«Talento Tech»

Desarrollo de Videojuegos

Unity 2D

Clase 06

Buenos Aires Ciudad

Clase N° 06 | Invocando

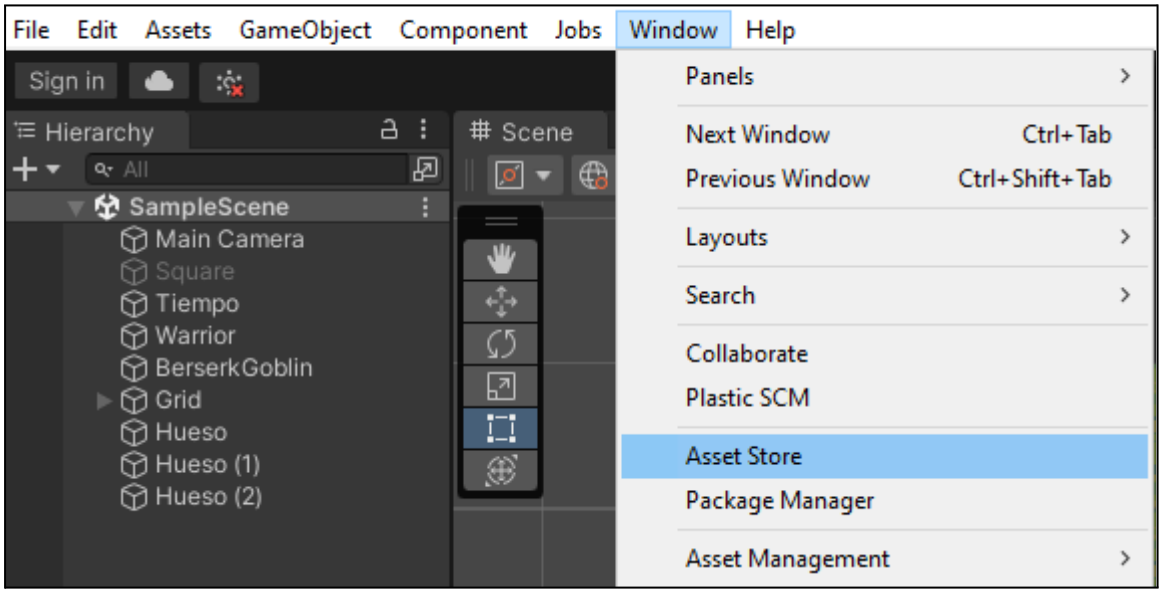
Temario:

- Instantiate
- Destroy
- Prefabs
- Package

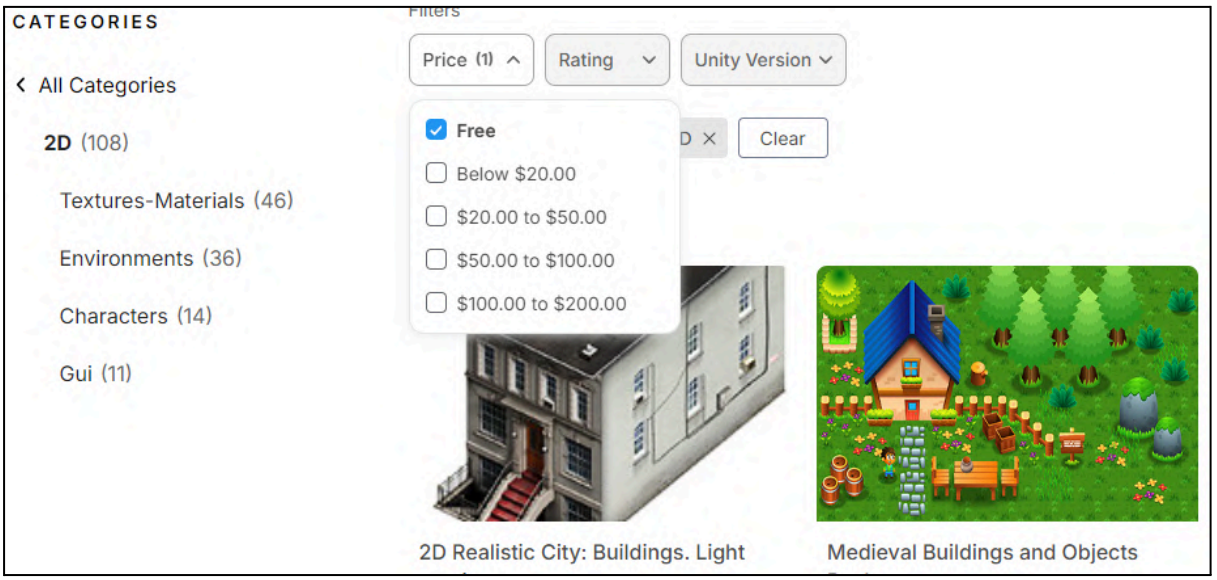
Descarga de Unity Packages.

Vamos a hablar sobre los *Unity Packages* (paquetes de Unity) que son un conjunto de assets que podemos armar y/o descargar para mover datos de nuestro juego con mayor facilidad, vamos a poder descargarlos y/o crearlos.

Para descargar, iremos a la solapa de “**Windows**” y seleccionaremos “**Asset Store**”



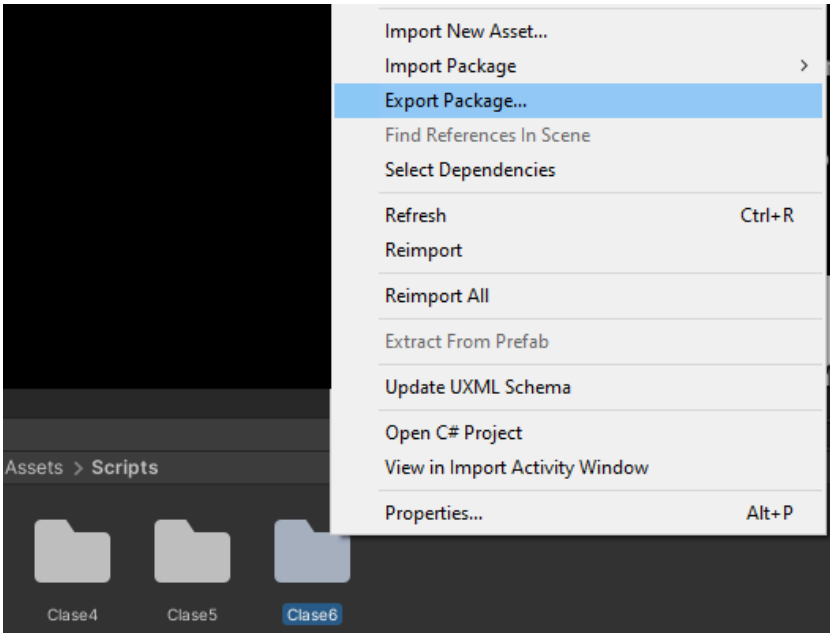
Nos llevará a una página. Desde la misma podremos configurar la búsqueda y descargar el Asset deseado.



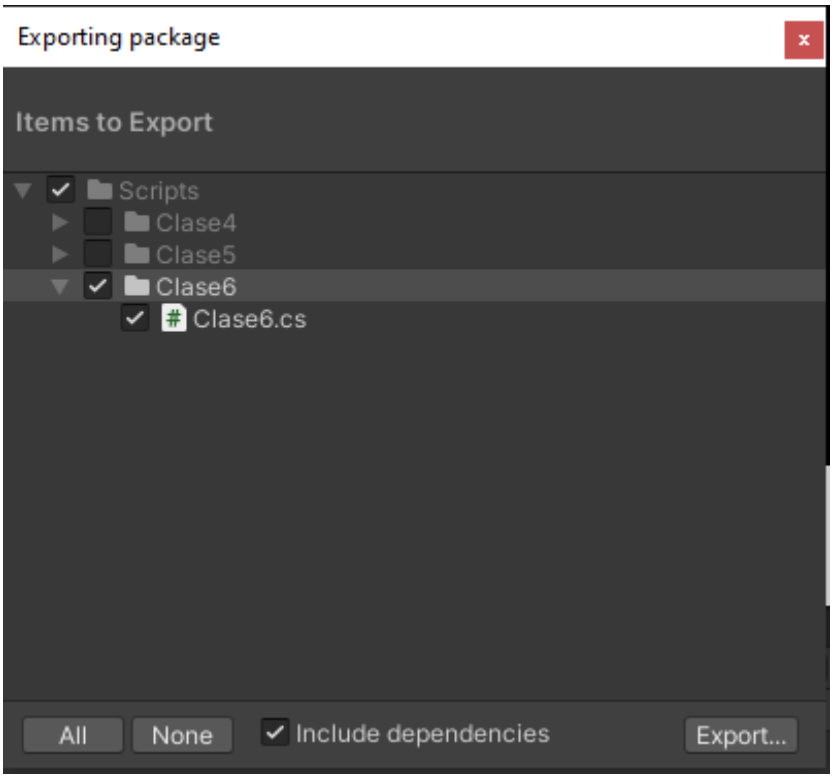
Creación de Unity Packages.

Como vimos, los paquetes son muy útiles para mover información de Unity, vamos a ver como crear nuestro propio paquetes, para compartir con otros usuarios y/o subir a algún lado.

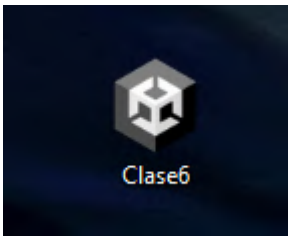
En nuestro proyecto tendremos carpetas, scripts, Sprites u otras cosas, que de seguro vamos a querer compartir, así sea por uso comercial o para trabajar en conjunto con otros/as.



Para hacer esto haremos clic derecho sobre la carpeta y luego clic sobre “Export Package…”



Nos aseguramos solo de exportar la carpeta deseada y seleccionamos “Export”:



Luego tenemos que elegir una carpeta de destino y ¡listo! Ya podemos pasarle nuestro paquete a alguien más ya sea por mail, drive o Discord si el paquete no es muy pesado.

Instantiate.

La **función *"Instantiate()"*** o "Instanciar", es una herramienta esencial en programación de videojuegos. Esta función nos **permite crear copias** de **objetos** en tiempo de ejecución, asignando una posición y rotación específicas.

Para utilizarla, la llamamos como cualquier otra función y le proporcionaremos tres **Argumentos** necesarios:

El objeto: Esto se logra mediante la creación de un **"prefab"** (abreviatura de objeto prefabricado). Un prefab es en esencia una plantilla del objeto que queremos instanciar, conoceremos más detalles sobre ellos más adelante.

La posición: Especificamos la **ubicación** en la que queremos colocar la instancia del objeto. Esto se logra proporcionando un **vector** de posición que indique los ejes **(X,Y,Z)**

La rotación: Similar a la posición pero definimos la rotación de la instancia mediante un vector de rotación.

Acá podemos ver un ejemplo de implementación:

```
    if (Input.GetKeyDown(KeyCode.F))

{

    //Instantiate(OBJETO, POSICIÓN, ROTACIÓN)

    Instantiate(obj, transform.position, Quaternion.identity);

}
```

Veamos…

¿Qué objetos podremos invocar? ¡El que queramos! Pero primero debemos crear una variable que sea de referencia.

```
   [SerializeField]

   private GameObject obj;
```

Y en este caso le asignaremos un dato desde el inspector. Podemos utilizar un objeto desde nuestra Hierarchy o usar un **Prefab(Que explicaremos más tarde).**

transform.Position:

Al usar transform (tengan cuidado de no poner Transform, con T mayúscula, porque llamarían a la Class y no al atributo) podremos acceder y/o modificar la Posición, Rotación o Escala. En este caso, pondremos la información de la posición, es decir, un Vector3.

Quaternion.identity:

El objeto está perfectamente alineado con el mundo o la orientación del objeto contenedor.

Ahora… ¿Qué es un Prefab?

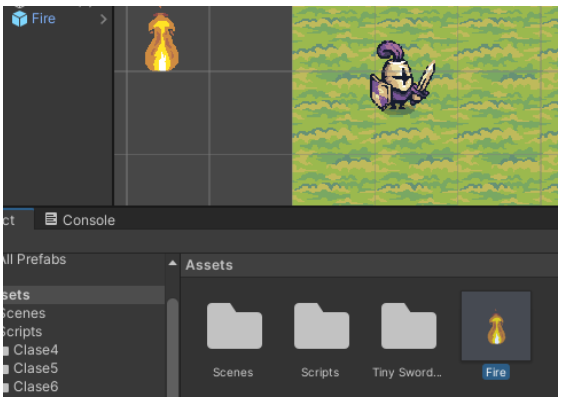
El prefab **actúa como una plantilla** a partir de la cual se pueden crear nuevas instancias del objeto en la escena. Cualquier **edición** realizada en un prefab asset se **reflejará** de inmediato en todas las **instancias** producidas a partir de él. No obstante, también es posible anular componentes y ajustes para cada instancia individualmente.

La forma más sencilla de crearlos es **arrastrar un objeto** de nuestra **jerarquía** a la ventana Project, idealmente dentro de una carpeta llamada “Prefabs” (recordá que podés crearla haciendo clic derecho en la ventana ***Project → Create → New Folder***).

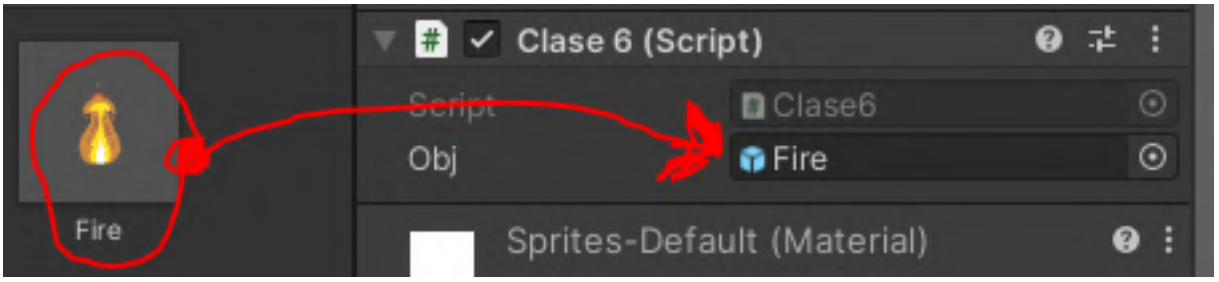
Esto cambiará el color del **nombre del objeto** en la jerarquía a **azul** o como vemos en el ejemplo, el ícono del objeto pasará de ser un **cubo** “vacío” a uno **“lleno”** , indicándonos que ya no es un objeto “común” y nos permitirá **crear copias** del mismo arrastrándolo desde nuestra carpeta a la misma jerarquía o directamente a la escena.

Antes:

Después:



Ahora que ya lo tenemos, podemos proceder a generar instancias o “copias” de nuestro objeto con nuestra función ***"Instantiate()"***. Solo debemos de arrastrar el Prefab a la variable dentro del inspector:



Destroy().

Seguiremos por el ***“Destroy()”***. Esta es una función que nos permite, como bien dice el nombre, destruir algún objeto o componente en particular.

```
private void OnTriggerEnter2D(Collider2D collision)
{
    Destroy(gameObject, 1f);
}
```

En esta imagen se usa el Destroy(), para indicar que cuando mi objeto entre en contacto con otro se destruirá.

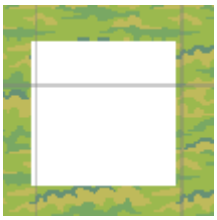
Si nos fijamos los **Argumentos** son dos:

-El objeto a destruir, este caso el nuestro.
-El tiempo en el que se destruirá. Para esto nos pide un **float**.

Ejemplo Práctico:

Crearemos un disparo sencillo, donde nuestra bala, al impactar, será destruida. Primero elegiremos un objeto que sea nuestra bala. O podemos agarrar un cuadrado, cambiarle el color y modificar su tamaño:

Antes:



Después:

Lo convertiremos en **Prefab** y empezaremos a trabajar el comportamiento de la bala.
Por ahora veremos 2 formas.

Primer método:

La primera forma que veremos es la de crear una referencia desde nuestro personaje y movilizar la bala desde ahí.

```
 public class Heroe : MonoBehaviour
{
    [SerializeField]
    private GameObject bullet;
      void Shoot(){
       if (Input.GetKeyDown(KeyCode.F))
       {
           GameObject b2 = Instantiate(bullet, transform.position,
Quaternion.identity);
           Rigidbody2D rbBullet = b2.GetComponent<Rigidbody2D>();
```

```
            rbBullet.velocity = Vector3.right * 10;
            Destroy(rbBullet, 2f);
        }

    }
```

Veamos…

Paso 1:
Creamos una variable del tipo “**GameObject**”. Esta será asignada con el **prefab** de la bala.

Paso 2:
Creamos la función **Shoot y el condicional** con la tecla deseada para disparar.

Paso 3:
Creamos una **variable** de referencia de tipo “**GameObject**”.

paso 4:
Le asignamos como valor, la instancia del objeto “**bullet**”.

Paso 5:
Creamos una variable del tipo **RigidBody2D** y le asignamos como valor el **RigidBody2D** de la instancia de “**bullet**” (si usan este método, acuerdense que deben colocarle el componente indicado al **prefab** de la bala).

Paso 6: Una vez hecho esto, accedemos al **velocity**, y lo modificamos.

Paso 7: Finalmente, le decimos que será **destruido** luego de un tiempo, específicamente 2 segundos. De esta manera podemos decidir el alcance de la bala, recuerden que:
Distancia = velocidad * tiempo.

Todo este código, hará que la bala instanciada se mueve a cierta velocidad y luego desaparezca. Claro que un profesor mío, siempre decía “**Hagamos que cada objeto sea responsable de su comportamiento**”. Por lo tanto, procederemos a la segunda forma.

Segundo método:

Esta forma se trata de crear un Script Propio para la bala:

```
[SerializeField]
private int speed = 10;

[SerializeField]
private float timeD = 2f;

private Rigidbody2D rb;

private void Start()
{
    rb = GetComponent<Rigidbody2D>();
    rb.velocity = Vector3.right * speed;
    Destroy(gameObject, timeD);
}
```

Veremos que es tan simple como asignar el **RigidBody2D**, darle velocidad y Destroy().

RECUERDEN: Pueden mover la bala de otras formas, usando **Addforce()** en el **FixedUpdate()** o usando el **Translate()** en el **Update()**, todo depende de lo que quieran generar.

Ejercicios prácticos:

1) **¡Te elijo a ti Pikachu!**
   Utilizando Instantiate, tu personaje debe poder invocar algún prefab. Este puede ser un objeto, ataque especial o criatura.

¡Recordá dotarlos de algún comportamiento básico para sumar a su utilidad!

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad