Curso en Desarrollo de Videojuegos en Unity

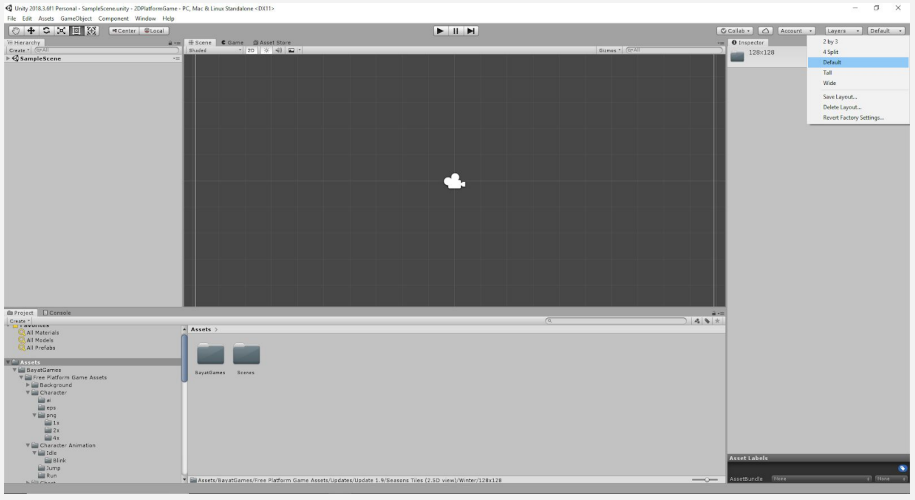
IES Puerta de Cuartos

Pablo Gómez Calvo

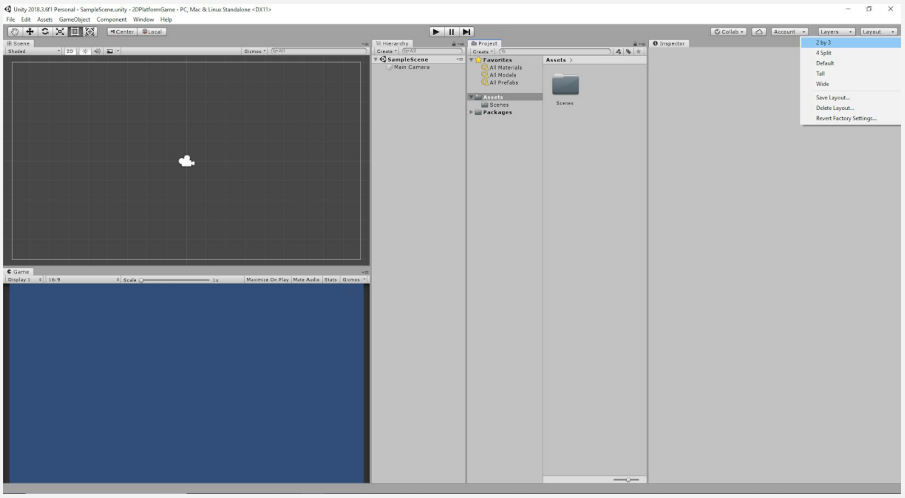
Índice

Capítulo 1: Set-up Unity

Lo primero que vamos a tener que hacer una vez abramos Unity será seleccionar la distribución que van a tener nuestro *Layout*: Para la parte más relacionada con diseño vamos a usar la ***Default***, pero una vez empecemos a programar y entremos más en la fase de *debugging* se recomienda usar ***2 by 3***.



Default



2 by 3

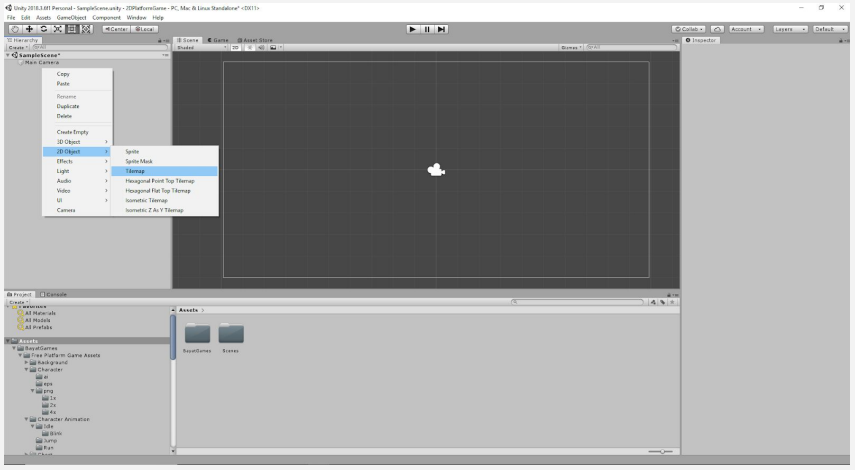
Capítulo 2: Creación de un Tilemap

Nuestro objetivo final va a ser crear un juego **2D Plataformas**. Para ello necesitamos a necesitar un mapa con el que debe interactuar nuestro personaje. Este tipo de mapas se llaman **Tilemaps**. Un *tilemap* está compuesto por *tiles*, que son trozos de una imagen, normalmente cuadrados, con los que poder hacer un mapa todo lo grande que queramos.

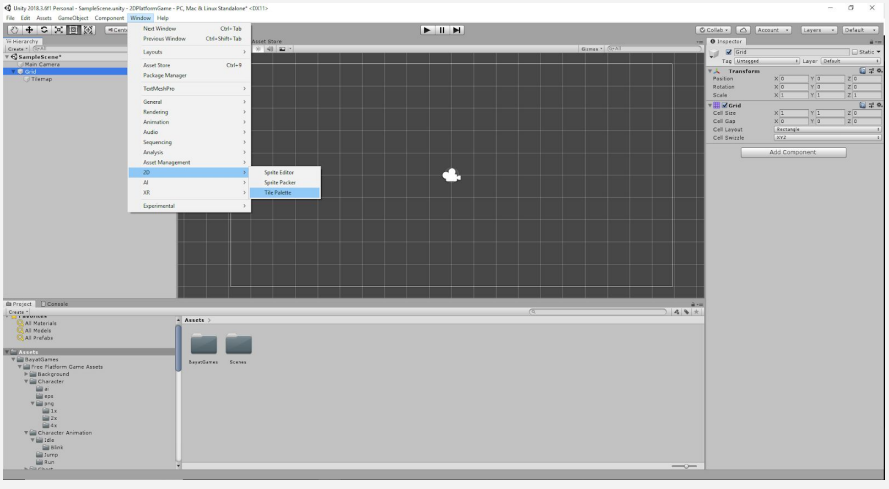
Para este curso se proporciona todo el arte que vamos a usar ya que el objetivo de estas sesiones es el aprendizaje tanto de un lenguaje de programación como es **C#** (leído C Sharp) como la utilización de una herramienta profesional con la que se hacen juegos AAA.

Los *tiles* que se van a usar para esta sesión se encuentran en la ruta **2DGame-For-High-School-Class\2DPlatformGame\Assets\BayatGames\Free Platform Game Assets\Tiles\png\256x256.**

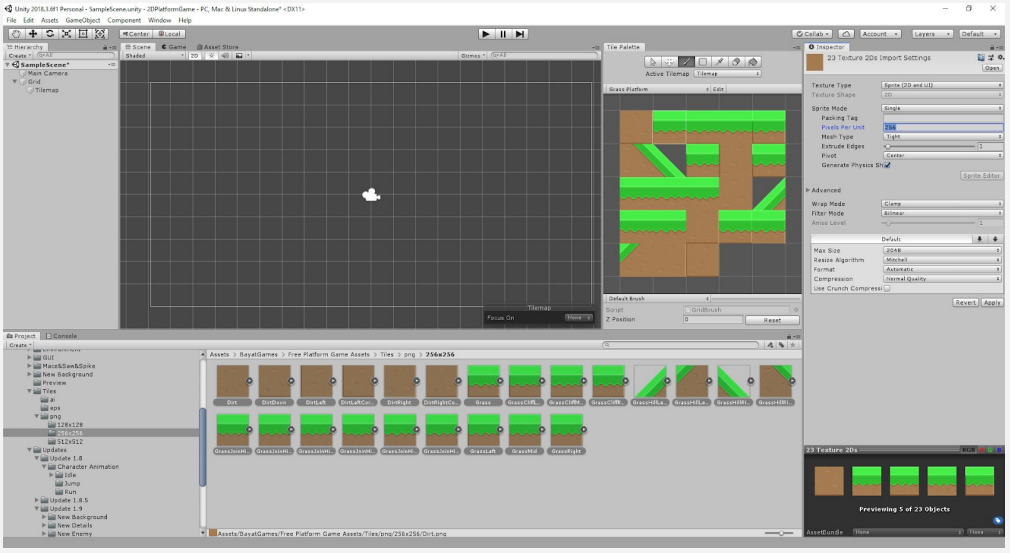
Una vez lo tenemos localizado, haremos click derecho justo debajo de la *Main Camera* > *2D Object* > *Tilemap*.



Una vez lo tengamos, nos iremos a *Window* > *2D* > *Tile Palette.*



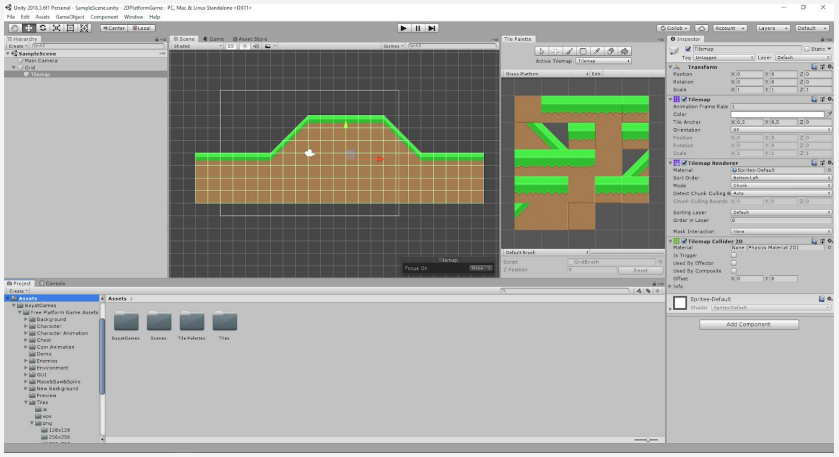
Nos aparecerá una ventana que podremos colocar donde nosotros queramos, la recomendación es dejarla en el *layout* al lado de la cámara. El siguiente paso será crear la nueva paleta de tiles que vamos a utilizar para crear nuestro mapa. Vamos a crearla con el nombre que queramos y posteriormente arrastraremos todos los tiles que están en la ruta mencionada anteriormente a la paleta de tiles que se nos ha creado.



El siguiente paso será cambiar una variable que tenemos en el inspector de la parte derecha de nuestro *layout*. Esta variable es ***Pixels per unit***. Esta variable define los pixeles que va a ocupar nuestra imagen tipo tile dentro de las unidades de nuestra escena de juego.

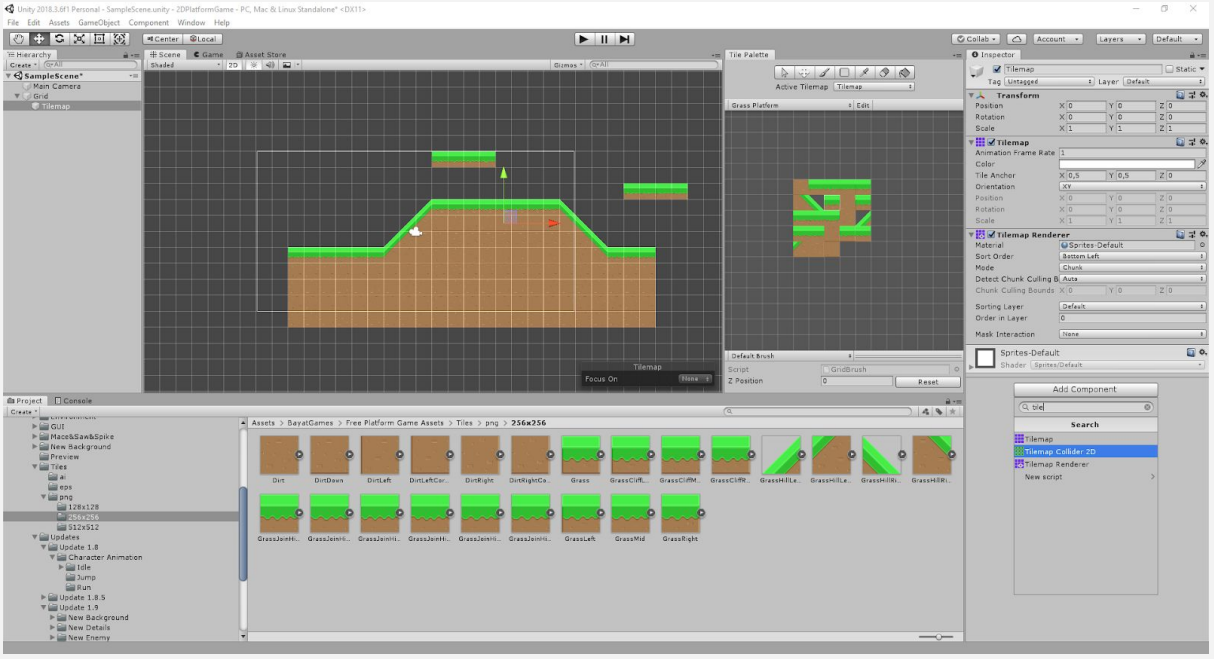
Como hemos cogido los tiles de **256x256**, tenemos que sustituir el **100** que viene por defecto por nuestro **256**. De ahí la importancia de que nuestros tiles sean cuadrados.

Una vez ya tenemos configurado todo esto, ahora hay que dar rienda suelta a nuestra imaginación e ir seleccionando cada tipo de tile y ponerlo en la escena de nuestro juego en la disposición que más nos guste. Aquí te dejo un ejemplo de una distribución en caso de que no se te ocurra nada.



Nuestro mapa ha quedado tal y como nosotros queríamos *(esto es modificable más adelante en caso de cambiar de idea)*. Ahora vamos a meter un concepto que en **Unity** es fundamental, el concepto de **Componente.**

Un **Componente** es un comportamiento que vamos a añadir a los **Objetos** de nuestra escena. Para hacer esto vamos a irnos a la lista de objetos que hay a la izquierda de nuestro *layout* y vamos a seleccionar el objeto **Tilemap** y a la derecha de nuestra pantalla aparecerá lo que vamos a llamar el inspector de nuestro objeto. En la parte baja del inspector hay un botón que pone ***Add Component***. Le damos click y escribimos ***Tilemap Collider 2D*** y seleccionamos esa opción del desplegable.



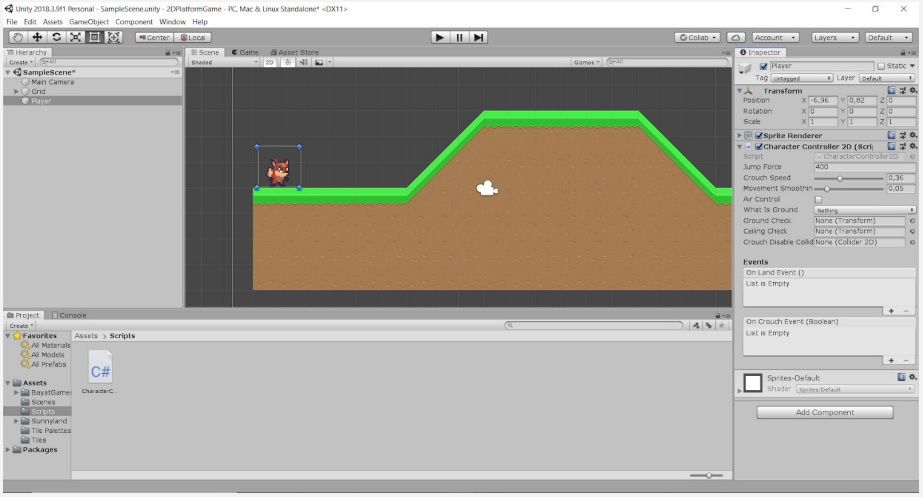
Esto lo que acaba de hacer es ponernos en cada *tile* un recuadro verde que en el siguiente capítulo veremos que tiene que ver con la colisión con nuestro personaje para que este no lo atraviese.

Capítulo 3: Personaje

3.1- Colisiones

Lo primero que vamos a necesitar una vez llegados a este punto es meter el personaje en la escena. Para esto, vamos a irnos a **Assets\Sunnyland\artwork\Sprites\player\idle** y de aquí cogeremos la primera y la arrastraremos a la escena. Seguramente aparecerá muy pequeño por lo que igual que hicimos con los muros, vamos a cambiar el valor de la variable ***Pixels per Unit.*** En mi caso con el valor 20 se veía con un tamaño adecuado para mi.

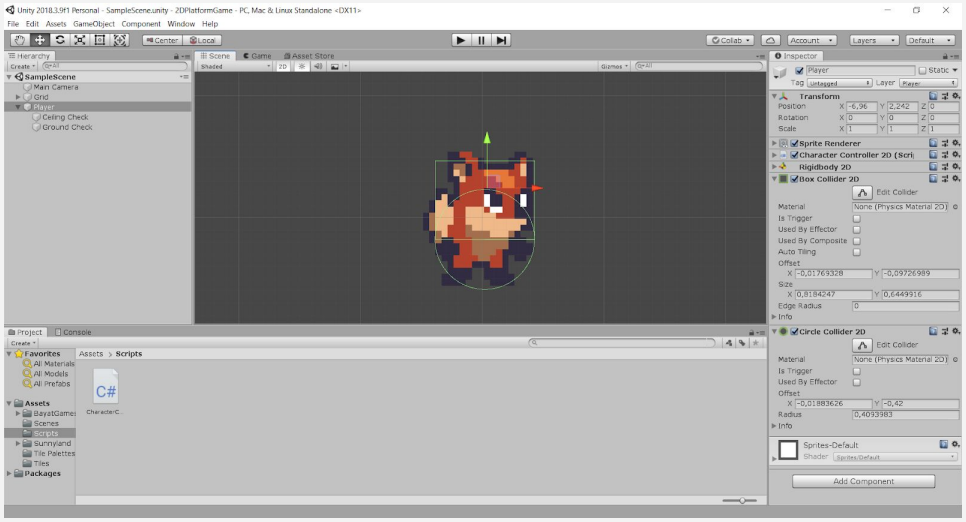
El siguiente paso va a ser darle un comportamiento de movimiento y para hacer las cosas mucho más fáciles, dentro del proyecto hay un script llamado ***Character Controller 2D***. Vamos a incluir este script a nuestro personaje.



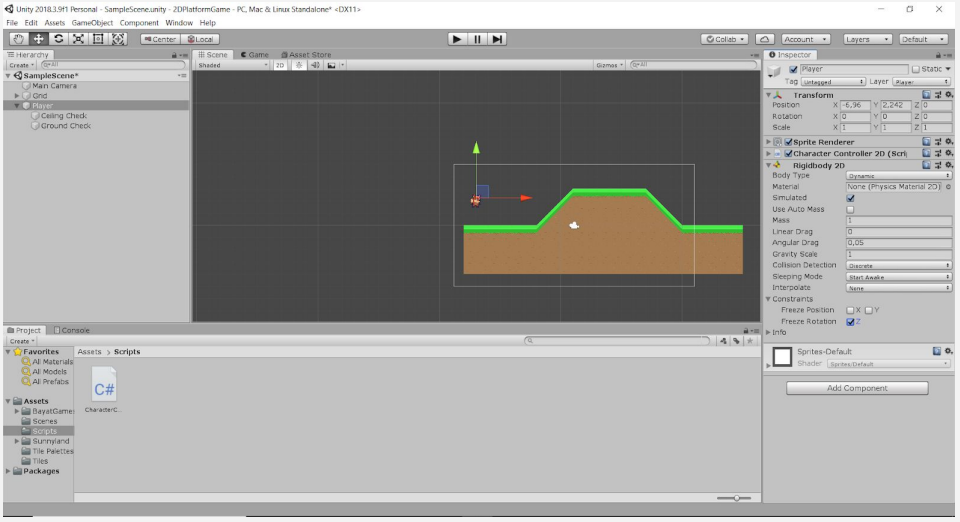
En los atributos del script podemos apreciar 3 casillas en las que pone **None(...)**, para esas 3 casillas vamos a necesitar crear unos objetos vacíos. Para esto, haremos click derecho encima del objeto **Player** y seleccionaremos ***Create Empty***. Este objeto lo colocaremos en la frente del personaje con la herramienta de mover (segundo icono de arriba a la izquierda). Una vez colocado, crearemos otro y lo pondremos entre los pies de

nuestro personaje muy cerca de la planta de los pies. Se recomienda cambiarlos el nombre, en mi caso los he cambiado a ***Ceiling Check*** y ***Ground Check*** respectivamente. Una vez hecho esto arrastraremos cada uno de estos a la casilla que tiene su mismo nombre en el *Inspector* del **Player** en el script que incluimos antes.

Después de esto vamos a utilizar 2 *Colliders* para hacer las colisiones con el mapa. Meteremos un **Box Collider 2D** y un ***Circle Collider 2D*** y modificaremos su forma con el botón de *Edit Collider* hasta conseguir algo parecido a esto:

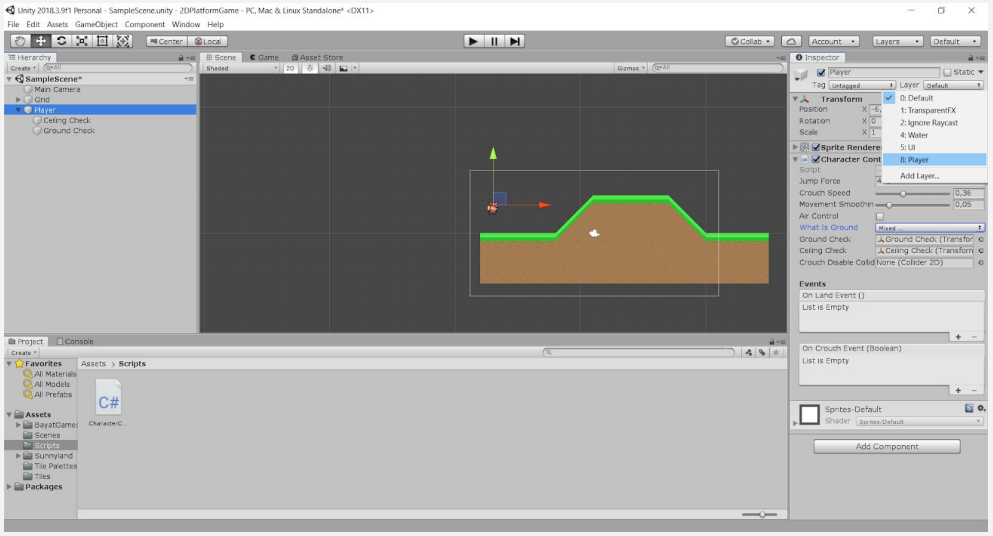


Ahora vamos a aplicar física a nuestro personaje para que caiga por gravedad. Para esto lo único que tenemos que hacer es añadir el componente ***Rigidbody 2D***. Con esto ya caería y se quedaría encima del terreno pero para evitarnos problemas en el futuro, vamos a bloquear su rotación en el **eje Z** para cuando se choque con algo, este no se ponga a dar vueltas sobre sí mismo. Para esto, haremos click en un *checkbox* en el *Rigidbody 2D* > *Constraints* > *Freeze Rotation*.



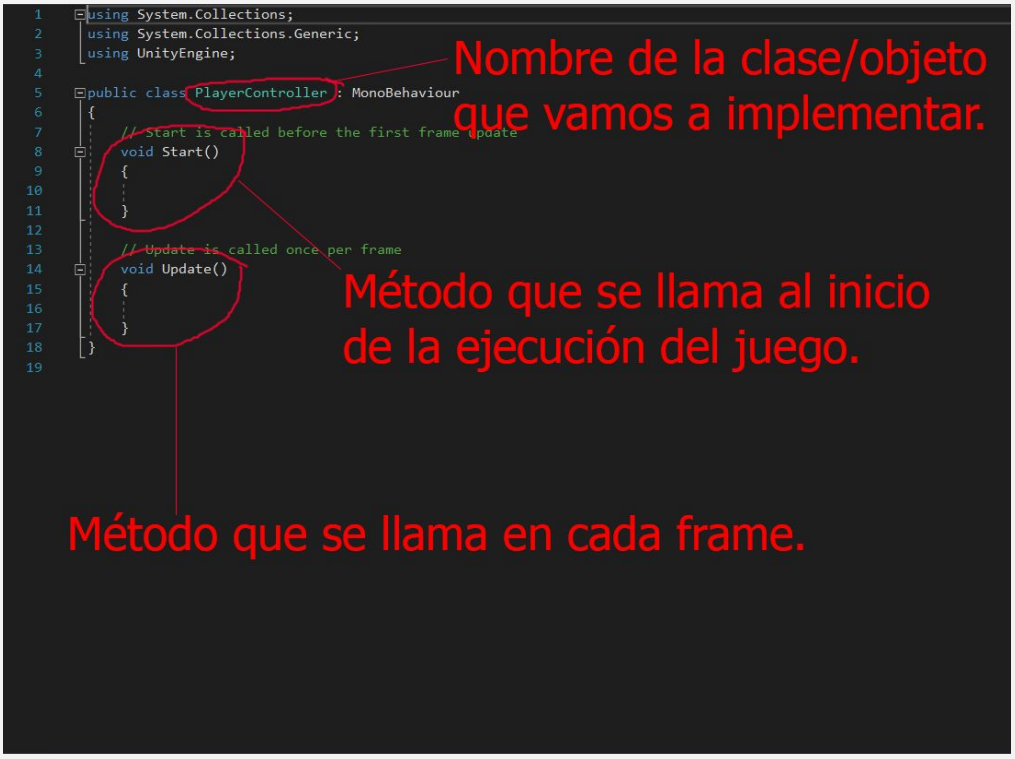
Hay un último parámetro que debemos modificar en el ***Character Controller 2D*** y es ***What Is Ground***. Este parámetro se usa para saber que cosas con las que nos choquemos, tiene que identificarlas como que son parte del suelo/mapa. De primeras pondremos everything pero entonces nuestro mismo jugador también estaría incluido y no queremos eso. Por lo tanto vamos a introducir un concepto nuevo que son las ***Layers***. En la parte más alta del inspector de nuestro personaje, hay un desplegable que dice **Layer : Default.** Abrimos el desplegable y pulsamos ***Add Layer…*** y vamos a la primera casilla que esté libre y nos deje escribir y pondremos ***Player.*** Pulsamos fuera del inspector y nos volvemos a

meter en nuestro personaje, abrimos el desplegable y asignamos a nuestro personaje la ***Layer : Player*** y en el desplegable del *Character Controller 2D* y deseleccionamos *Player*.



3.2- Movimiento

Y ahora es cuando vamos a empezar con la programación. Para empezar nos iremos a nuestro directorio **Scripts** y haremos ***click derecho > Create > C# Script***. Lo llamaremos por ejemplo ***PlayerController***. Lo abrimos y nos encontramos con esto:



Lo primero que vamos a necesitar es una referencia al script que tanto tiempo hemos invertido en moldear anteriormente, ***Character Controller 2D.*** Para conseguir esta referencia haremos:

```
public CharacterController2D controller;
```

Esto lo haremos en la zona de las variables de la clase. Haciendo esto, en el inspector de *Unity* nos habrá salido una casilla en nuestro *script* de *PlayerController.* Lo que haremos será arrastrar el script de ***Character Controller 2D*** a esa casilla y así lo habremos referenciado. Con esto hecho ya podemos tener acceso a todo lo que contenga el script a través de la variable controller que hemos creado.

Cuando nosotros movemos cualquier objeto, este tiene una velocidad. Es por eso que en videojuegos también tenemos que usar la velocidad si queremos mover un objeto

una distancia grande en poco tiempo. Vamos crear nuestra variable velocidad y vamos a darle el valor 40.

```
public float velocity = 40f;
```

Ahora veremos cómo capturar el evento de ***Input*** de teclado en este caso. Tenemos la suerte de que *Unity* nos define ya varios tipos de acciones entre las que se encuentra el movimiento horizontal y el salto. El movimiento horizontal en teclado tiene 3 posibles valores (0 si no pulsas nada, -1 si pulsas la "A" o la flecha de dirección izquierda y 1 si pulsas "D" o la flecha de dirección derecha). Este valor vamos a guardarlo en una variable que posteriormente multiplicaremos por la velocidad y haremos que nuestro personaje se mueva. Como el ***Input*** es algo que tenemos que mirar todo el rato, haremos esto en cada **Update** pero la declaración de la variable la haremos arriba junto al resto.

```
float movimientoHorizontal = 0;
```

```
void Update()
{
    movimientoHorizontal = Input.GetAxisRaw("Horizontal") * velocity;
}
```

Ahora que hemos guardado la dirección y la velocidad a la que nos vamos a mover, toca moverse y para eso vamos a llamar a la clase que nos guardamos en la variable *controller* antes. Como queremos que nuestro personaje se mueva a la misma velocidad vayamos a 20fps que a 120fps vamos a utilizar la función **FixedUpdate**.

```
void FixedUpdate()
{
    //Mover al personaje
    //Time.fixedDeltaTime se usa para que no importe lo rápido que sea tu ordenador. Es el tiempo entre un frame y otro

    controller.Move(movimientoHorizontal * Time.fixedDeltaTime, false, false);
}
```

Los 2 **falses** que hay en la llamada al método **Move** significan si queremos que nuestro personaje se pueda agachar y pueda saltar. Eso será lo siguiente que implementaremos y quitaremos esos valores de **false**.

Para saltar haremos lo mismo que con el movimiento horizontal. Crearemos una variable igual que *movimientoHorizontal* pero que se llame salto.

```
bool salto = false;
```

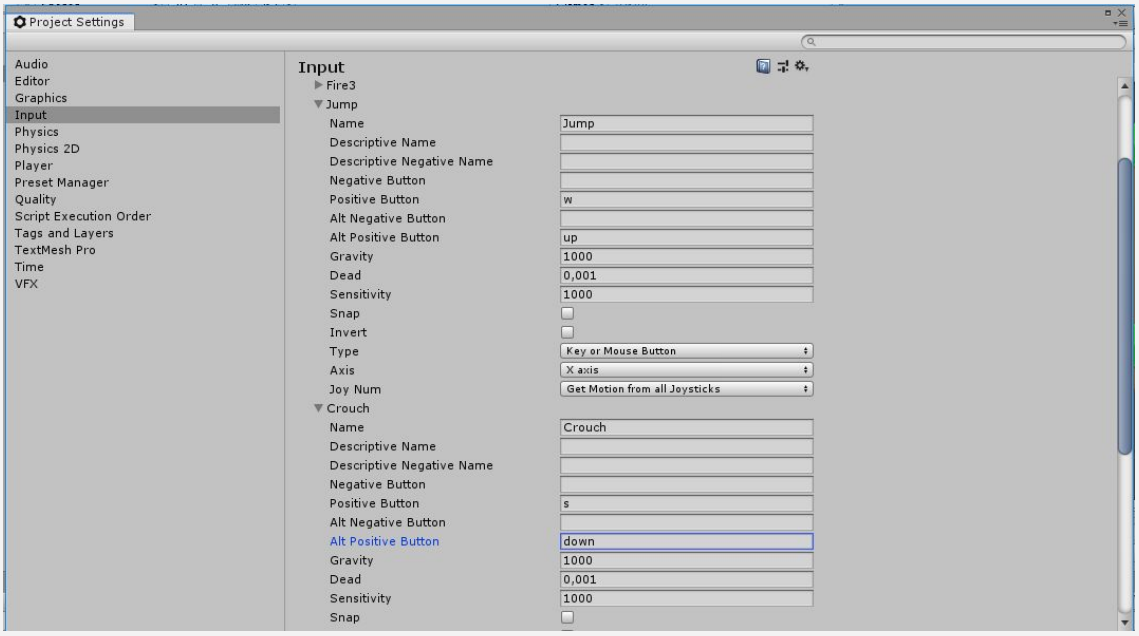
Lo siguiente que haremos será ver si el botón de salto está pulsado, que pongamos esa variable a *true*. Por suerte para nosotros *Unity* nos da el **Espacio** como botón de salto por defecto.

```
if (Input.GetButtonDown("Jump"))
{
    salto = true;
}
```

Lo último que queda para el salto es cambiar el segundo ***false*** del **FixedUpdate** por el valor de la variable *salto* y una vez el personaje salte, poner la variable a *false* de nuevo para que no pueda saltar hasta el infinito.

```
controller.Move(movimientoHorizontal * Time.fixedDeltaTime, false, salto);
salto = false;
```

Vamos a repetir el proceso para que nuestro personaje pueda agacharse en caso de que tengamos una parte del mapa a la que sea necesario entrar agachado. Esto también es una excusa para que veamos cómo podemos modificar los botones que nos da *Unity* por defecto en el caso en el que nosotros no queramos saltar con el **Space** sino con la **W.** Para hacer esto vamos a irnos a **Edit > Project Settings… > Input.** Una vez aquí, abriremos el primer **Jump** que veamos y el parámetro **Positive Button** lo sustituiremos por la **w** y en el parámetro **Alt Positive Button** pondremos **up** para referirnos a la flecha de dirección hacia arriba. Como se puede ver, **Crouch** no está así que lo creamos haciendo click derecho en **Jump** y seleccionamos **Duplicate Array Element.** Cambiamos el nombre del nuevo **Jump** que se nos ha creado a **Crouch** y ponemos **Positive Button** y **Alt Positive Button** a los valores **s** y **down** respectivamente:



Ahora podemos volver al código y hacer lo mismo que con *salto* pero con la diferencia de que sólo queremos parar de agacharnos cuando soltemos la tecla de agacharse.

```
if (Input.GetButtonDown("Crouch"))
{
    agachado = true;
}
else if (Input.GetButtonUp("Crouch"))
{
    agachado = false;
}
```

Y sustituimos el false que queda en la llamada del FixedUpdate por la variable agachado.

```
controller.Move(movimientoHorizontal * Time.fixedDeltaTime, agachado, salto);
```

Si ejecutamos, a primera vista no vemos si estamos agachados o no pero lo podemos notar porque la velocidad a la que vamos disminuye.

3.3- Animaciones