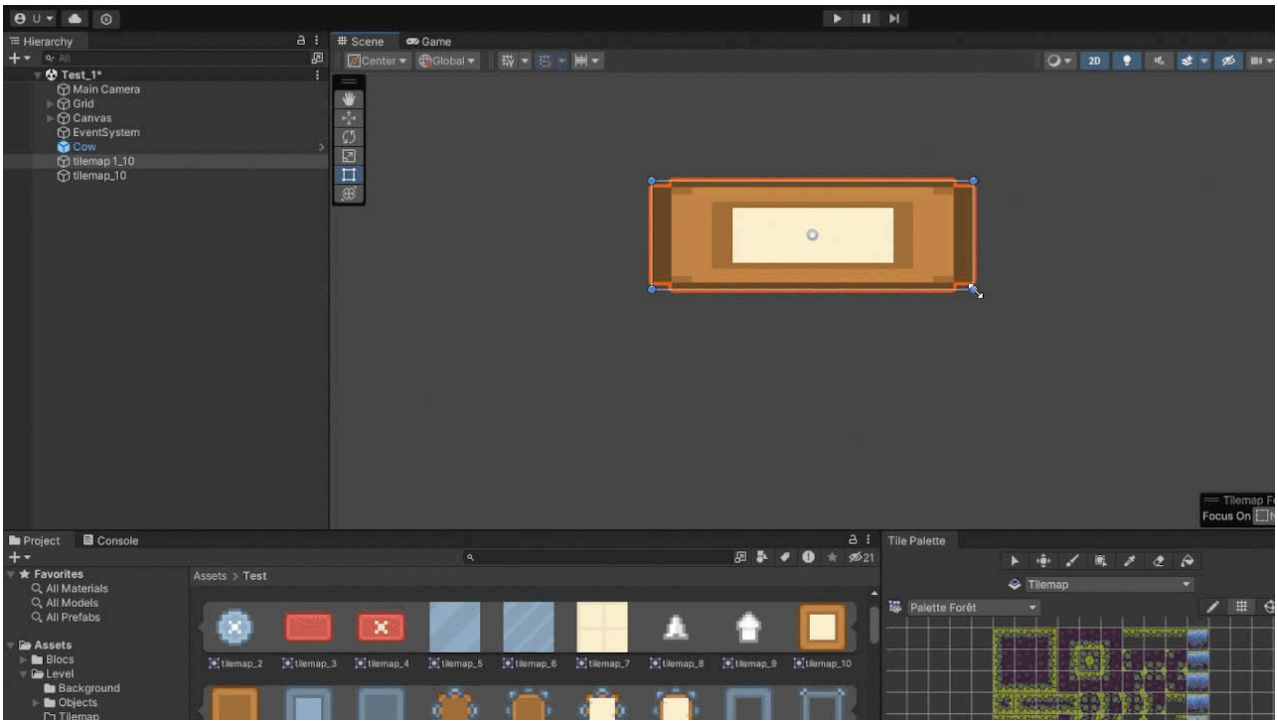
```
public float speed;
```



A noter que ces notions de scope s'appliquent aussi aux fonctions créées dans les classes.

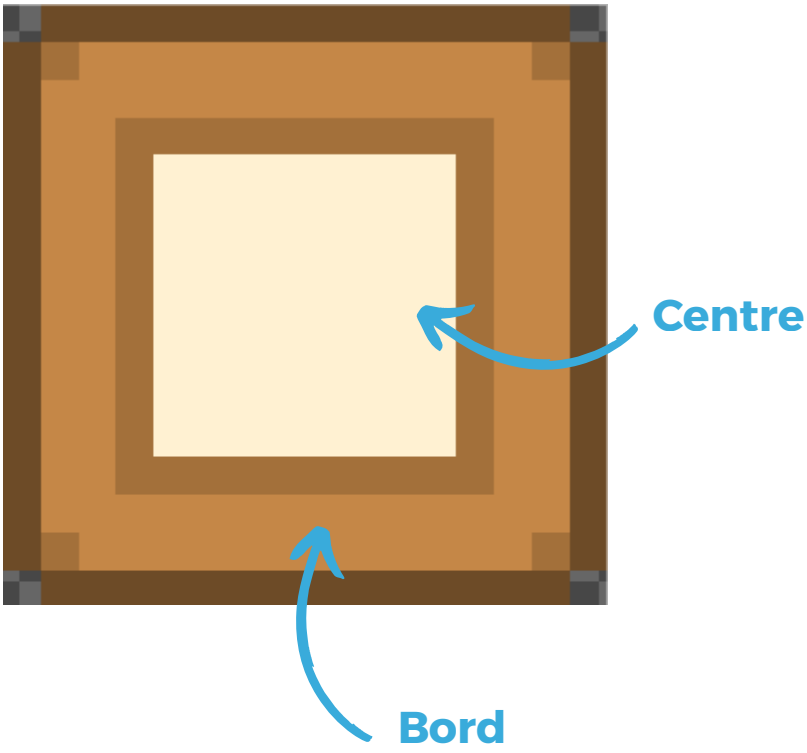
SLICED SPRITE

Quand vous changez les dimensions d'un Sprite, Unity va étirer l'image source pour occuper l'espace demandé. Il va souvent pour cela déformer l'image, dégradant son aspect.

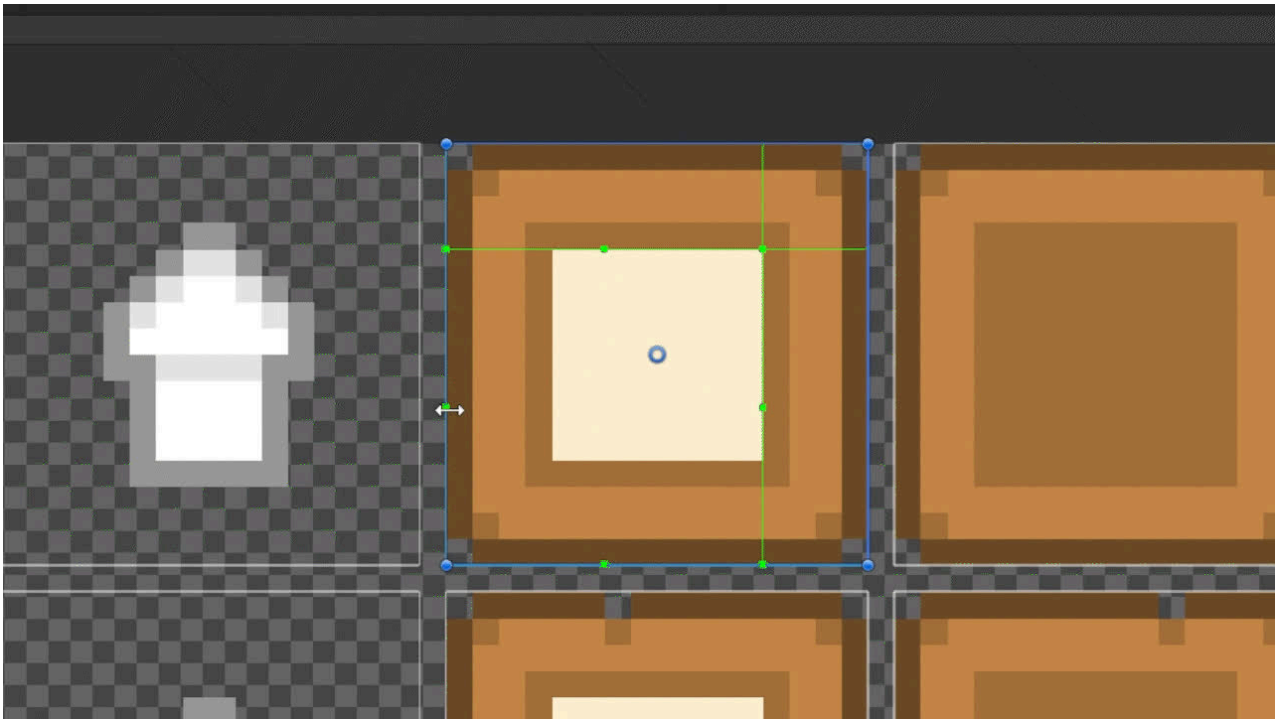


Il peut y avoir des éléments pour lesquels il est utile de pouvoir définir leurs dimensions sans déformer l'image. Il existe un outil pour cela : les Slices Sprites (sprites découpés).

Pour les utiliser il vous faut une image d'un Sprite avec des bords et un centre.

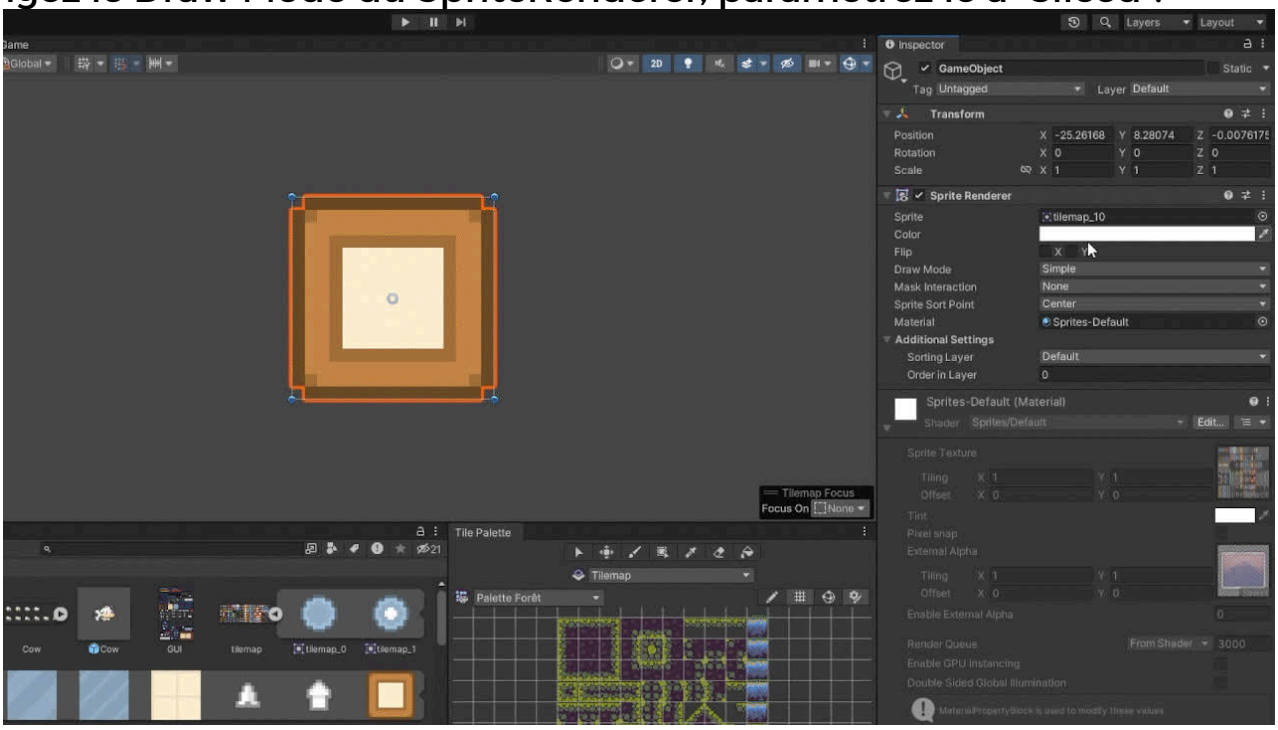


Dans le Sprite Editor, cliquez-glissez les petits carrés verts pour séparer les bords du centre.



Terminez en cliquant sur “Apply”.

Une fois le Sprite correctement découpé, assignez le à un SpriteRenderer ou une Image (dans le Canvas.
Changez le Draw Mode du SpriteRenderer, paramétrez le à “Sliced”.

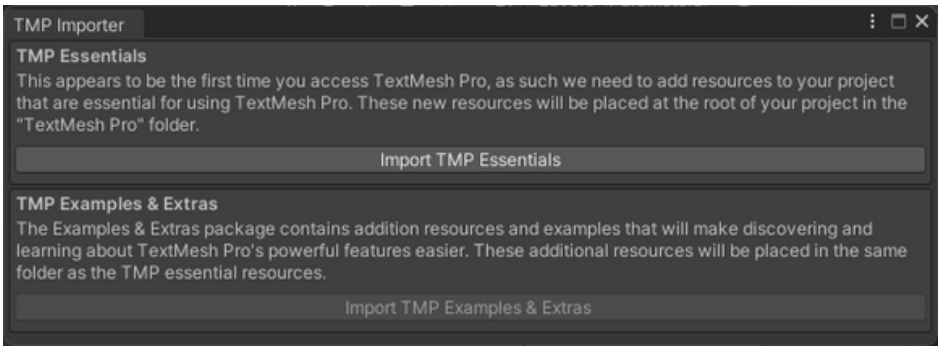


TEXTMESHPRO

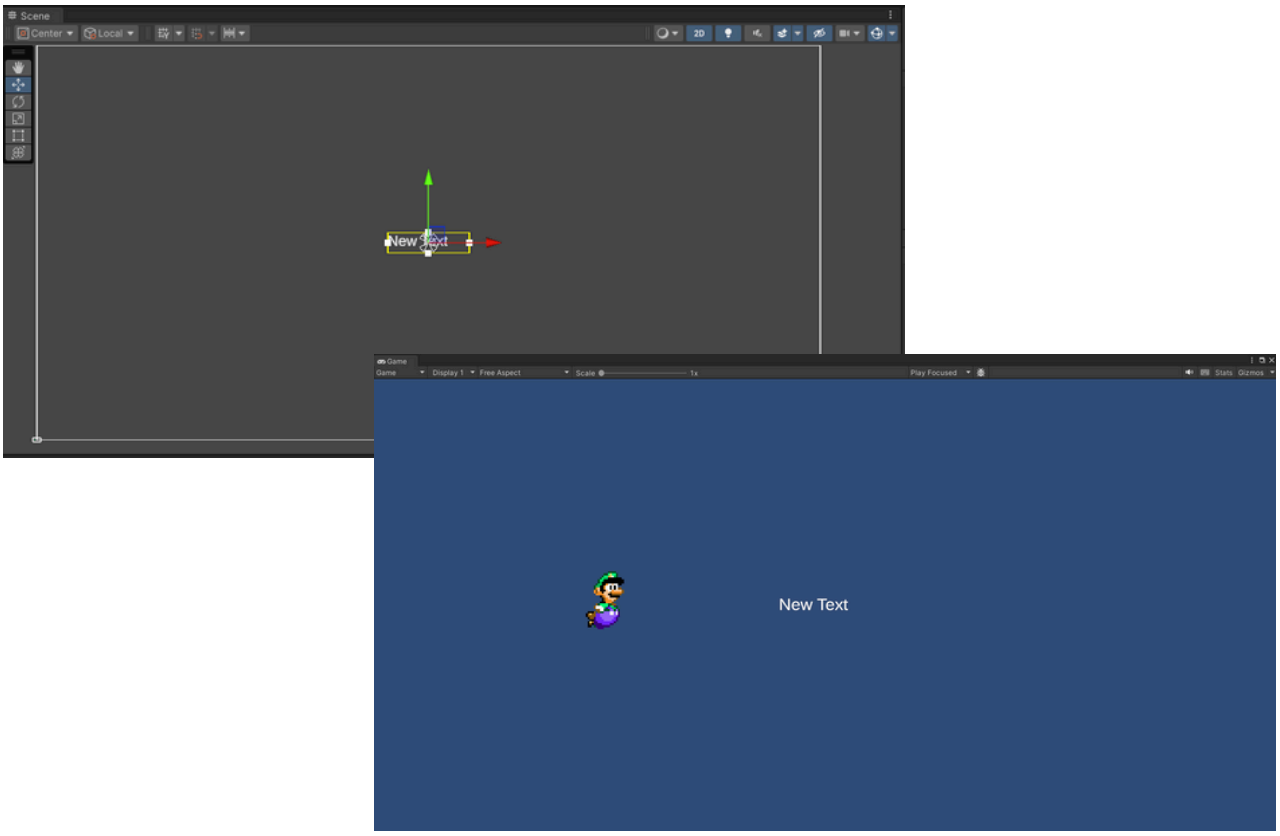
Si vous voulez ajouter du texte dans l'interface de votre jeu (UI) vous aurez besoin d'utiliser les TextMeshPro.

Ces derniers sont des GameObjects qui viennent se placer comme enfants (voir "Children - Parents") d'un Canvas.
Pour créer votre premier TextMeshPro faites clic droit dans une zone libre de la hiérarchie. Placez ensuite votre curseur sur "UI" puis cliquez sur "Text - TextMeshPro"

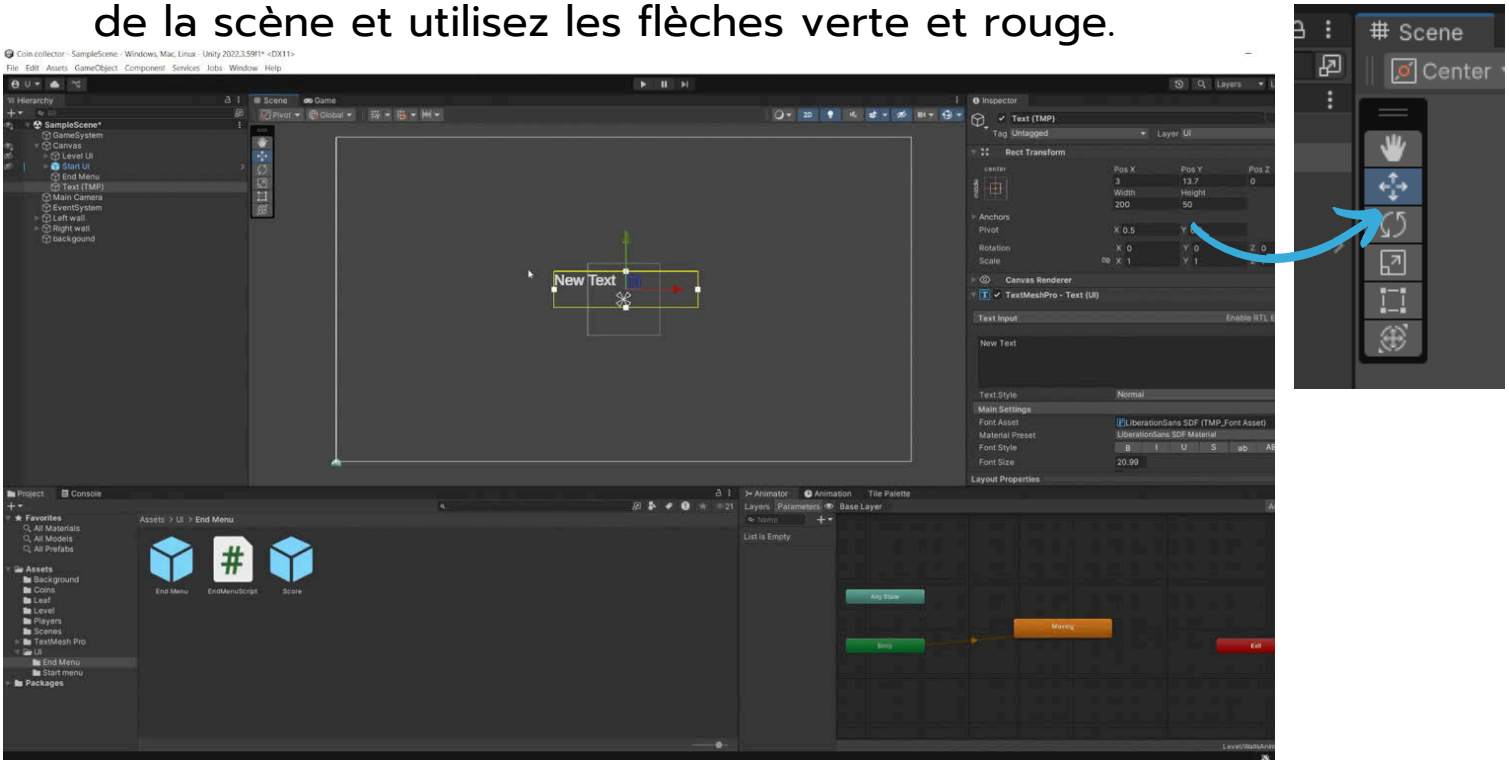
La première fois que vous créez un TextMeshPro vous devrez importer un module, pour ce faire, cliquez simplement sur Import TMP Essentials.



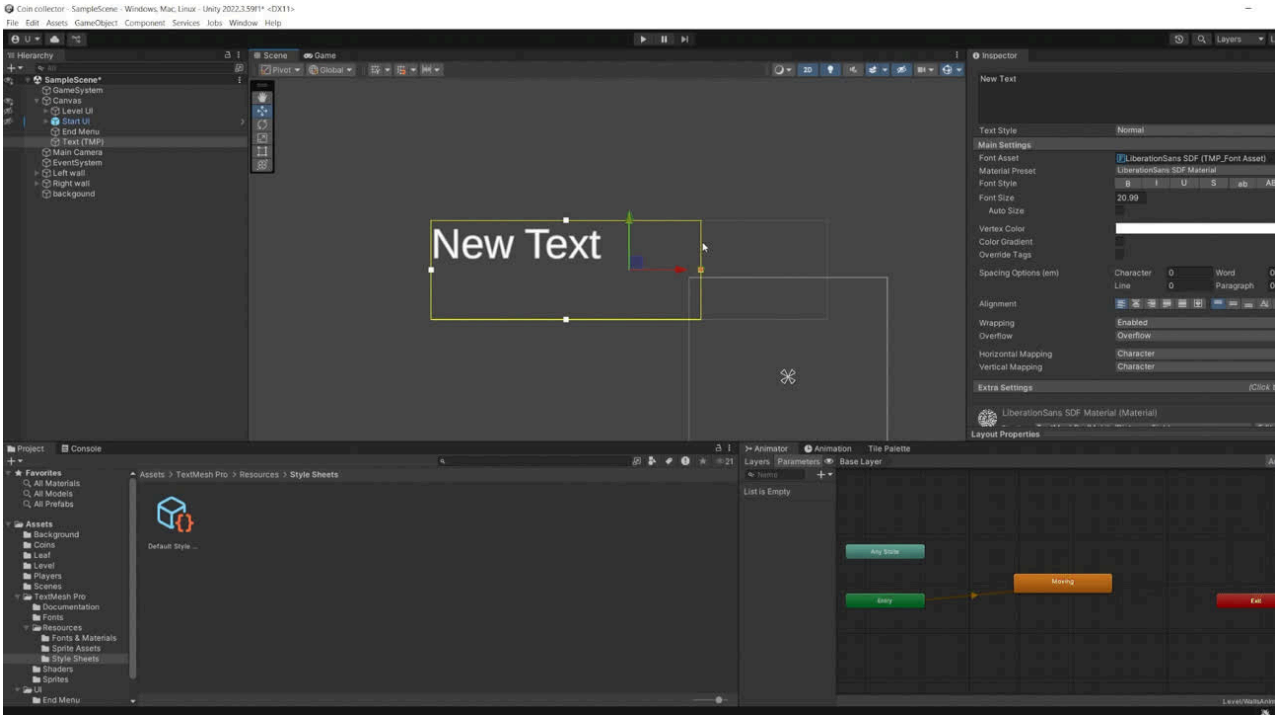
Une fois le TextMeshPro créé vous pourrez le voir dans le Canvas et dans le jeu dans la fenêtre Game.



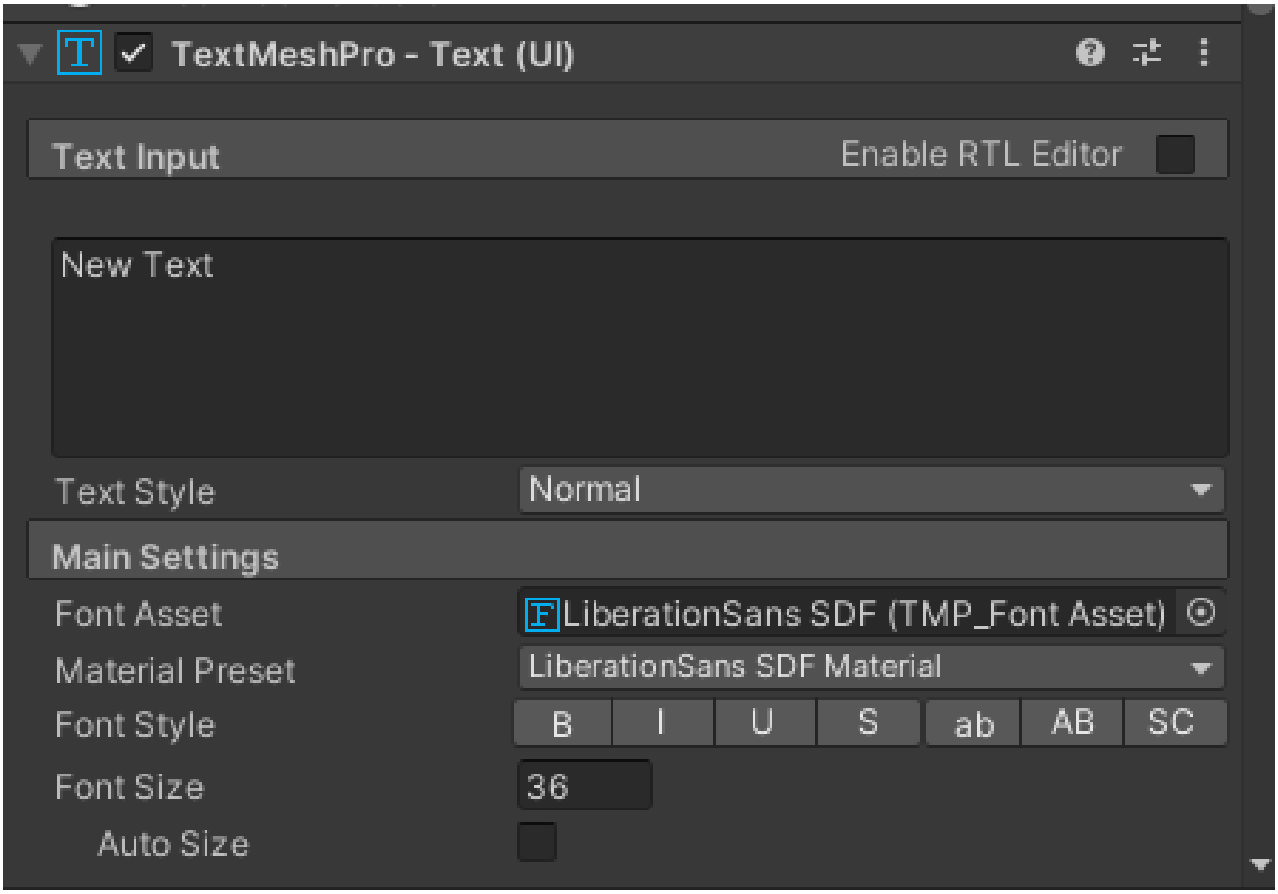
Vous pouvez déplacer le TextMeshPro dans le Canvas pour changer sa position dans l'écran du jeu. Pour ce faire, sélectionnez-le, cliquez sur l'outil "Move" en haut à gauche de la scène et utilisez les flèches verte et rouge.



Le rectangle jaune que vous voyez est la zone dans laquelle le texte sera écrit. Vous pouvez changer ses dimensions en cliquant (maintenir) et glissant les petits carrés blancs.



Le TextMeshPro comporte un composant du même nom qui a de nombreuses propriétés. Nous allons les voir ici.



La zone de texte va pouvoir définir le texte qui sera affiché.

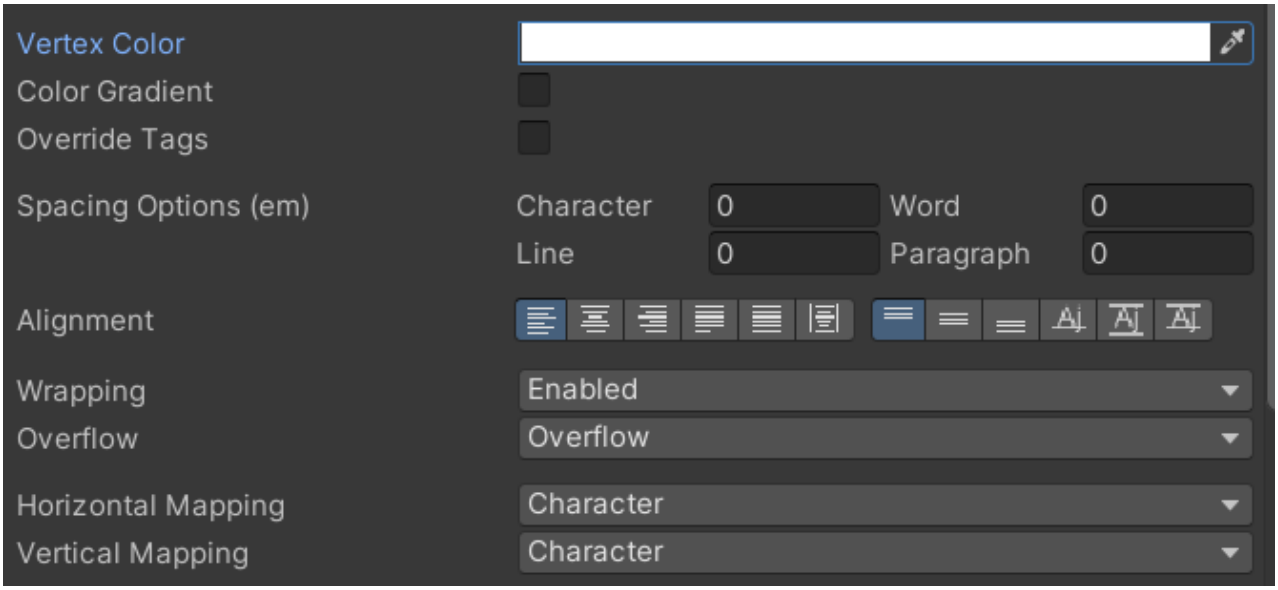
Le Text Style permet de définir des styles que l'on va appliquer au texte et qui vont utiliser des paramètres prédéfinis pour toutes les propriétés. Ils sont compliqués à utiliser, je vous conseil de ne pas les utiliser maintenant.

Le Font Asset va définir quel asset de police (Font) va être utilisé pour le texte (voir "Importer des polices").

Le Material Preset permet de rajouter des effets aux polices mais est aussi une fonctionnalité plus avancée.

Les Font Style vont permettre d'appliquer différentes modifications au texte (gras, italique, souligné, barré, minuscule, majuscule, etc).

La Font Size va définir la taille de la police.



Le Vertex Color permet de modifier la couleur de la police.

Le Color Gradient permet de mettre un gradient de couleur dans la police (nous en verrons pas comment l'utiliser).

Ignorez Override Tags, nous en verrons pas comment l'utiliser.

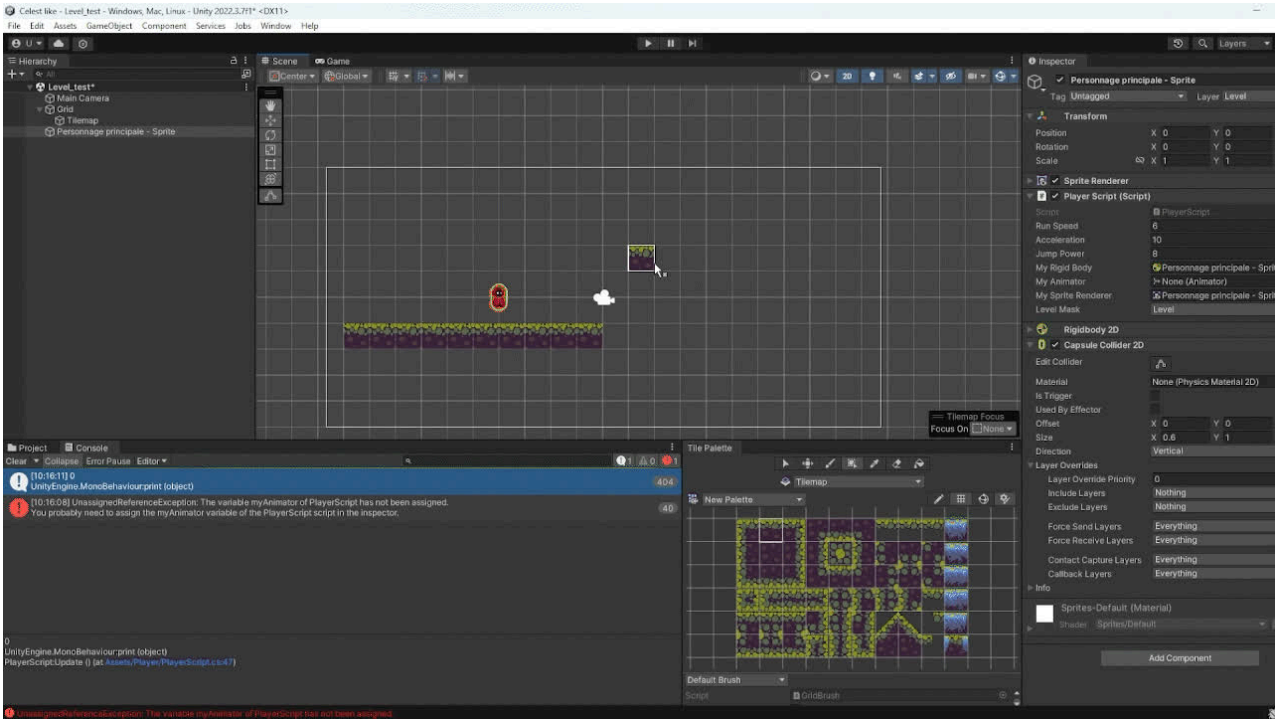
Les Spacing Options permettent de changer l'espacement entre les lettres (Character), les mots (Word), les lignes (Line) et les paragraphes (Paragraph).

Les Alignement permettent de changer la manière dont le texte sera aligné dans la zone jaune.

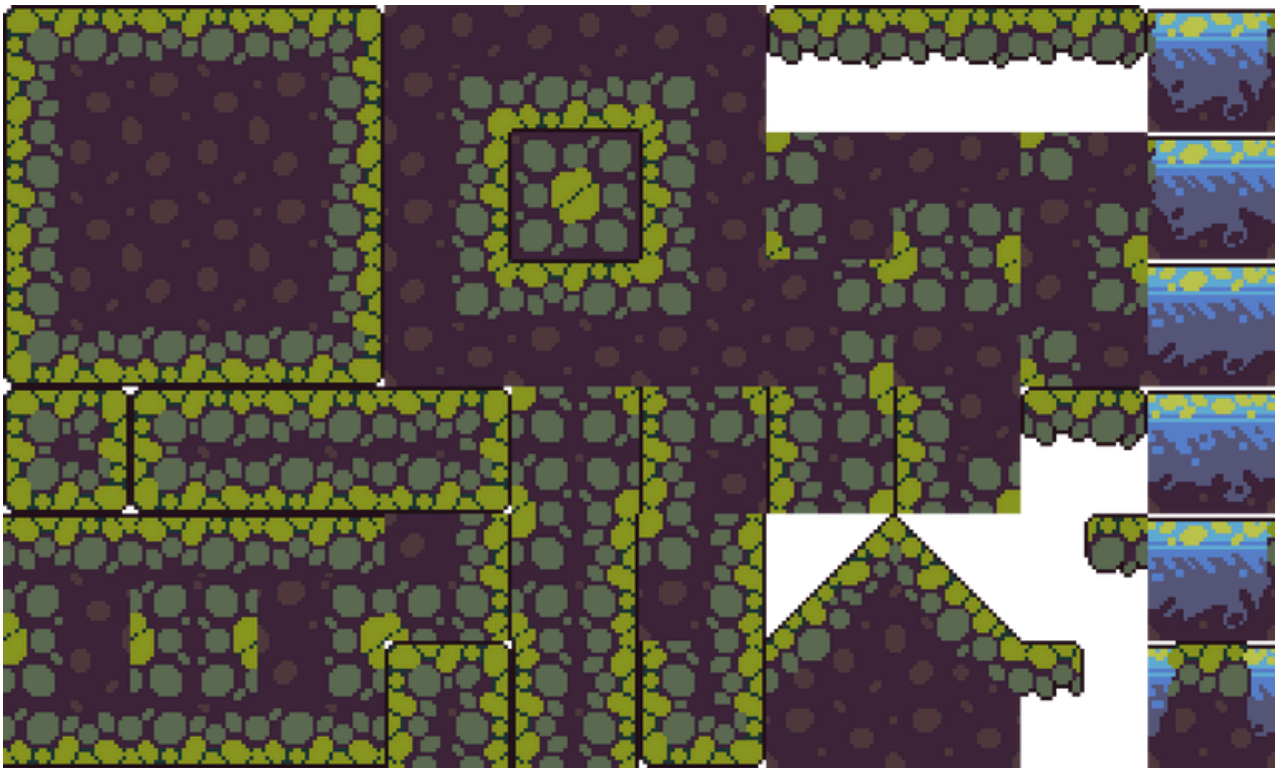
Ignorez les paramètres suivants, nous en verrons pas comment l'utiliser.

TILEMAP

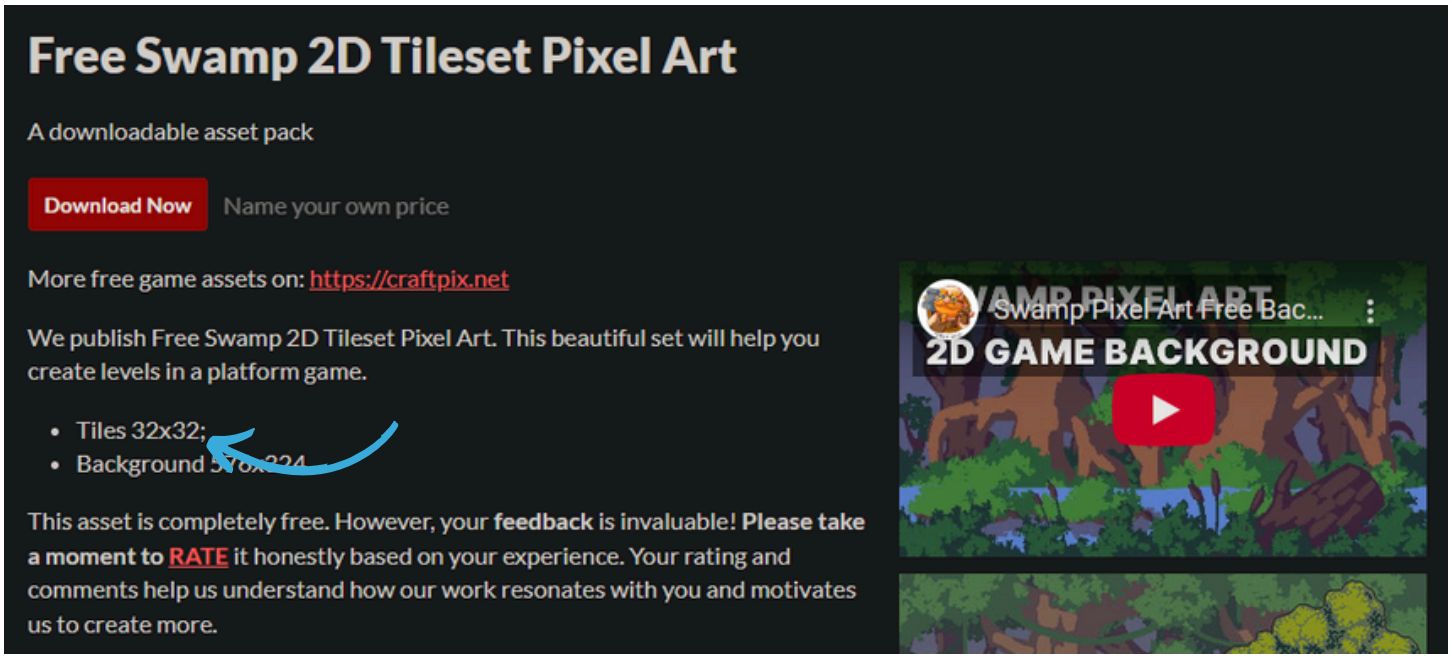
Les Tilemaps (cartes de tuiles) sont des outils qui permettent de “peindre” le décor d’un niveau plutôt que de déposer ses blocs un par un.



Pour utiliser une Tilemap vous devrez avoir un Tileset (jeu de tuiles), c’est à dire une image ou un ensemble d’images des Sprites des tuiles.

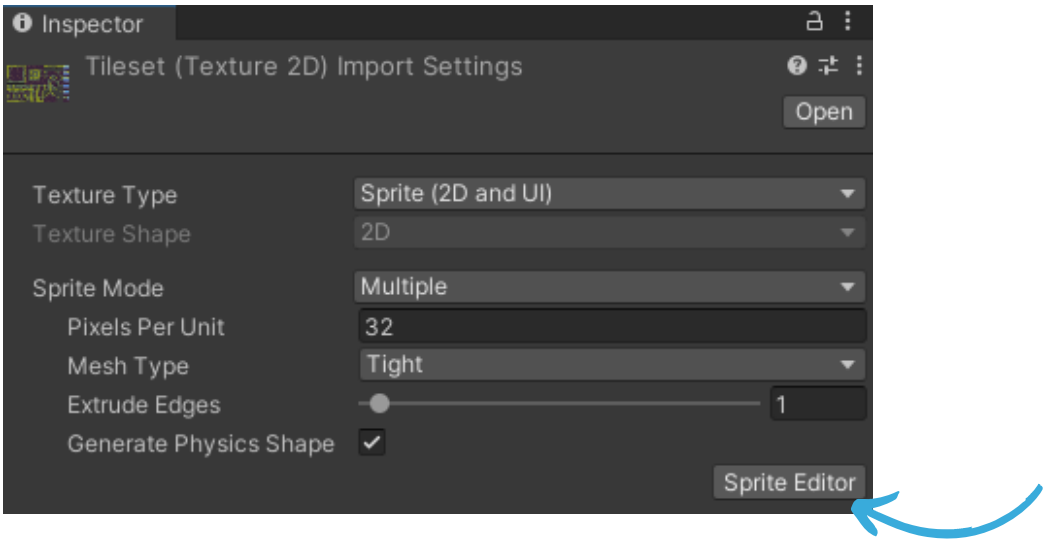


Une fois votre image de Tileset importée il faut correctement paramétrer les pixel per unit. Une unité de distance doit correspondre à une Tile du Tileset. Chaque Tileset a des dimensions qui sont généralement indiquées sur les pages où vous trouverez les assets.

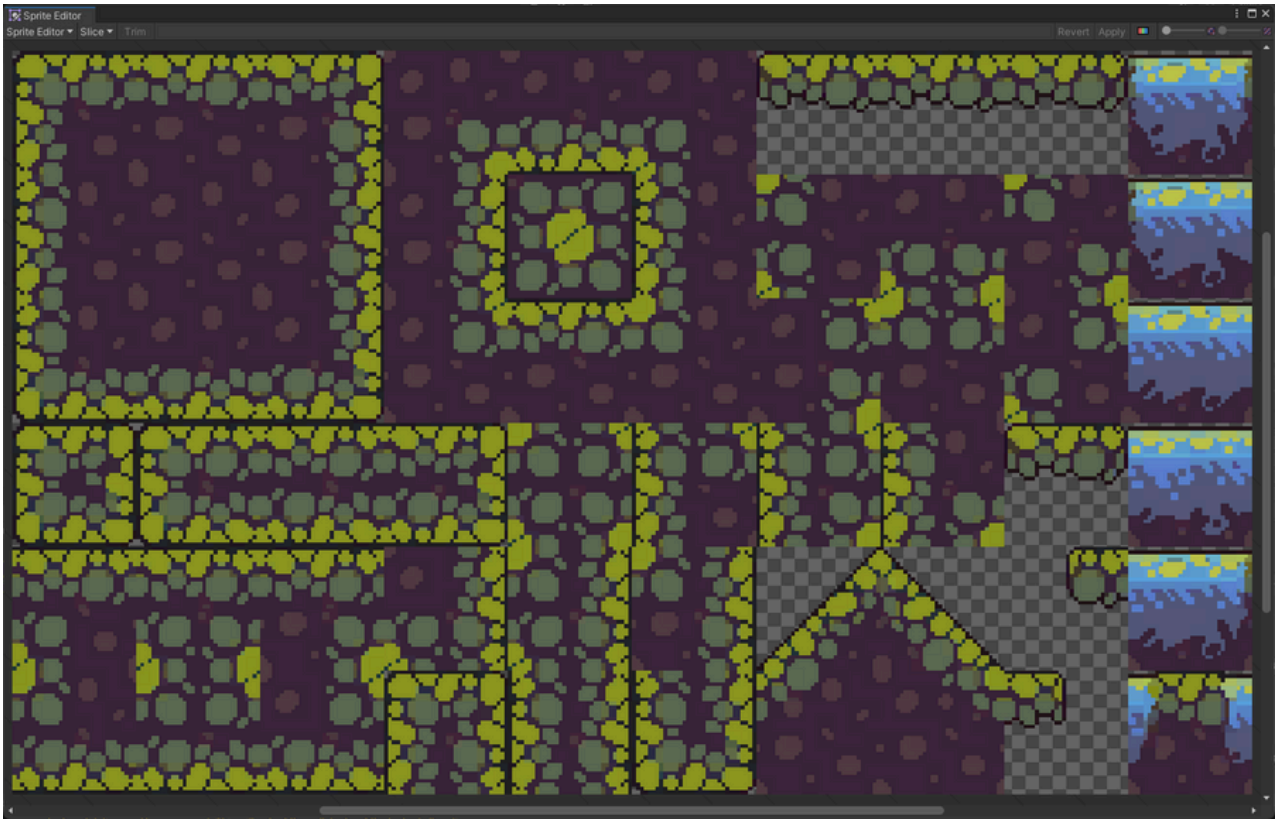


Dans mon exemple les Tiles font donc 32 pixels su 32. Les pixels per unit doivent donc être mis à 32.

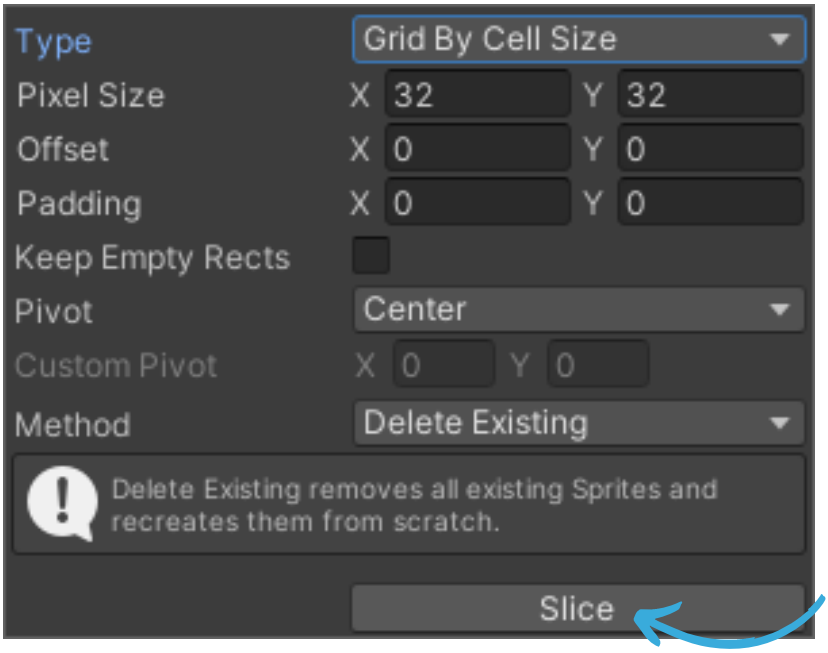
Vous devrez ensuite découper l'image en Sprites. Pour ce faire, sélectionnez là et cliquez sur "Sprite Editor"



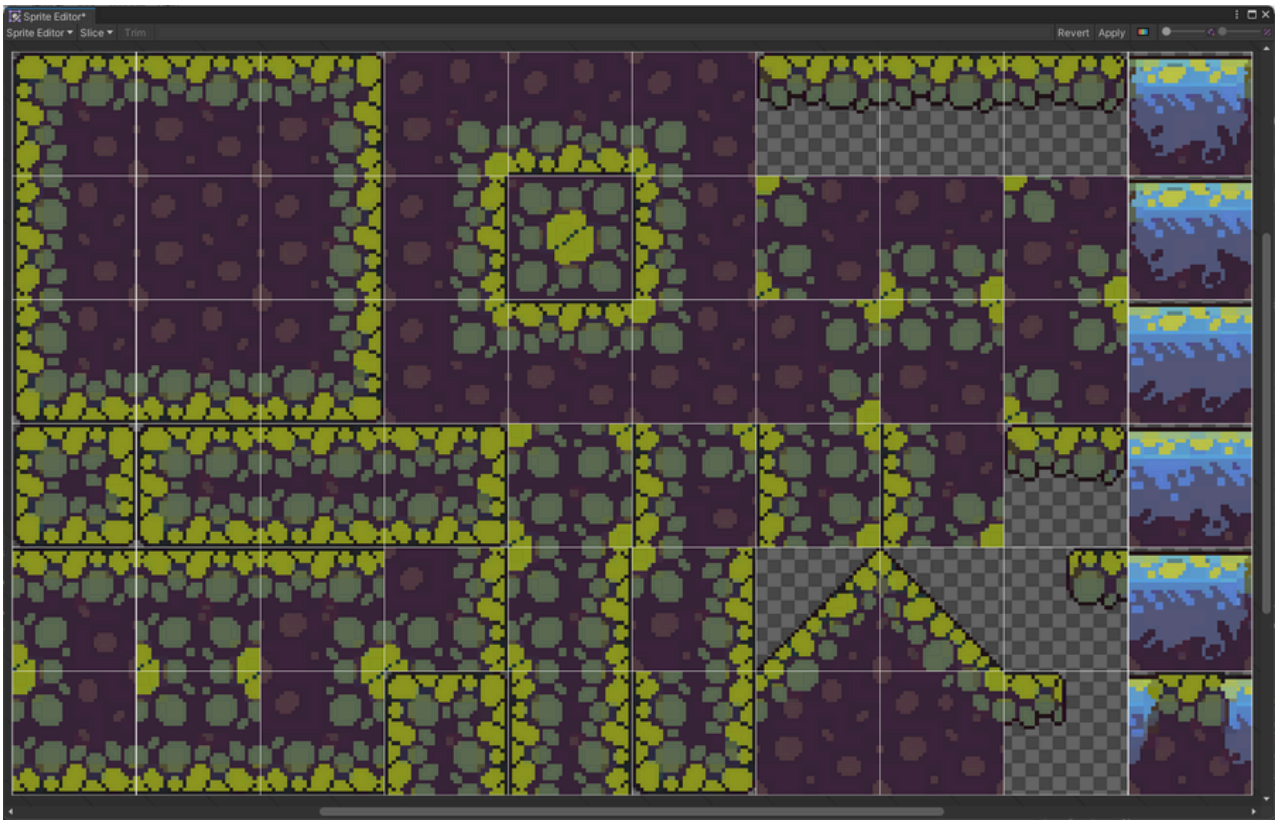
Une nouvelle fenêtre va s'ouvrir.



Cette fenêtre va vous permettre de découper l'image en plusieurs Sprites.
Cliquez sur "Slice en haut à gauche. Un menu s'ouvrira avec différentes options.
Pour le Type sélectionnez "Grid By Cell Size".
Pour les deux Pixel Size mettez la taille des Tiles (dans mon exemple 32).
Finissez en cliquant sur "Slice.

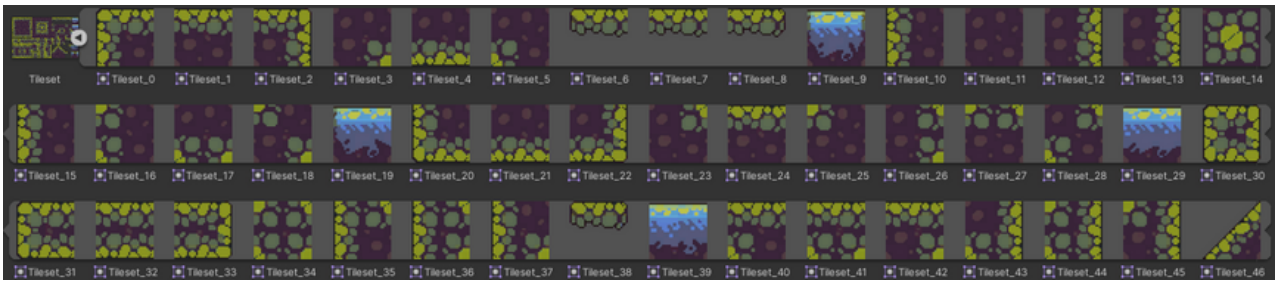


Si vous revenez sur le Sprite Editor vous verrez alors les Sprites découpés par des lignes blanches.



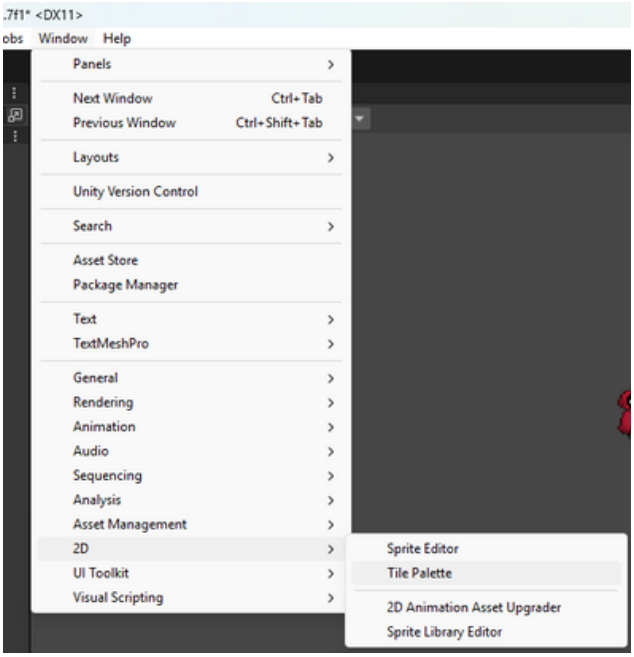
Finissez en cliquant sur “Apply” en haut à droite et fermez le Sprite Editor.

Si vous cliquez sur la flèche à droite de l’image vous verrez alors tous les Sprites créés.

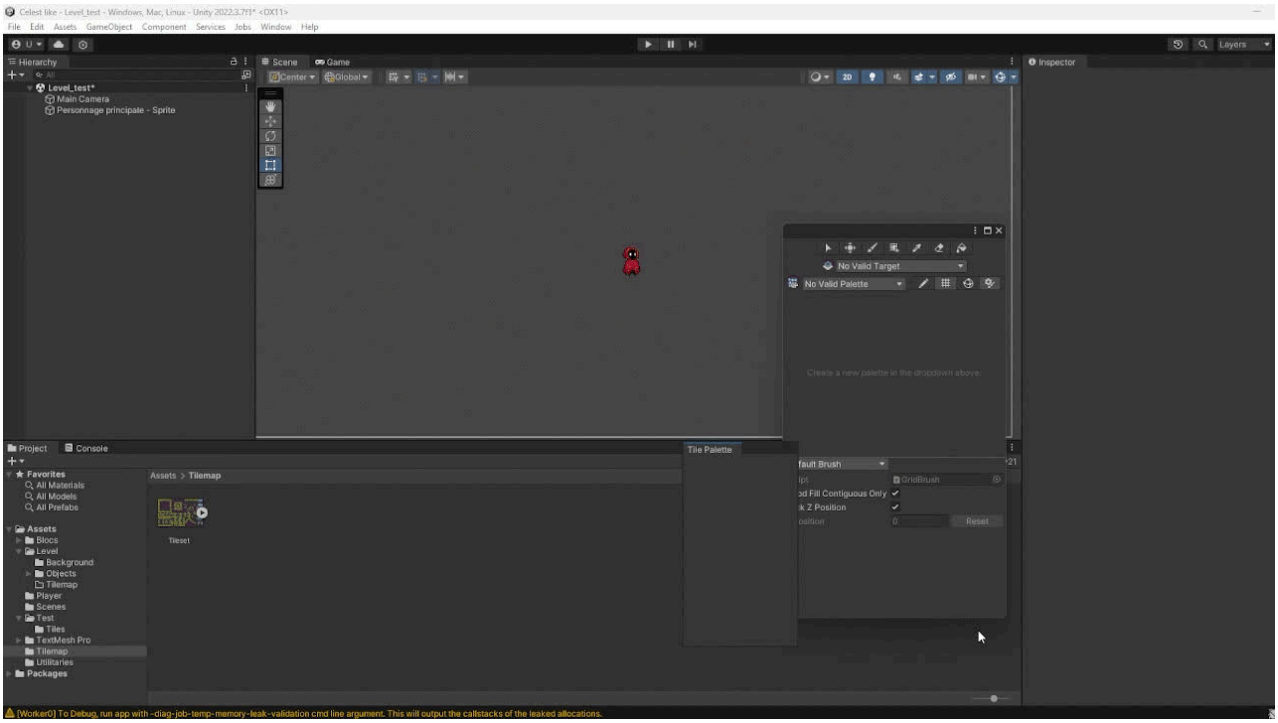


Il nous faut maintenant créer une Tile Palette, c'est un outil qui va contenir les Tiles et nous permettre de les sélectionner pour les peindre.

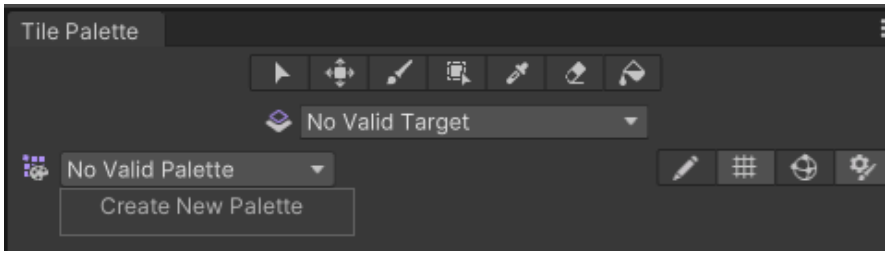
Commencez par cliquer sur "Windows" en haut à gauche de l'éditeur Unity, puis 2D et enfin Tile Palette.



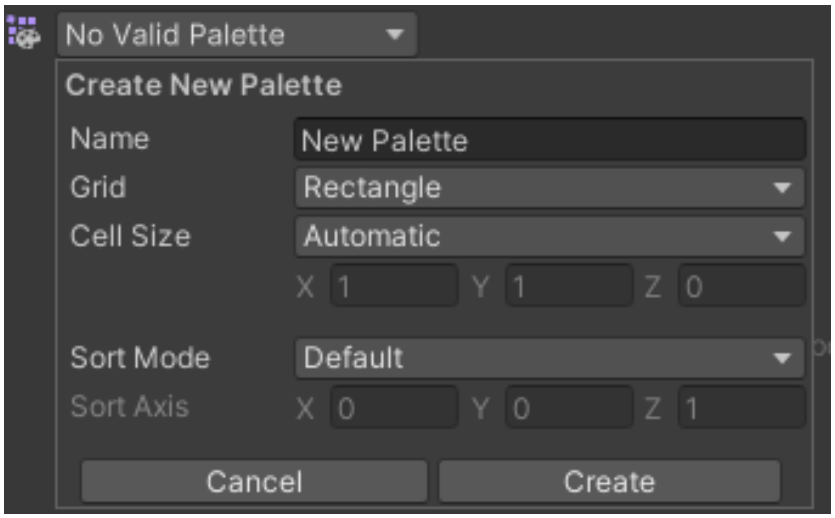
Vous pouvez déplacer la fenètre créée et la placer où vous voulez dans l'interface.



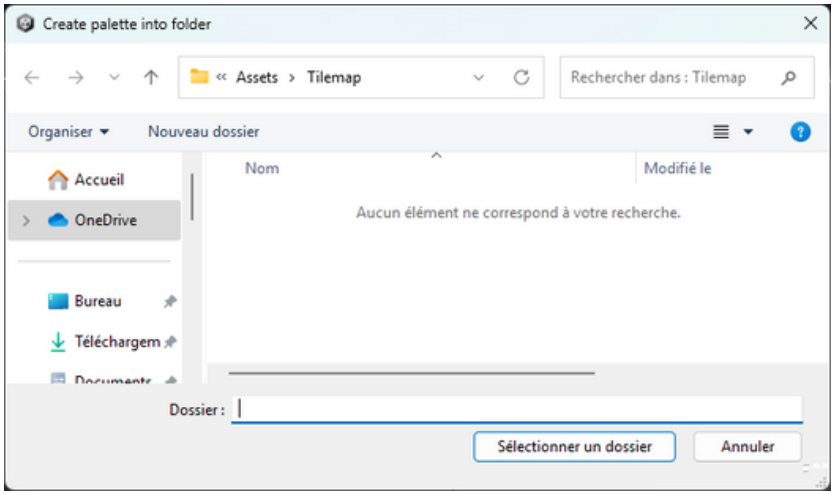
Une fois l'outil ouvert, cliquez sur le menu déroulant à Gauche (il devrait indiquer "No Valid Palette", puis cliquez sur "Create New Palette".



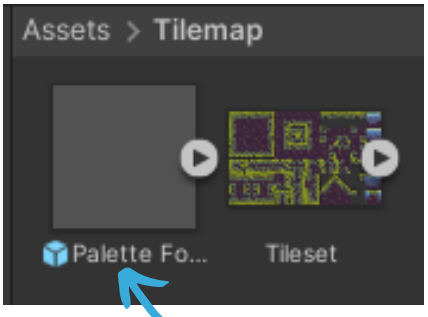
Un menu va s'ouvrir, vous pourrez choisir le nom de votre palette et vérifier que les autres paramètres sont corrects (Rectangle, Automatic et Default).



Un menu va s'ouvrir, vous permettant de choisir où l'asset de la Palette sera enregistré.

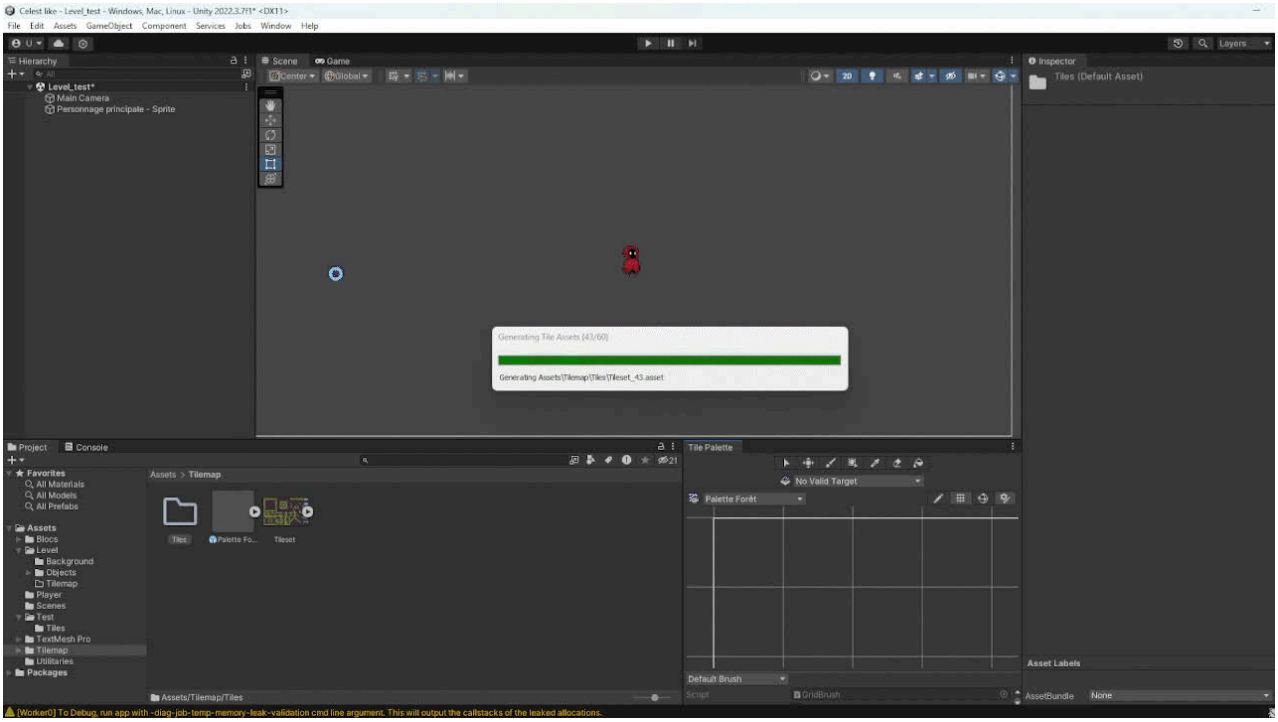


Une fois le bouton "Sélectionner un dossier" pressé, votre palette sera créé à l'endroit sélectionné.



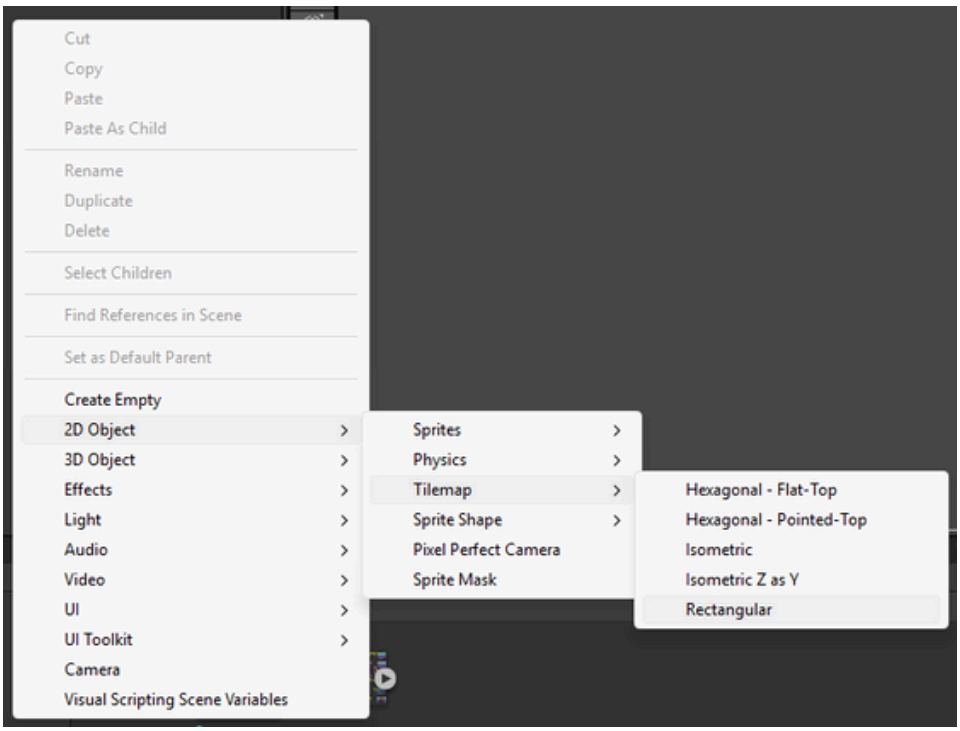
Vous pouvez maintenant prendre les Sprites que vous avez découpé et les ajouter à la Palette. Pour ce faire, glissez-déposer l'image dans la Palette. Vous devrez choisir un dossier dans lequel les Tiles seront enregistrées.

Chaque Tile sera un Asset (atout, à traduire comme "ressource" ici) qui devra donc être enregistré quelque part dans les fichiers du projet.



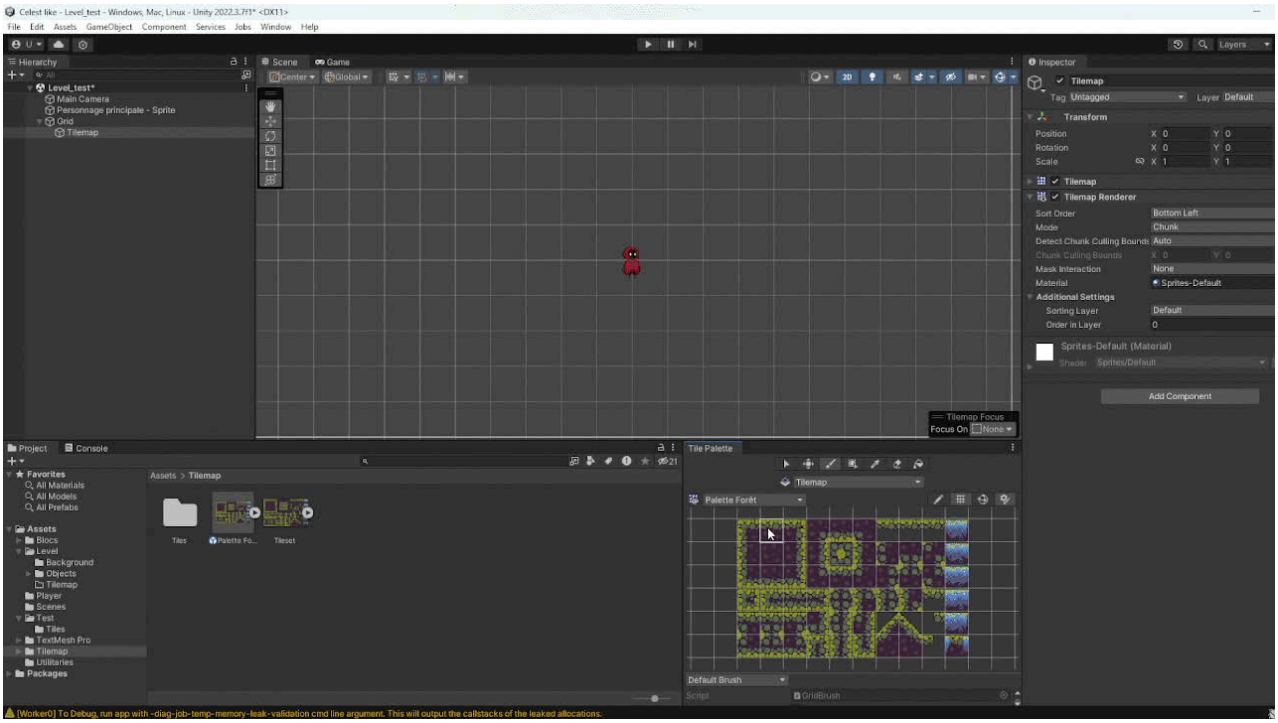
Vous vous retrouverez avec votre Palette remplie de Tiles. Il nous manque cependant quelque chose. Un peintre a certes besoin, pour peindre, d'une palette mais aussi d'une toile. Pour les Tiles la toile est une TileMap (carte de tuiles).

Pour créer la TileMap, faites clic droit dans l'arborescence, sélectionnez " 2D object", puis "Tilemap" et enfin "Rectangular".

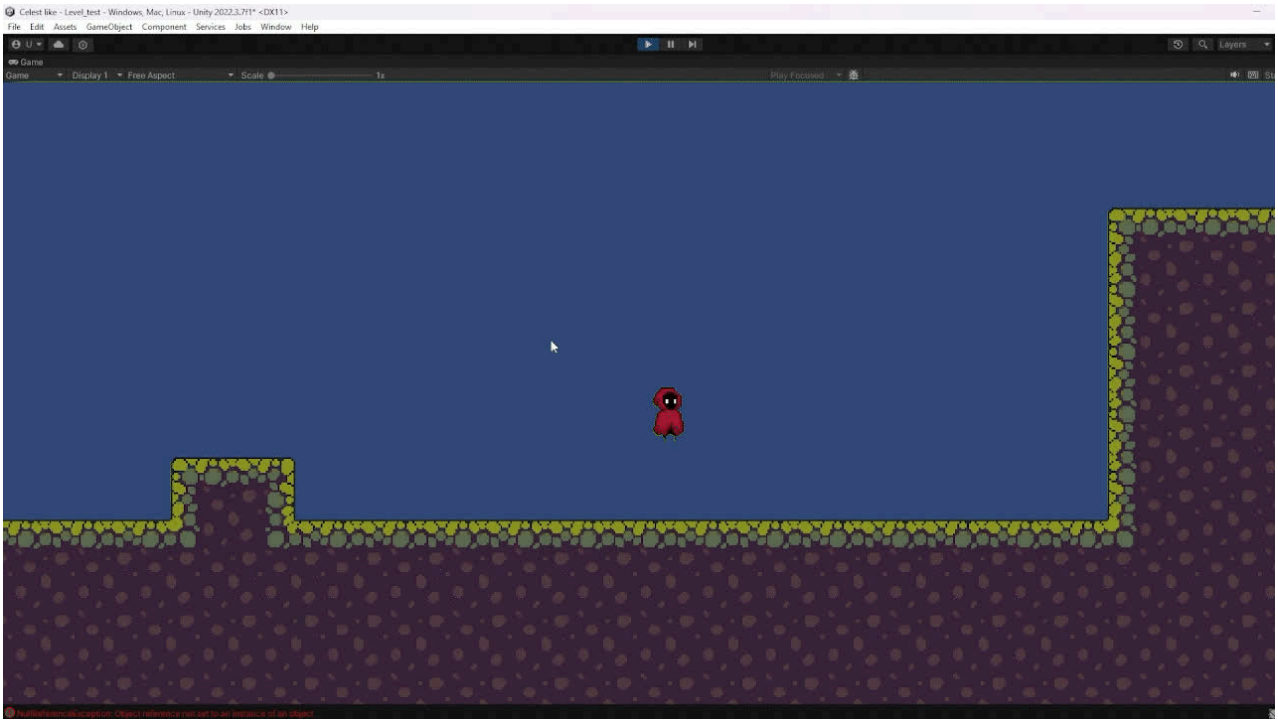


Deux GameObjects seront créés, un premier nommé "Grid" et un second nommé "Tilemap". Le premier est un objet qui défini une grille dans laquelle les Tiles vont venir se placer. Le second va contenir les Tiles et permettre de leur ajouter des composants.

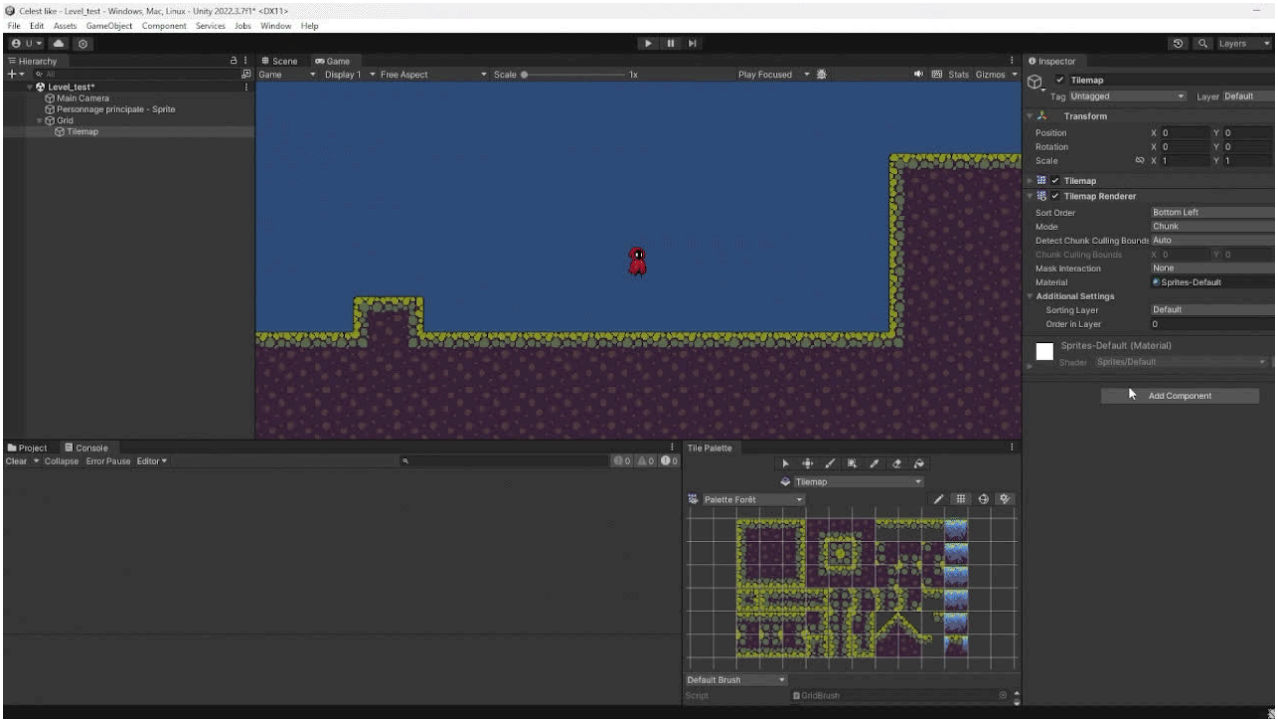
A partir de ce niveau ci vous pourrez peindre les Tiles sur la Tilemap. Pour ce faire, assurez-vous d'avoir le pinceau, sélectionné, sélectionnez une Tile en cliquant dessus et cliquez où vous voulez sur la Tilemap.



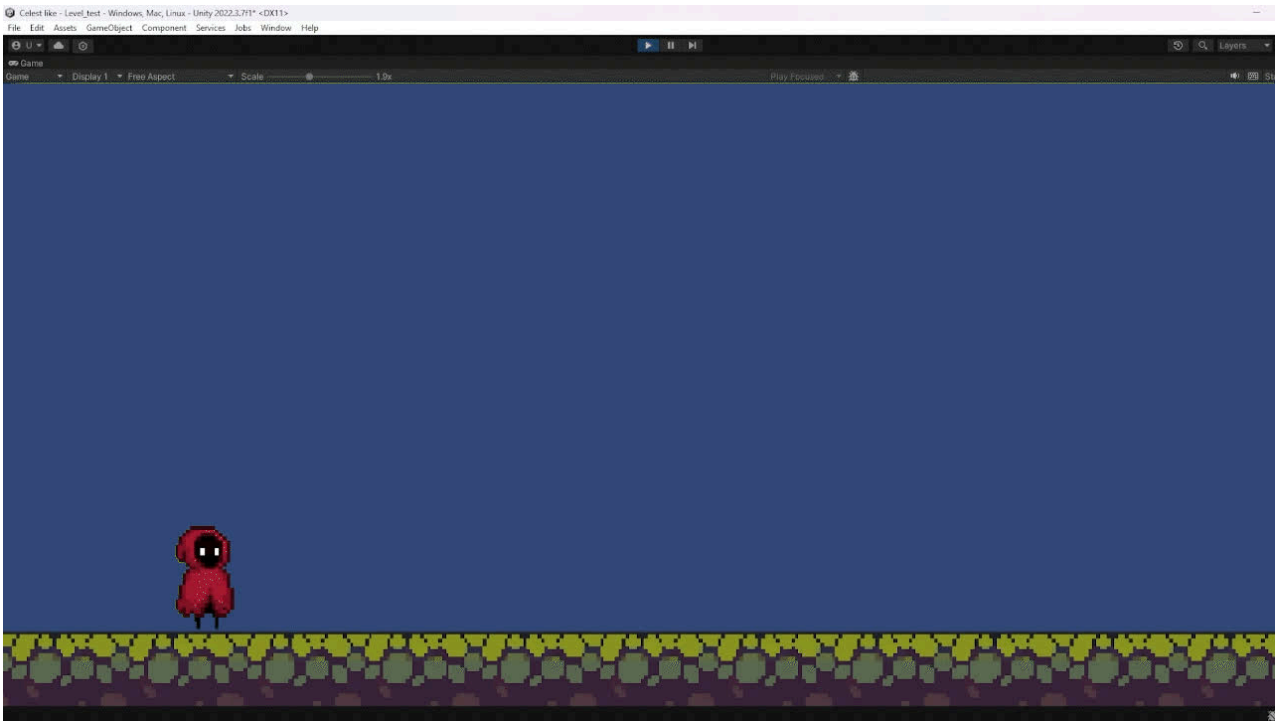
Cependant si vous lancez le jeu à cette étape vous verrez que votre personnage passe au travers des blocs peints.



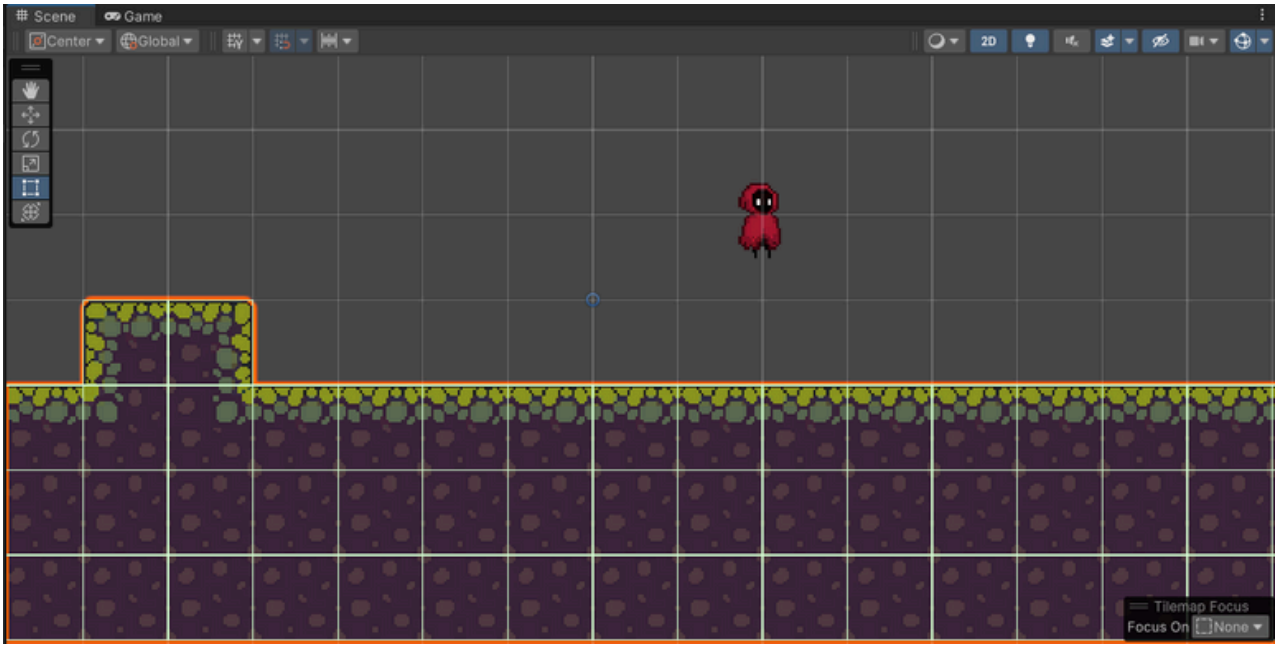
Il manque à la Tilemap un Collider, heureusement il existe un Collider spécialement pour elles. Ajoutez donc un composant TilemapCollider.



Vous observerez que le personnage ne passe plus à travers les blocs mais si vous essayez de le faire marcher vous le verrez comme sautiller.



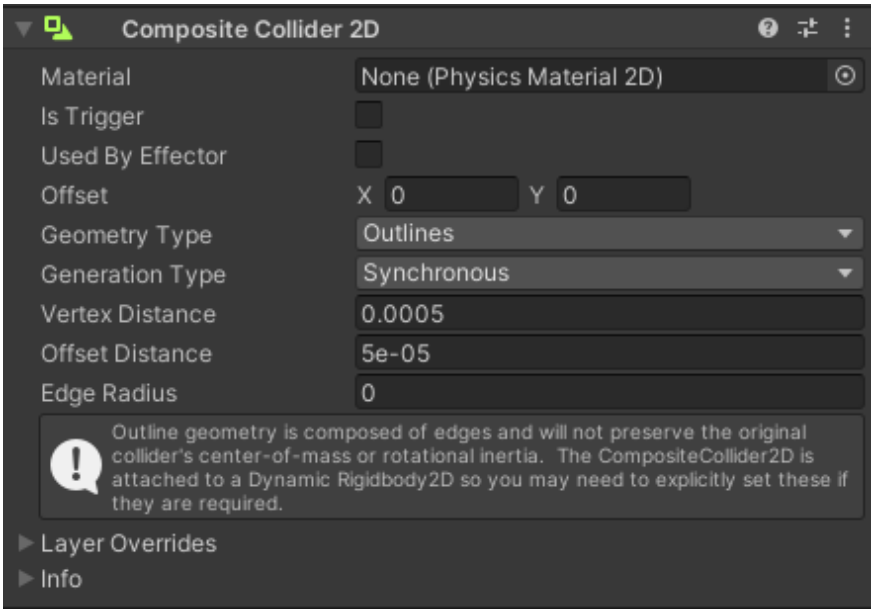
Ceci est dû au fait que chaque Tile a son propre Collider carré. Quand le personnage va arriver au bord d'une Tile il risque de rebondir dessus.



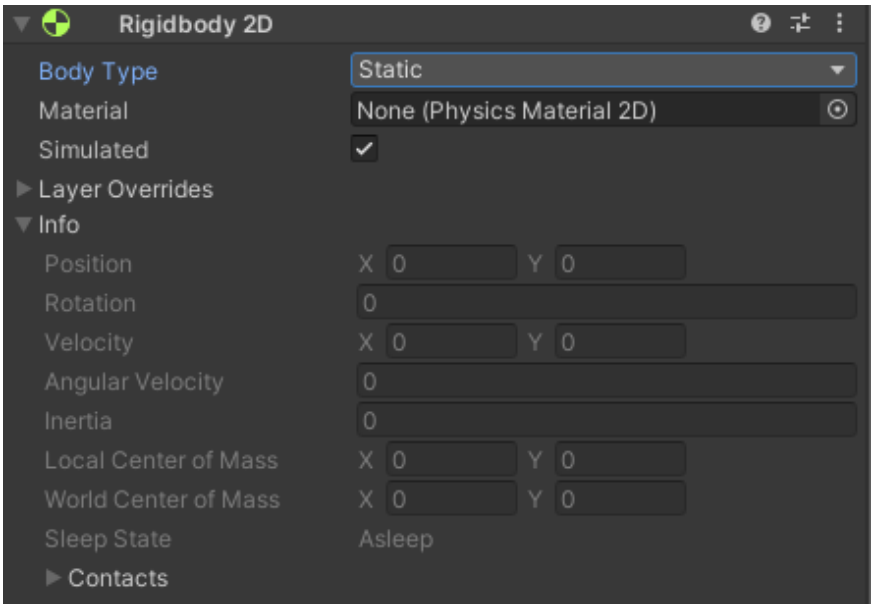
Pour régler ce problème il faudrait que tous les colliders des tiles de la Tilemap soient regroupés en un Collider.

Pour faire cela il existe un composant nommé Composite Collider. Il va prendre tous les Colliders attachés au même GameObject et en créer un unique qui recouvre tous les autres.

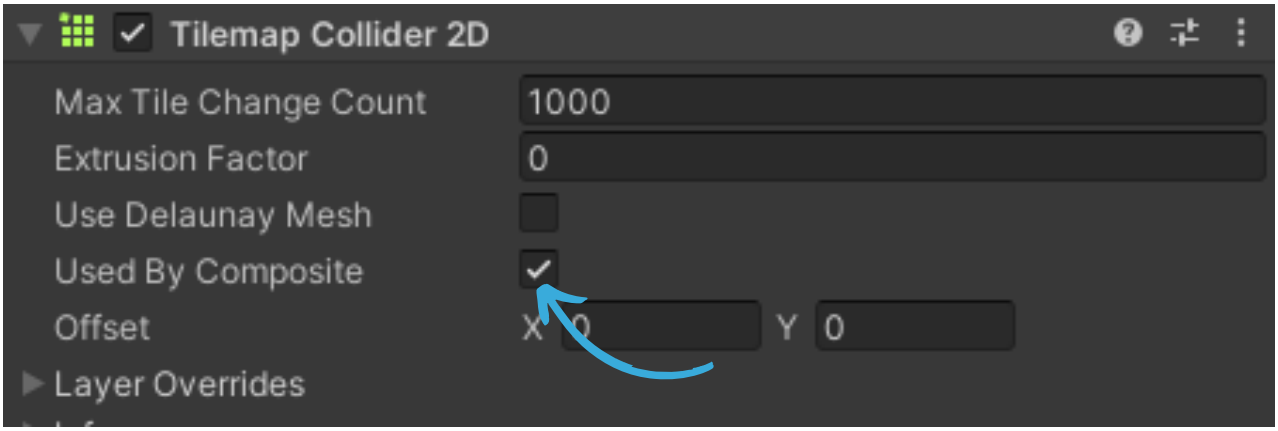
Commençons par ajouter le Composite Collider.



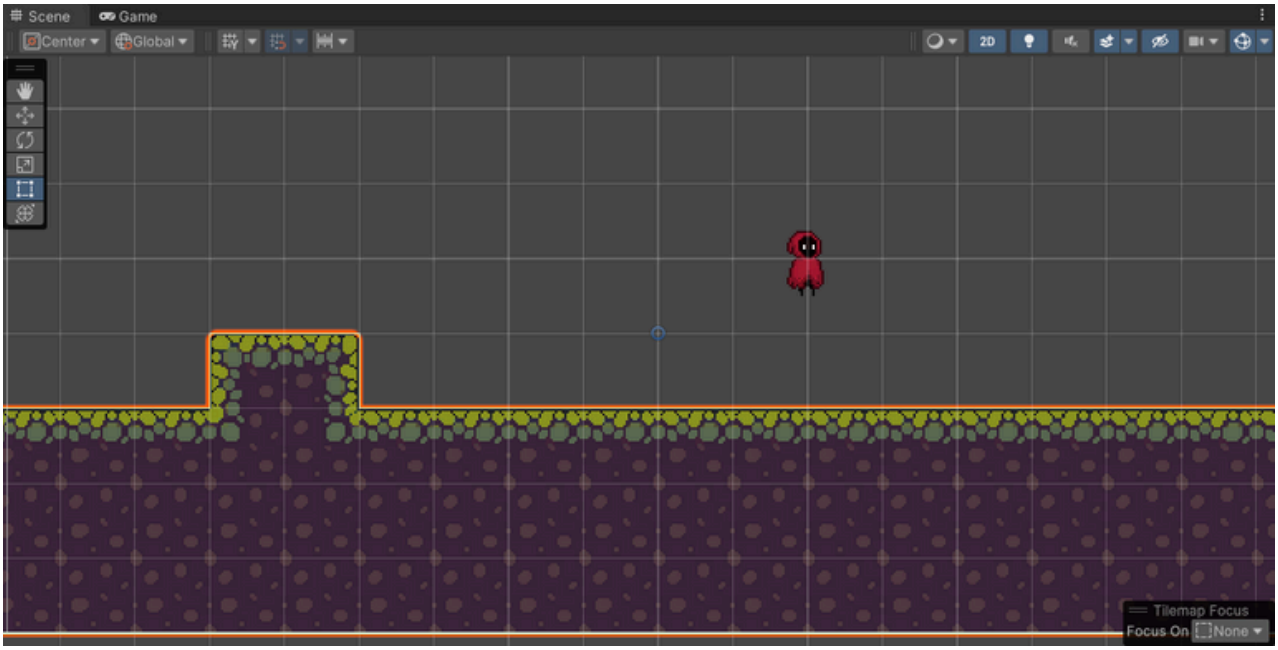
Un Composite Collider a besoin d'un RigidBody2D pour fonctionner. il aura été ajouté automatiquement quand vous avez ajouté le Composite Collider. Il faut changer le type de Body vers Static pour empécher la Tilemap de bouger.



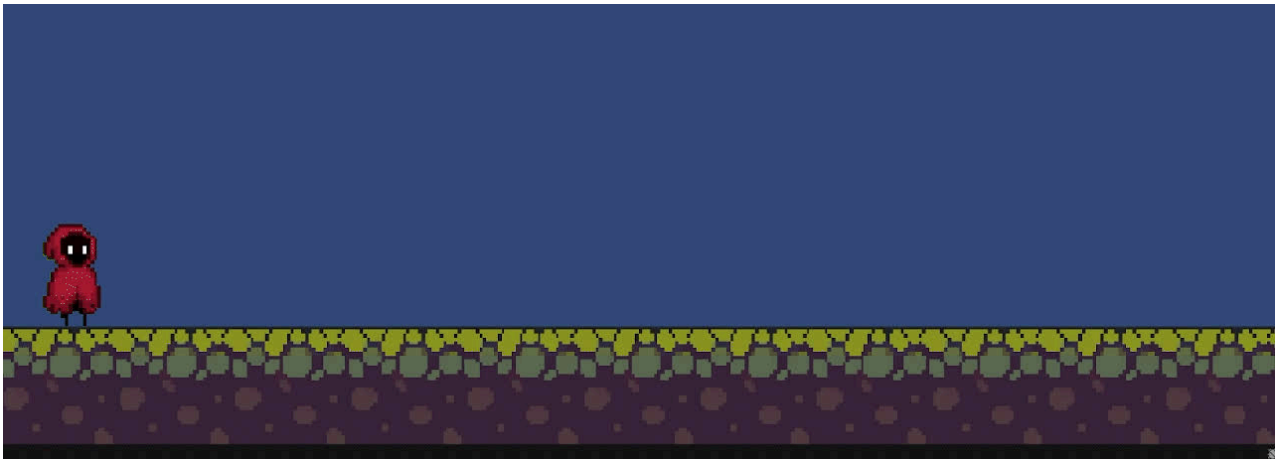
Il faut encore indiquer quel(s) Collider(s) vont être rassemblés par le Composite. Pour ce faire, cochez la case “Used By Composite” dans le Tilemap Collider.



Vous pourrez voir que le Collider couvre maintenant la totalité des Tiles.



e personnage devrait maintenant se déplacer sans rebondissements.



UI IMAGE

Les fonctions OnCollision et OnTrigger sont appelées sous certaines conditions liées au Collider attaché au même GameObject que le script.

OnCollisionEnter est appelé quand un Collider attaché au même GameObject que le script entre en collision avec un autre Collider.

OnCollisionExit est appelé quand un Collider attaché au même GameObject que le script ne touche plus un Collider qu'il touchait à la frame précédente.

Vous aurez une Font qui sera créé dans Unity, il ne vous restera plus qu'à sélectionner cette Font dans la case adéquate dans tous les textes qui doivent être écrits avec cette Fonte.