# Comment faire un platformer en 4 heures

---

Dans ce workshop vous allez poser les bases d’un platformer 2D sur unity. Ce workshop est plutôt à destination de novice d’Unity.

Si au cours de ce workshop vous avez des problèmes pour la réalisation de votre projet, n'hésitez pas à relire le pdf, toutes les informations nécessaires y sont présentes, vous pouvez aussi poser vos questions à l’organisateur de l'événement.

Ce workshop est en deux parties, dans cette première partie nous parlerons des bases d’unity, et de l’utilisation composant / script.

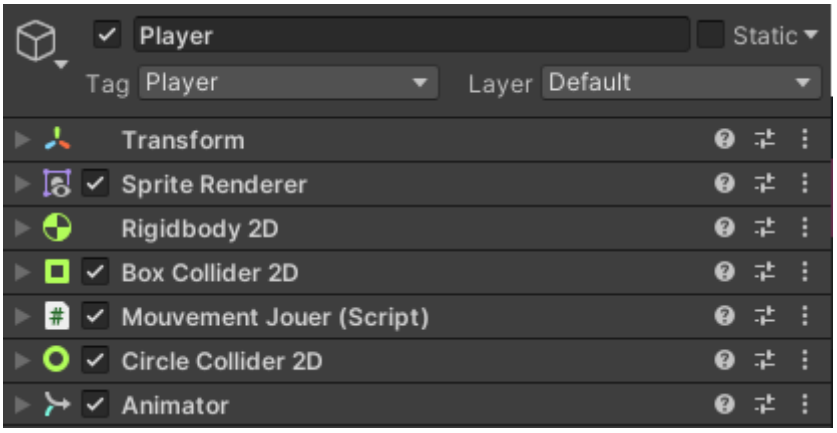
# Le fonctionnement d’Unity

---

Unity fonctionne avec l’usage de gameObject, de composant et de script (C#).

Un gameObject est définie par ces composants. Par exemple un gameObject “Player” peut à la fois être un “dossier de gameObject ” contenant d’autre gameObject concernant ce Player, ou bien / et contenir directement tous les composants nécessaires à ce Player.

Ici un gameObject Player, est un “dossier”, mais possède aussi plusieurs composants.

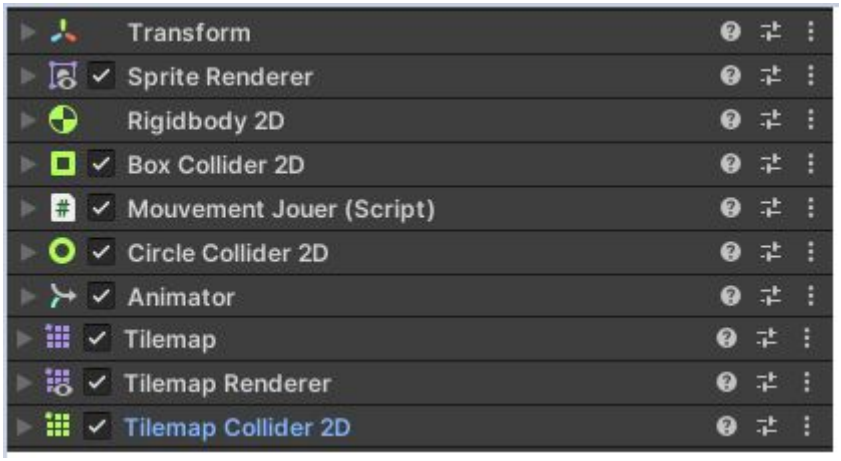



# Les composants

---

La partie la plus importante de ce que vous allez manipuler est le mélange composants / scripts.

Commençons par les composants suivants :




Ils composent la majorité de ce dont vous allez avoir besoin pour les fondations d'un platformer 2D.

`Transform :` c'est un vecteur 3D qui contient les informations de positions, de rotations et de scale. C'est le composants de base d'un gameObject , à la création d'un gameObject un composant transform lui est fourni.

`Sprite Renderer :` Il contient une image et son orientation. C'est la base de l'animation d'un personnage.

`Rigidbody 2D :` il rajoute de la gravité à un gameObject, il contient donc les informations de cette gravité.

`Box Collider 2D :` C'est une boite de collision 2D.

`Circle Collider 2D :` C'est un cercle de collision 2D.

`Animator :` Il contient les informations d'une animation, et du changement d'animation ( de IDLE à RUN par exemple).

`Tilemap :` C’est le composants nécessaire pour créer un décors 2D

`Tilemap Renderer :` C’est aussi un composants nécessaire pour un décors 2D, Il permet de manipuler les Tiles , il contient aussi le niveau de profondeur de ce décors.

`Tilemap Collider 2D :` Il ajoute de la collision aux Tiles du même gameObject.

`Scripts :` c’est ce qui contient du code et qui permet de faire le reste de votre projet, déplacement, conditions, règles du jeu …

# Les Scripts

---

Avec unity on peut plus ou moins ne pas faire de code. Cependant ça serait se priver du potentiel de cet outil.

Ici les fichiers contenants du code sont appelés “Script” , pourquoi ? C’est leurs noms de composants, ces scripts sont des classes C#.

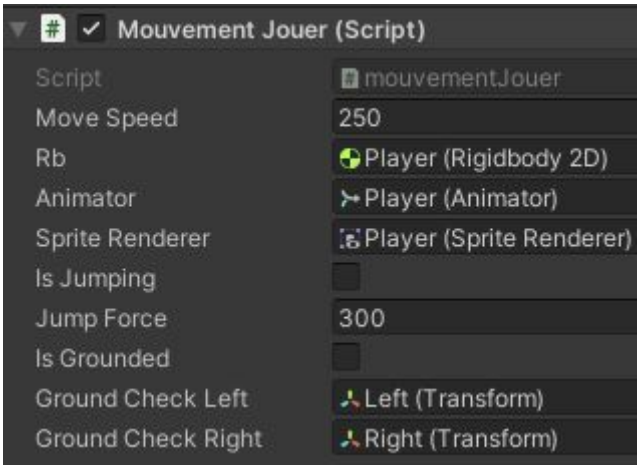
Vos scripts auront accès au composants de unity (méthodes, variables), chaque composants est une classe. Pour ce faire vos scripts auront :

```
public class mouvementJouer : MonoBehaviour
```

```
using UnityEngine;
```

**Vous allez être amené à utiliser ces classes sans pour autant en connaître tout le contenu, je vous conseille donc  google.**

Comment faire un lien entre un script et vos composants ?




Ici, si on fait attention, on remarque que mon script a des variables (MoveSpeed, Rb, animator ...), certaines de ces variables sont en fait des composants, on peut lire Rigidbody 2D, Sprite Renderer, etc etc.

En fait c’est simple, on peut glisser déposer des composants sur les variables de nos scripts, de cette manière le lien est fait entre nos script et composants.

# Posons les bases

---

Si vous n’avez pas la moindre idée de l'ordre à suivre pour faire votre projet, je vous propose de suivre les étapes suivantes:

- le terrain
- le joueur
- la caméra

Si vous avez finis ces trois étapes, il ne vous reste plus qu’à améliorer votre projet (objectifs, vie, conditions de mort / victoire, etc etc).

# 1- Le terrain

---

Tout d’abord il vous faut créer un nouveau projet sur Unity, faite bien attention à ce que ça soit un projet 2D.

Il vous faut une “Sheet”, c’est ce qui contient les blocs graphiques de votre projet.
Il vous faut modifier les paramètres suivants :

- 

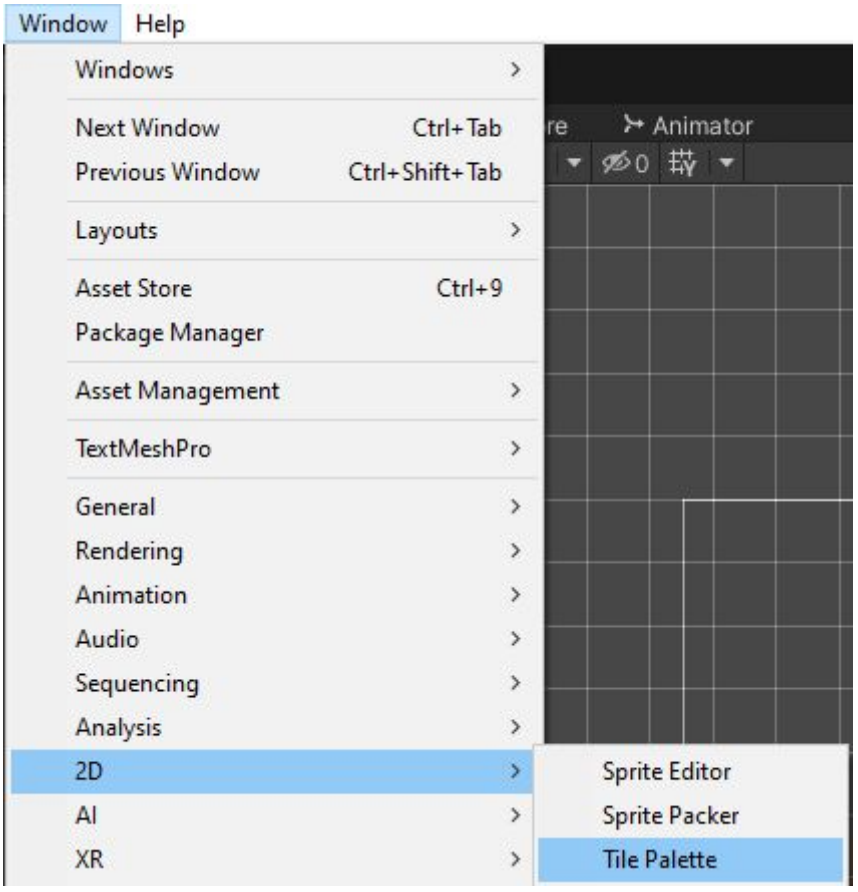
- 

- 


Il nous faut maintenant découper cette sheet : sprite editor -> slice -> 

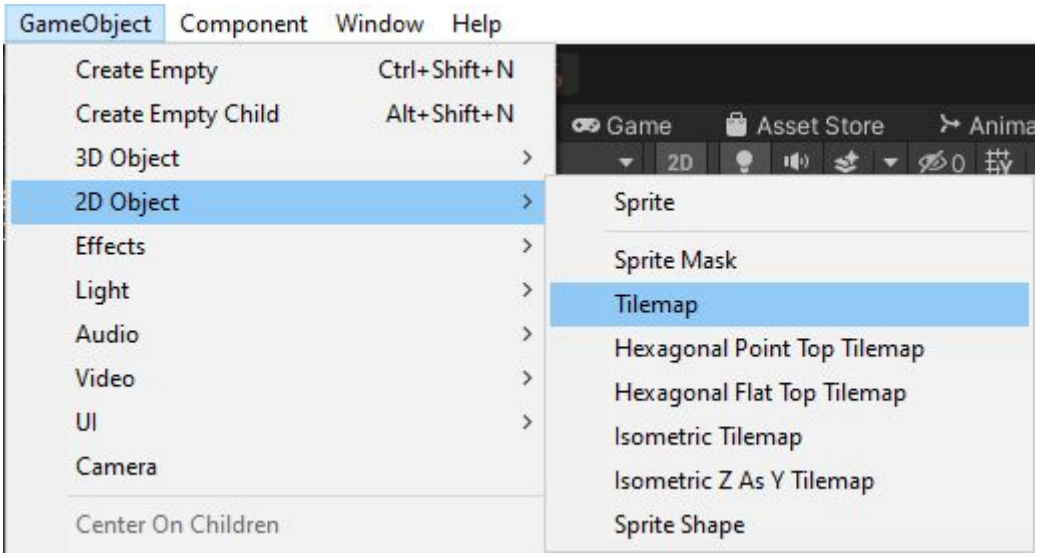
-> slice -> apply.

Nous devons créer une “tile palette” avec pour base notre sheet découpé:



Puis créer une nouvelle palette. Une fois cette palette créer nous devons faire un glissé déposé de notre sheet dans notre palette.

Nous allons maintenant utiliser cette palette. Pour cela nous allons créer un gameObject tilemap:

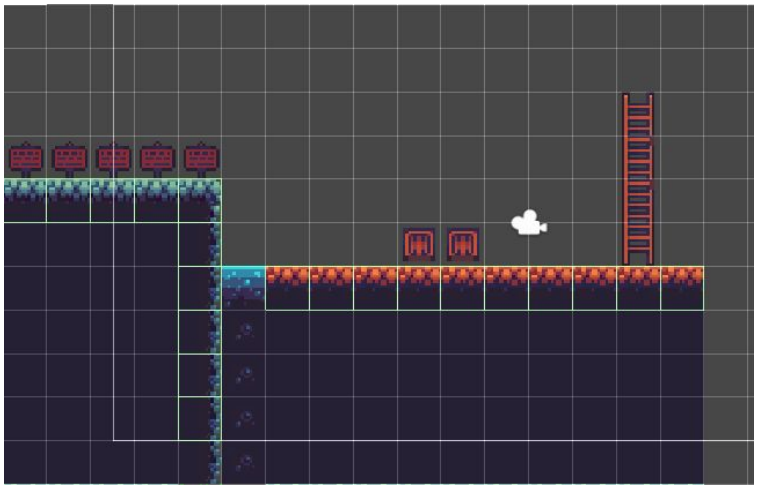



Ce qui nous créer les gameObjects Grid et Tilemap.

Grid va être notre “dossier” de Tilemap.
Pour donner de la profondeur à notre monde, nous allons créer deux tilemap, une première qui possédera : Tilemap, Tilemap Renderer, Tilemap Collider 2D. Et un second sans Tilemap Collider 2D.

Pour notre profondeur on a besoin de modifier le paramètre Order in Layer.
ça définit l’ordre d’affichage, nous avons besoin que notre gameObject contenant le tilemap collider 2D est un order in layer supérieur à notre seconde tilemap.
Je vous recommande de mettre des valeurs négatives pour les tilemaps secondaires.

Pour peindre notre décors nous devons désormais une tilemap puis placer les blocs de notre palette sur la scène.



# 2- Le joueur

---

Nous devons maintenant nous intéresser au Player.
Tout comme notre tilePalette, nous devons découper notre sheet contenant les sprites de notre Player.

On va pouvoir faire un glisser déposer d'une sprite de notre player sur la scène.

Ce qui nous a créé un nouveau gameObject !!

Notre joueur a besoin de plusieurs composants, on à besoin d'une hit box et d'être soumis à la gravité (on parlera de l'animation plus tard).

On va s'occuper du mouvement du joueur. Pour cela on va créer un script.
Dans ce script, comme il s'agit du jouer, nous allons travailler dans une fonction

```
void FixedUpdate()
```

Il est très important que notre fonction d'entrée soit celle ci.

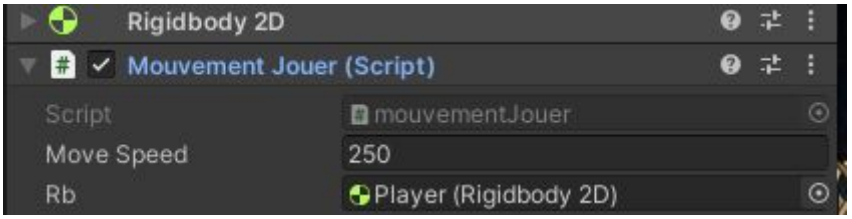
Pouvoir déplacer notre joueur avec les touches fléchés nous devons rajouter plusieurs variable :

```
public float moveSpeed;
private Vector3 velocity = Vector3.zero;
public Rigidbody2D rb;
```

Ainsi que les lignes suivantes:

```
float horizontalMovenment = Input.GetAxis("Horizontal") * moveSpeed * Time.deltaTime;
Vector3 targetVelocity = new Vector2(horizontalMovenment, rb.velocity.y);
rb.velocity = Vector3.SmoothDamp(rb.velocity, targetVelocity, ref velocity, .05f);
```

Les variables que nous avons déclarées précédemment doivent être liées aux composants de notre joueur, pour cela nous retournons sur unity et faisons un glisser déposer de composants sur la variable (dans le composant script).




Pour le saut de notre joueur nous aurons besoins de :

```
if (Input.GetKey("up"))
{
    rb.AddForce(new Vector2(0f, jumpForce));
}
```

Ainsi que d'un variable float jumpForce (je conseil une initialisation à 300 dans unity).

Maintenant que votre personnage se déplace et saute, nous devons régler quelque problémes :

- les saut multiples
- le sprite ne change pas de directions quand je me tourne
- le sprite n'est pas animé

Et je ne vais pas vous donner la réponse comme ça.

Pour les sauts, je vous conseil d'utiliser la fonction `Physics2D.OverlapArea` qui retourne un booléen, cette fonction vous prévient si il y a un objet avec une hit box entre deux transform.

Pour le sprite qui ne se retourne pas, vous pouvez modifier son Sprite Renderer `SpriteRenderer.flipX =` (c'est un booléen), il ne vous manque plus qu'a trouver une condition, je vous conseil de regarder du côté de `rb.velocity`.

# 3- La caméra

---

Nous allons maintenant regarder du côté de la caméra.
Notre objectif est d'avoir une caméra qui suit notre personnage.

Et comme j'ai été plutôt sympas jusque là, bonne chance…

Pour notre mouvement de caméra, nous allons travailler dans la fonction Update.
Nous n'avons besoins que d'une seul ligne:

```
transform.position = Vector3.SmoothDamp
```

Maintenant, Vector3.SmoothDamp prend 4 paramètres.
Le second paramètre est la somme de la position du Player et d'un vector3, bien que nous soyons en 2d, il y a un principe d'axe Z (de profondeur), ce vector3 a pour but d'englober notre profondeur (pensez à notre order in layout).

**Si vous êtes arrivé jusque ici, félicitation, la suite vous attend bientôt avec la partie deux de ce workshop.**