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 la suite du workshop “Comment faire un platformer en 4 heures”, dans cette seconde partie nous allons regarder du côté d'éléments indispensables des platformers comme l’animation des personnages ou bien le comportements des ennemis.

Posons les bases

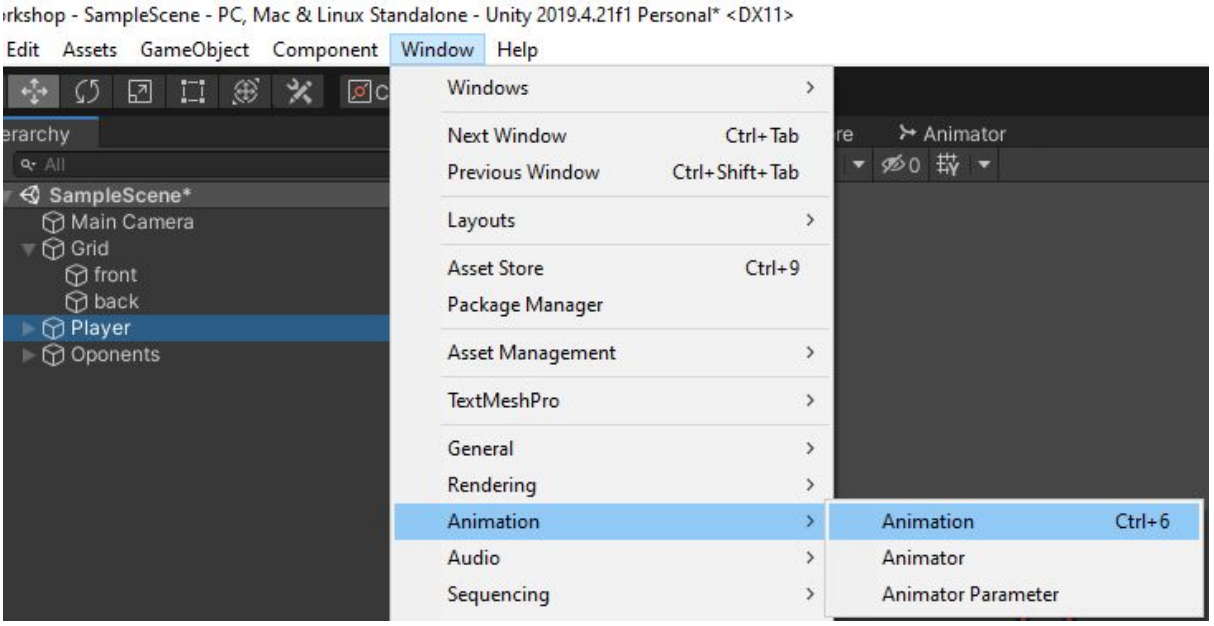
Si vous n’avez pas la moindre idée de l'ordre à suivre pour faire votre projet, je vous propose de suivre les étapes suivantes:

- les animations
- les adversaires
- la hitBox

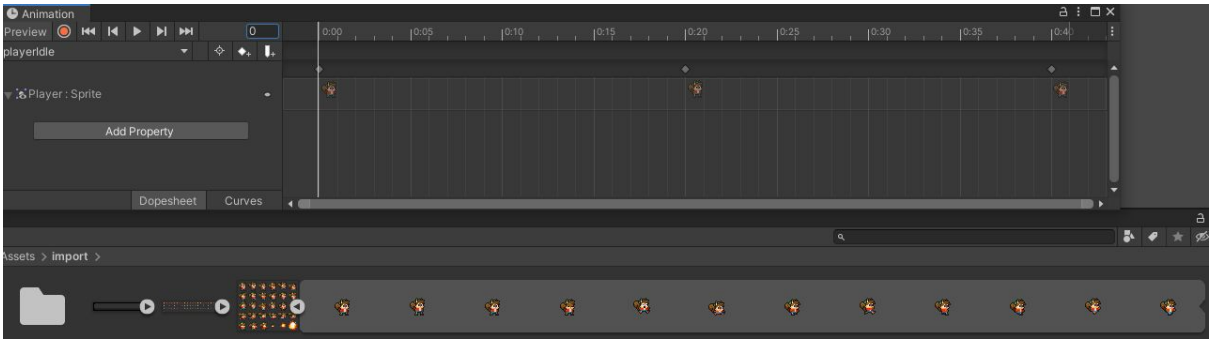
Si vous avez finis ces trois étapes, il ne vous reste plus qu’à améliorer votre projet (objectifs, vie, conditions de mort / victoire, etc etc).

1- Animation de sprite

Pour animer un sprite, il nous faut son sprite sheet, et savoir quelle image on va utiliser pour notre animation.



Maintenant ce n’est que du glisser déposer d'image sur notre timeline.

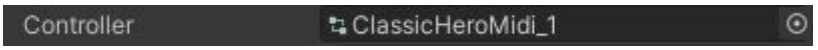


Si vous voulez bien faire les choses, pour votre joueur, je vous conseil de faire deux animation:

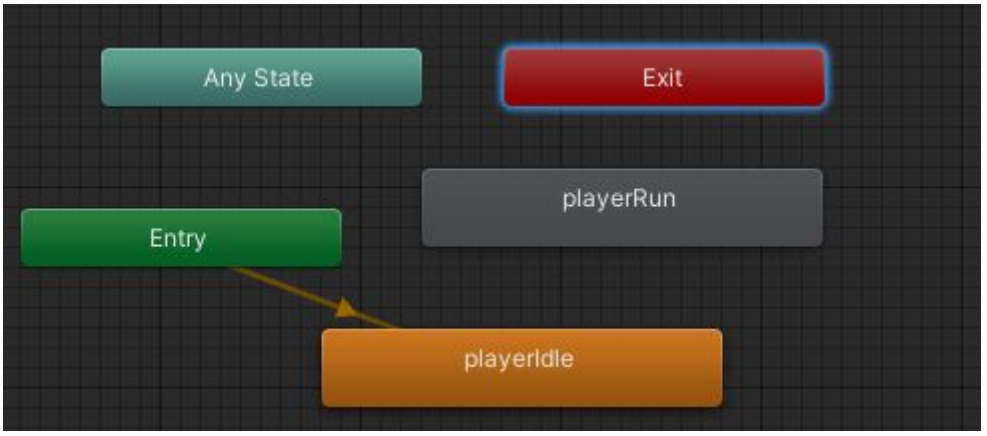
- celle d’idle (quand le joueur ne bouge pas)
- celle de course

Maintenant, il faut dire à notre programme lequel utiliser à quel moment.

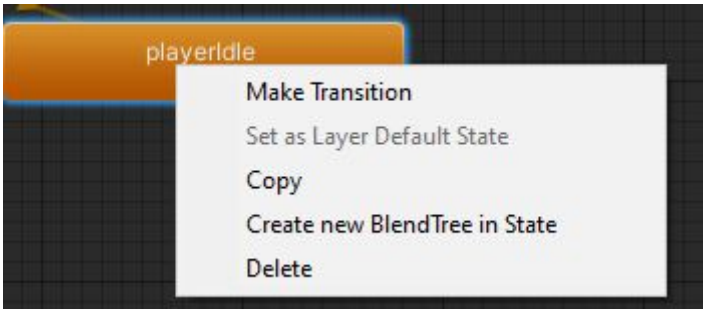
Le fait d’avoir rajouter des animation a notre gameObject lui a aussi rajouter un composant

On double clique sur 

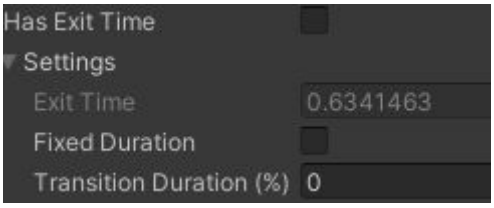
Et là, on panique.



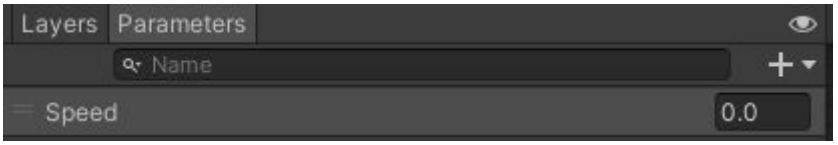
Le lien entre Entry et playerIdle défini quelle animation est jouée de base, Pour changer les liens on peut faire clic droit sur n'importe quelle animation .



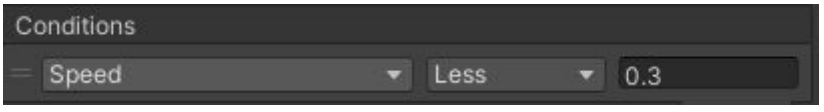
Faite des transition entre playerRun et playerIdle, vous pouvez cliquer sur les liens pour modifier certaine information de transition , comme nous somme ne pixel art, je vous conseil fortement d'utiliser les informations suivantes:

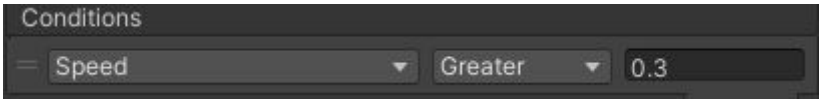


Il nous faut aussi prévoir une condition pour passer de Idle à Run. Pour cela nous devons déjà créer une variable liée au animation, je vous conseil fortement de faire une variable Speed (float):



Nous pouvons setUp les conditions:





Tout ça c'est beau … mais ça ne marche pas, pourquoi ?
Et bien notre variable Speed a été créée dans unity, mais sa valeur n'est jamais changée.
Pour cela nous devons retourner dans notre script de déplacement de joueur pour rajouter les lignes suivantes :

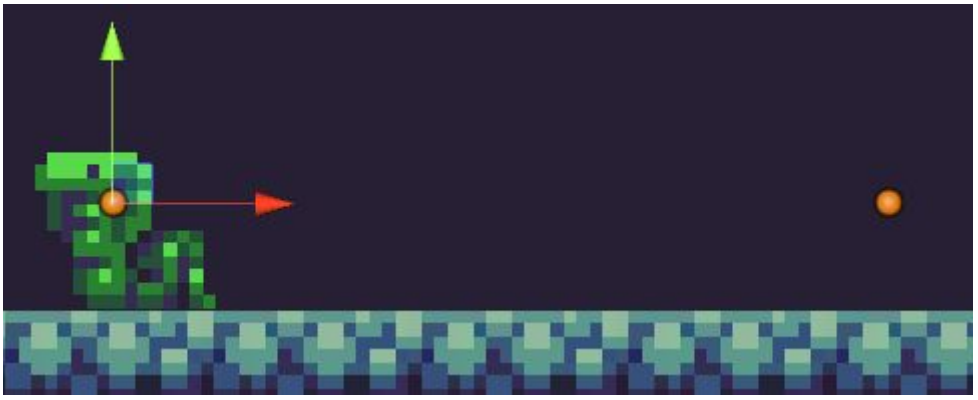
```
float characterVelocity = Mathf.Abs(rb.velocity.x);
animator.SetFloat("Speed", characterVelocity);
```

Et là normalement tout fonctionne.

2- Les ennemis (déplacement)

Pour cette dernière partie, on va regarder le mouvement et la gestion des dégâts.
Pour cette partie je vous conseille d'avoir déjà placé votre gameObject, ainsi que ces animation (ici pas besoin d'animation d'Idle).

Pour le mouvement de ces gameObjects, on va donner un pointA et un pointB, nos ennemis se déplaceront entre ces points (ici représenté en ronds orange).



Pour notre script, on va avoir besoin de deux transforms (qui sont nos deux points).

- D'une variable de vitesse.
- Une array de transform.
- Un transform;
- Un entier.
- Un SpriteRenderer

Pour ma part je les ai nommés "speed, waypoints, target, destPoint, graphics".

- "target" va servir de cible pour nos déplacements.
- "destPoint" (destination point), va nous servir a savoir quel transform contenue dans "waypoint" est notre cible, il va s'incrémenter au fil du temps.
- "target" est notre cible, une fois atteinte, passe au point suivant de la liste, si on a atteint le dernier point, on recommence au premier.

Dans la méthode Start, penser à initialiser votre variable "target".
Toute la logique se fera dans la méthode Update.

Pour le déplacement, comme pour notre joueur, on va utiliser un Vector3.
Il doit être égal à la comparaison des positions du transform "target" et du transform de notre ennemis.

```
Vector3 dir = target.position - transform.position;
```

Nous devons ensuite déplacer notre ennemis, pour cela il faut modifier son transform a la méthode “Translate”, penser à utiliser les paramètre suivant :

```
dir.normalized * speed * Time.deltaTime, Space.World
```

Maintenant , on a un adversaire qui va d’un point A à B, et c’est tout, on doit s’occuper de la condition pour changer de direction.
Et là encore, bon courage, vous allez avoir besoin, pour votre condition de comparer, la vitesse (0.3f) et la méthode Distance d’un Vector3.
Pensez aussi à ajouter les lignes suivantes dans votre condition:

```
destPoint = (destPoint + 1) % waypoints.Length;
target = waypoints[destPoint];
graphics.flipX = !graphics.flipX;
```

Et la normalement tout marche, sauf si vous avez oublié un composants, ou bien mal placé vos transform A et B. Si votre ennemi ne fait pas ce qui est attendu mais que votre code est bon, c’est que vous n’avez probablement pas bien compris le placement de vos point A et B, relisez le code!!!!

3- Les ennemis (dégâts)

On va maintenant placer un point faible a notre adversaire, pour (comme à la mario), le battre en lui sautant dessus.

Pour une fois ça va être simple. On commence par créer un nouveau gameObject fils de notre ennemis, on rajoute un box collider 2D en cochant le paramètre “is trigger”, on modifie la hit box pour qu’elle soit placée sur la tête de notre adversaire. On ajoute maintenant un script a ce gameObject.

On utilise une fonction d’unity “OnTriggerEnter2D” qui prend un Collider2D en paramètre.

Une fois dans cette fonction, on va utiliser la comparaison “CompareTag”.

```
if (collision.CompareTag("Player"))
```

Unity nous permet de créer des tags, et ces tag peuvent être liées à nos gameObject.
Nous allons donc lier notre player a notre tag “Player”:

Et nous allons détruire le gameObject de notre ennemis, tout cela grâce à la fonction Destroy qui prend un gameObject en paramètre.
Je vous rappelle ici, qu’un gameObject possède un transform, et que ce transform a accès à son gameObject père, et le père de son père aussi…

Si vous êtes arrivé jusque ici, félicitation, il ne vous reste plus qu'à améliorer votre projet.