«Talento Tech»

Desarrollo de Videojuegos

Unity 2D

Clase 14

«Talento Tech»

Buenos Aires Ciudad

Clase N° 14 | SceneManager

Temario:

- SceneManager
- Cambio de Scene
- Permanencia de datos entre Scenes

SceneManager.

En Unity, un SceneManager es una clase que permite administrar las escenas en un proyecto. Una escena en Unity es un entorno que contiene objetos, configuraciones, y componentes que forman parte de un juego o aplicación, como terrenos, personajes, cámaras, luces y más.

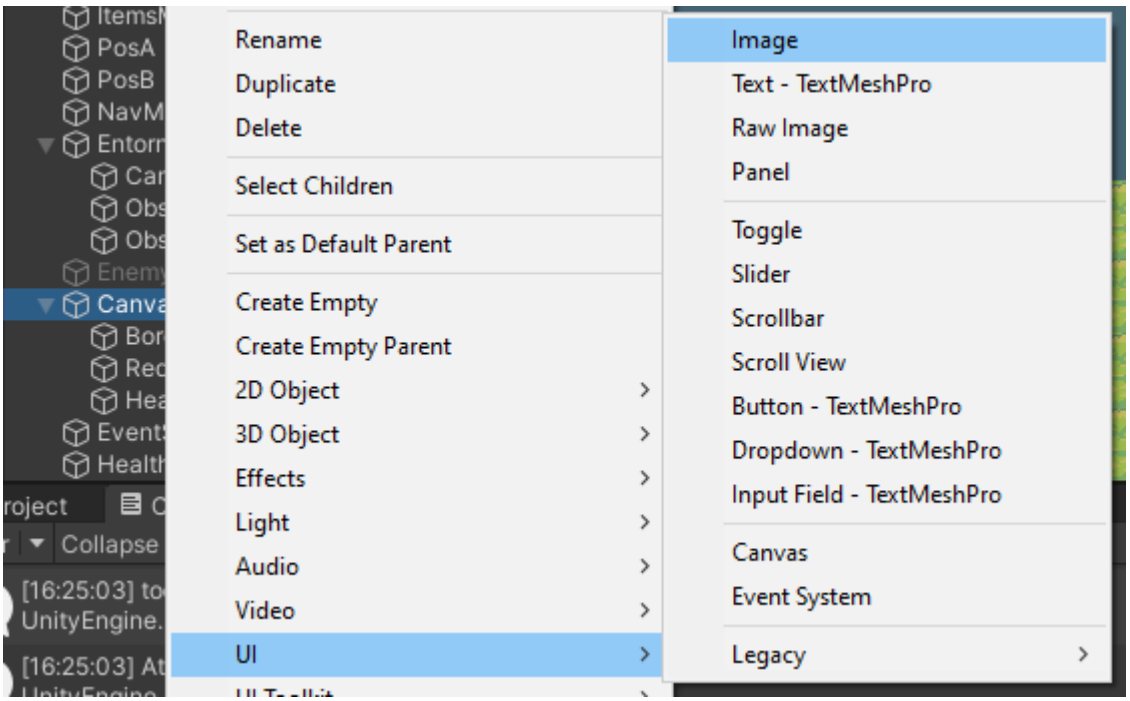
El SceneManager es parte de la biblioteca **UnityEngine.SceneManagement** y proporciona funciones para cargar, descargar, cambiar y gestionar escenas de manera eficiente durante la ejecución del juego.

Cambio de Scene (Menú Inicial).

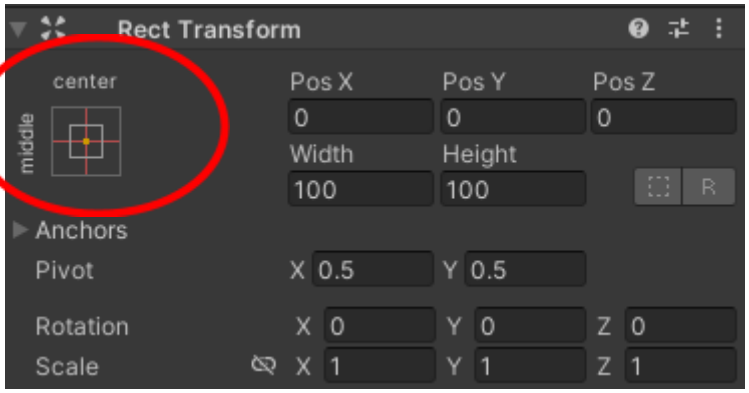
Para empezar a trabajar con esta idea, iremos creando un menú inicial, haciendo que al apretar un botón inicie el juego.

El fondo.

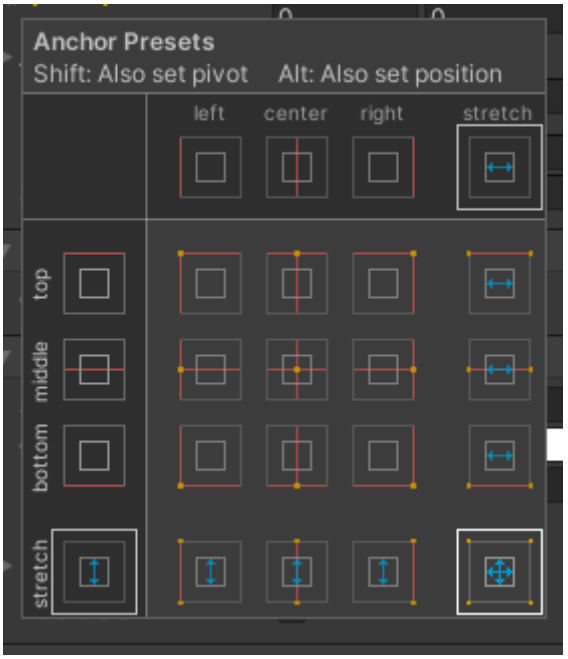
Lo primero que vamos a hacer es crear un fondo para nuestro menú. Para esto, y todo lo que sigue, vamos a estar usando elementos que están dentro de la pestaña de UI, como cuando hicimos el *Canvas*. Vamos a crear un nuevo objeto dentro del *Canvas* de tipo *Image,* una vez creado vamos a llamarlo “Fondo”:



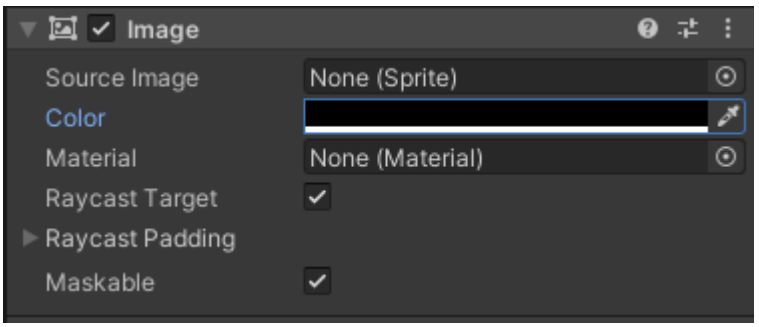
Necesitamos que esta imagen ocupe todo el espacio del *Canvas* para que cumpla su función de fondo. Vamos a seleccionar el fondo, y vamos a notar que el *transform* es distinto a otros objetos. La lógica de posicionamiento de los objetos UI es distinta a la convencional:



Podemos cambiar el ancho, el alto y la posición relativa al objeto padre (en este caso el *Canvas*. Para acelerar el proceso, vamos a aprender cómo rellenar un objeto de forma rápida: Vamos a hacer clic sobre el cuadro de posiciones (marcado con un círculo en la imagen de arriba), manteniendo la tecla Alt , vamos a seleccionar el botón que está en la esquina inferior derecha.



Ahora vamos a ver que tenemos todo el *Canvas* cubierto. Vamos a cambiar el color del fondo desde el componente *Image*.

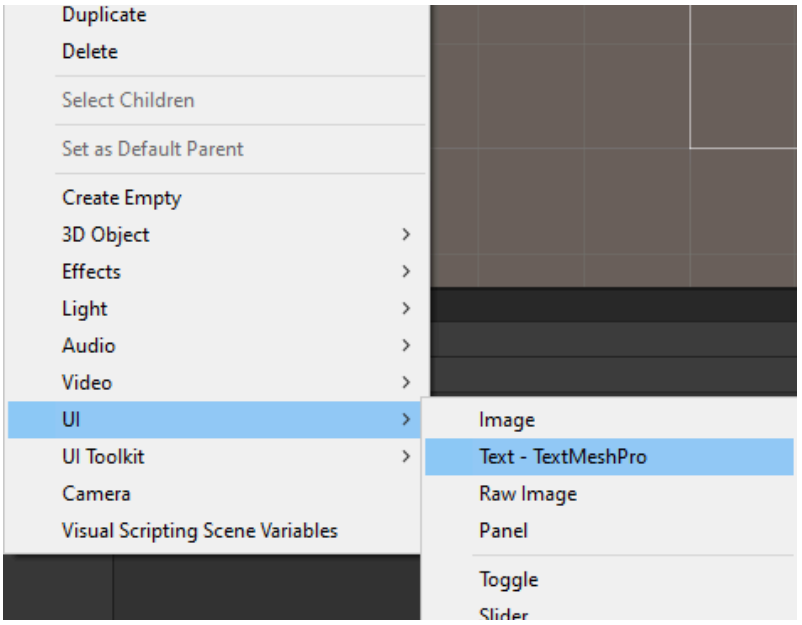


Lo ideal sería ya comenzar a crear los textos. Para hacerlo, podemos tener en cuenta como lo hicimos con el Canvas, ya que es muy similar.

El Texto.

Para escribir texto, vamos a utilizar TextMeshPro, un componente de texto muy flexible, que nos permite realizar un montón de cosas que tenga que ver con el contexto.

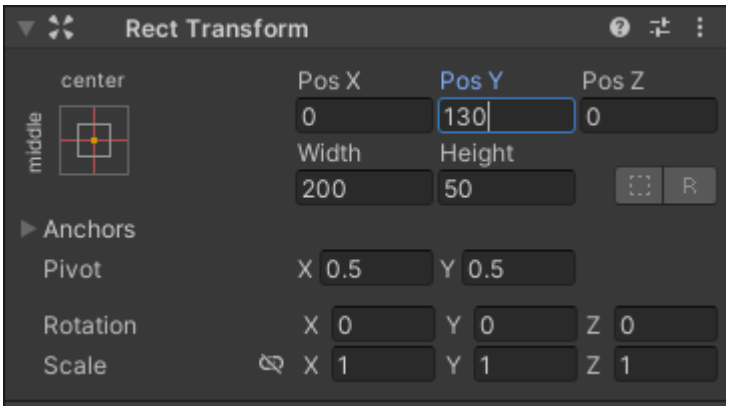
Haciendo clic derecho sobre el *Canvas*, vamos a ir *UI -> TextMeshPro*.



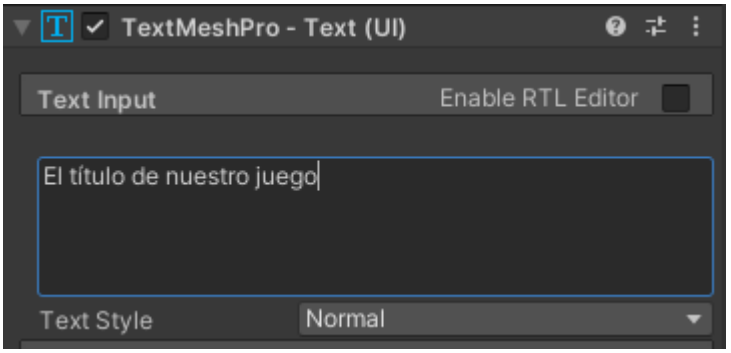
Al objeto creado le vamos a renombrar como "**Título**".

En el componente de **TextMeshPro** de nuestro nuevo objeto, vamos a cambiar el alineamiento para que quede prolijo. Y luego vamos a cambiar la posición en el eje Y de forma positiva para que quede más arriba el texto.

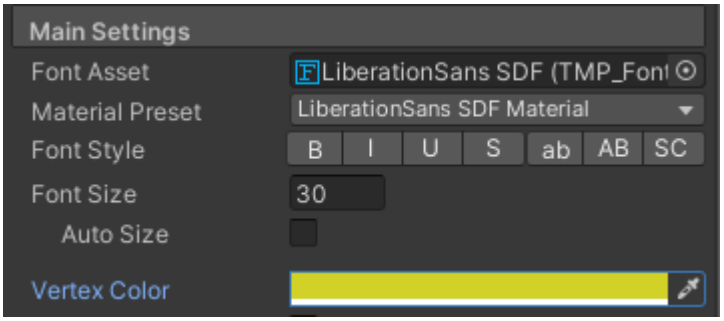
Alineamiento



Vamos a ver un campo de texto donde podemos escribir lo que va a mostrarse. Por Default siempre dice *"New Text"*. Vamos a escribir el nombre de nuestro juego (o lo que queramos).

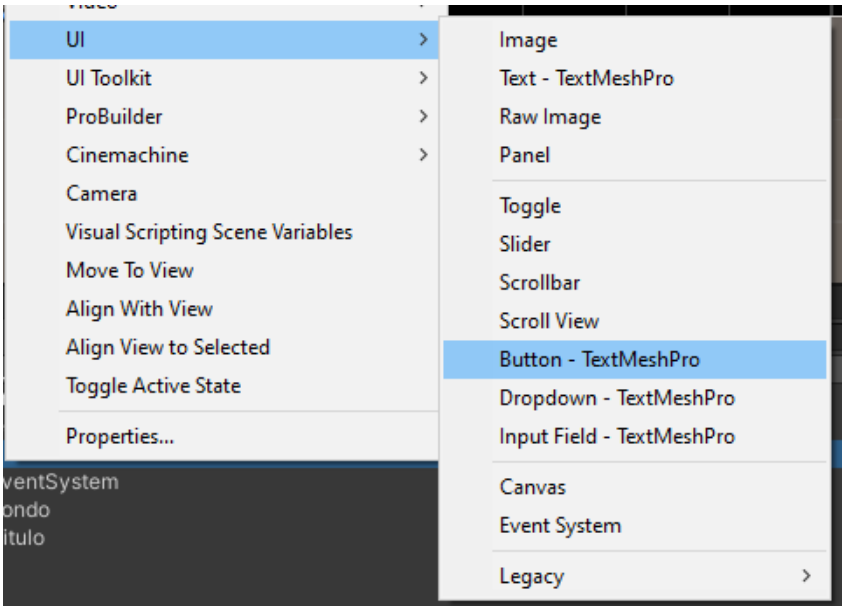


También podemos cambiar el tamaño de las letras con *Font Size* y el color de las letras con *Vertex Color*:

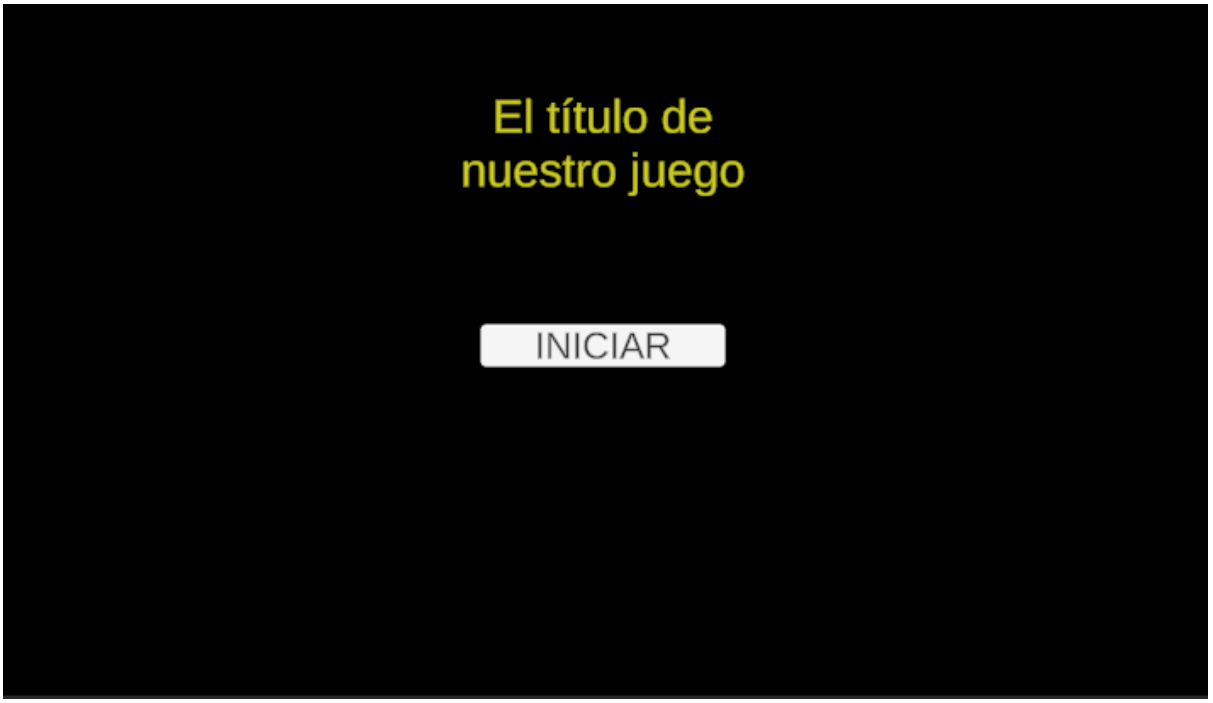


Botones.

Ahora vamos a crear un botón que nos va a servir para cambiar de escena. Vamos a hacer clic derecho sobre el *Canvas* -> UI -> Button - TextMeshPro:



Al objeto creado lo vamos a llamar "**Boton_jugar**" y vamos a notar que este objeto ya trae un texto como hijo. Podemos acceder al "**Text (TMP)**" para cambiarle el texto al botón, como hicimos con el título; podemos escribirle algo que indique que ese botón sirve para iniciar el juego.



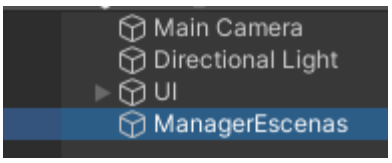
Así quedó nuestro menú

Ahora vamos a escribir un poco de código para que podamos cambiar de escena cuando apretamos nuestro botón.

Implementación de Scripts.

Primero vamos a crear un objeto que va a controlar el cambio de escenas. Este va a ser un objeto invisible dentro de nuestro juego, que va a administrar el código necesario para que podamos pasar de una escena a la otra. Estos objetos se conocen como **Managers**, son muy importantes, ya que se encargan de manejar distintos aspectos de nuestro juego. Cabe aclarar que un manager no es obligatorio, es solo un nombre que se usa por convención, pero puede llamarse de otra forma.

Creamos un nuevo objeto vacío, y le ponemos “ManagerEscenas”:



Ahora vamos a crear un nuevo C# script y lo vamos a llamar de la misma manera “ManagerEscenas”.

Dentro del script vamos a escribir el siguiente código:

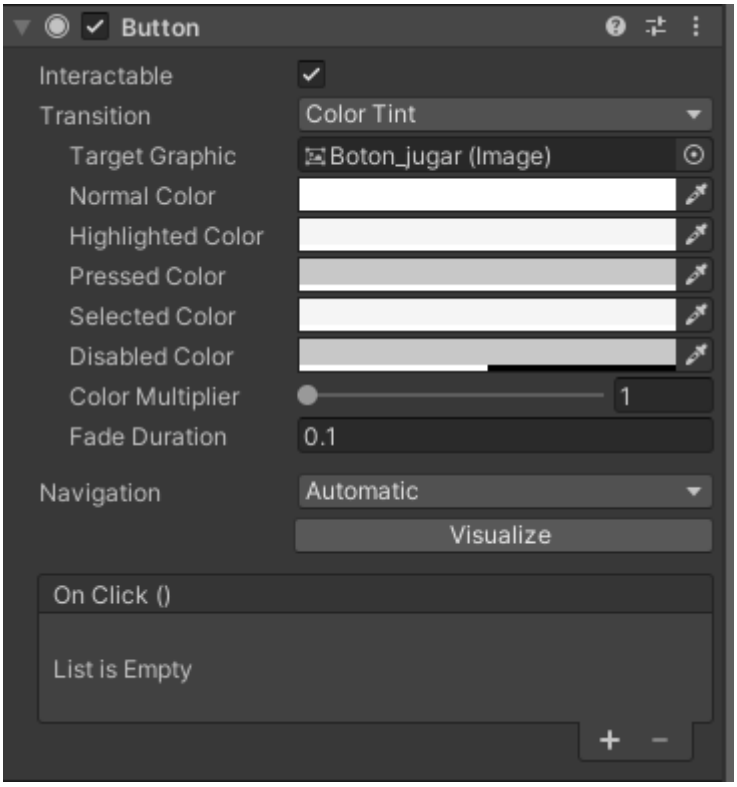
```
using UnityEngine.SceneManagement;
public class ManagerEscenas : MonoBehaviour{
  public void CambiarEscenas(string a){
   SceneManager.LoadScene(a);
  }
```

Cosas a notar en este script: Vamos a ver que agregamos una biblioteca extra. Sin esta línea no vamos a poder acceder a *SceneManager*. Usamos un parámetro llamado “a” que vamos a utilizar para saber a qué escena vamos a cambiar, ya que en general, tenemos más de dos escenas en un proyecto.

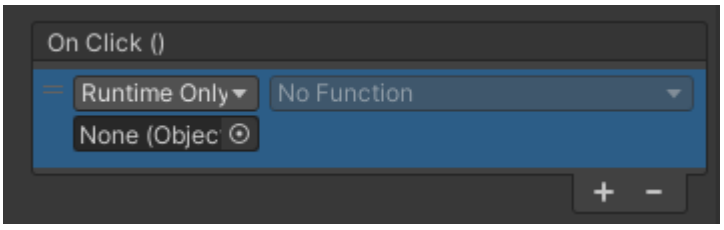
Con este script vamos a estar bien, pero nos queda una serie de pasos para que este botón funcione:

Vamos a agregar este nuevo script al objeto “ManagerEscenas”:

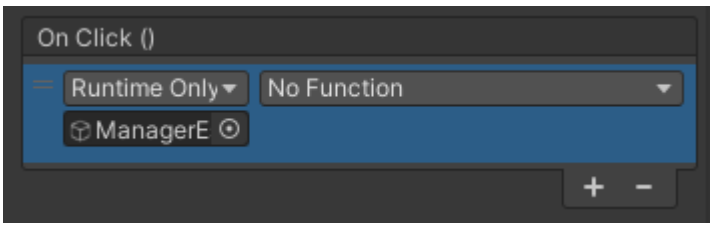
Vamos a seleccionar nuestro botón en la *Hierarchy* y vamos a buscar el componente de *Button*:



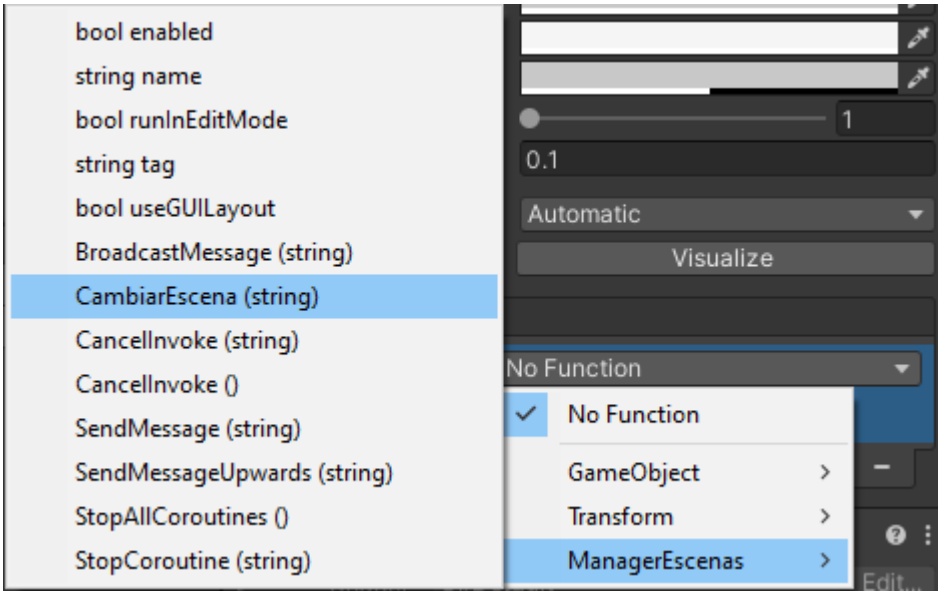
En Unity, cuando apretamos un botón, este llama un evento que hace que suceda algo. Estos eventos aparecen en la lista de abajo donde dice “On Click ()”.
Como el botón es nuevo y no le asignamos ningún evento, está vacío. Nuestro objetivo es hacer que cuando lo apretemos, se reproduzca el código que escribimos anteriormente. Vamos a hacer clic sobre el símbolo del “+” y nos va a aparecer un campo vacío donde vamos a agregar el objeto que aloja el código.



Vamos a agarrar el objeto (no el script) de “ManagerEscenas” y lo vamos a arrastrar hasta donde dice “None”, tiene que quedar así:



Una vez hecho esto, vamos a abrir el *dropdown* que dice "*No Function*" -> MangerEscenas -> CambiarEscena(string)

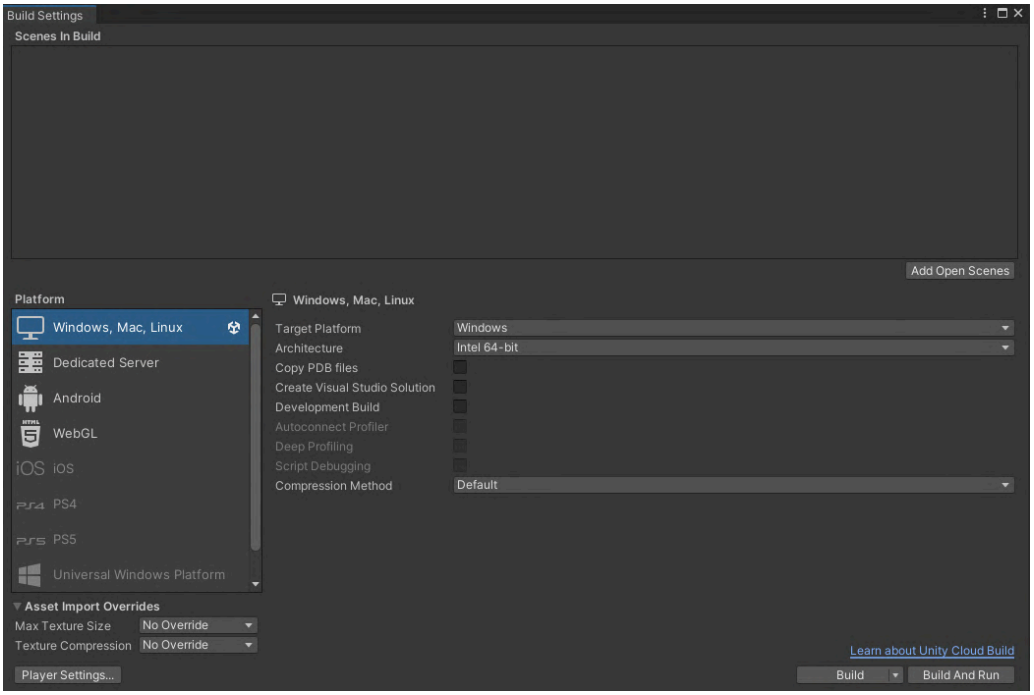


Acá lo que estamos haciendo es seleccionar la función que creamos anteriormente, y el espacio para escribir (un string) es el parámetro "a" que mencionamos antes. En este campo vamos a escribir el nombre de la escena que vamos a conectar con el botón.

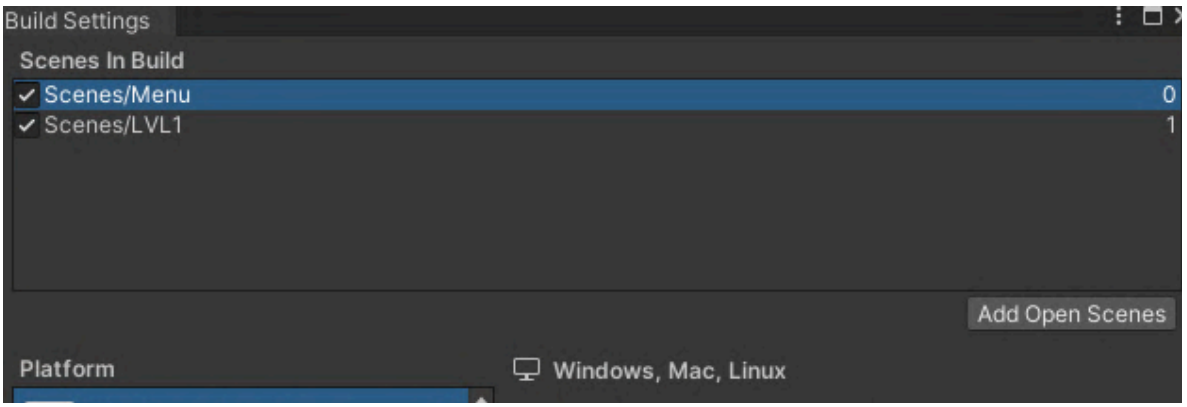
Ya casi estamos, falta un paso más para que nuestro botón funcione.

Build Settings.

Para que las escenas estén conectadas, tenemos que agregarlas a la build del proyecto. vamos a ir a File -> Build Settings para que nos aparezca la siguiente ventana.



En la lista de arriba vamos a agregar las escenas que queremos conectar. Vamos a agarrar las escenas y las vamos a arrastrar hasta la lista.



Con esto hecho, vamos a probar el botón y si sale todo bien, vamos a tener un botón que pasa de escena.

Pase de datos entre Escenas.

¿Qué pasaría si quisiéramos pasar datos de una escena a otra?

Por ahora presentaremos 2 formas

Variable Static

Como ya vimos en la clase 12, podemos crear variable “**static**”, que lo que hacen es contener un dato “inamovible”, siendo siempre el mismo entre todas las **Instancias** de la **Class**. Por lo tanto bastaría con crear una variable, por ejemplo llamada “Score”(Puntaje), hacerla Static y simplemente utilizarla desde la Class.

En este caso, crearemos un Script llamado “GameManager” y le crearemos la variable “Score”

```
public class GameManager : MonoBehaviour
{
   public static int Score;
```

Y ahora utilizaremos el dato desde cualquier otro lugar que queramos. Recuerden que no hace falta instanciar la Class en algún objeto en la Scene.

```
private int score;
   void Start()
   {
      GameManager.Score = 2825;
      score = GameManager.Score;
```

Ya no importará en qué Scene estemos, podremos acceder a la variable GameManager.Score, desde cualquier lado (*¿muy peligroso?*).

DontDestroyOnLoad.

DontDestroyOnLoad es una función que evita ser destruido al objeto que la contenga.
Esto nos permitirá mantener Objetos como mi “ManagerEscenas” o un “GameManager” entre los distintos cambios de nivel

Primero setiamos la forma de cambiar el **score**

```
[SerializeField] private int _score;
public int score{
 get { return _score; }
 private set{
  //pregunto si el dato es un int
  if (value.GetType() == typeof(int)){
     _score = value;
  }
  else{
    throw new System.ArgumentException("score debe ser int");
```

```
    }
  }
}
void ChangeScore(int a)
  {
    score += 5;
  }
```

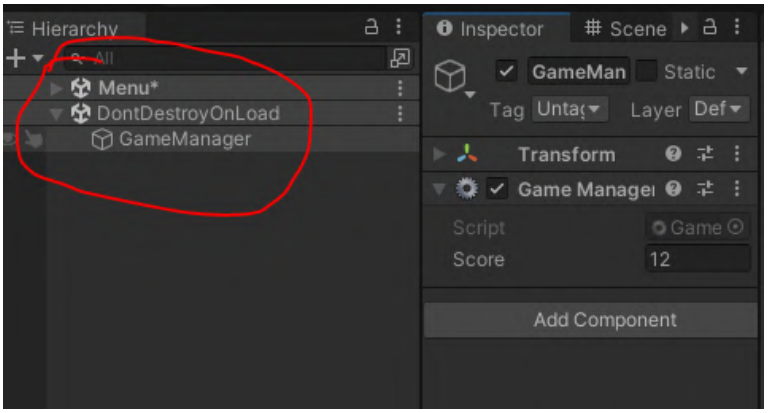
Ahora usaremos la función “Awake” y llamaremos al DontDestroyOnLoad().

```
    private void Awake()
  {
    DontDestroyOnLoad(gameObject);
  }
```

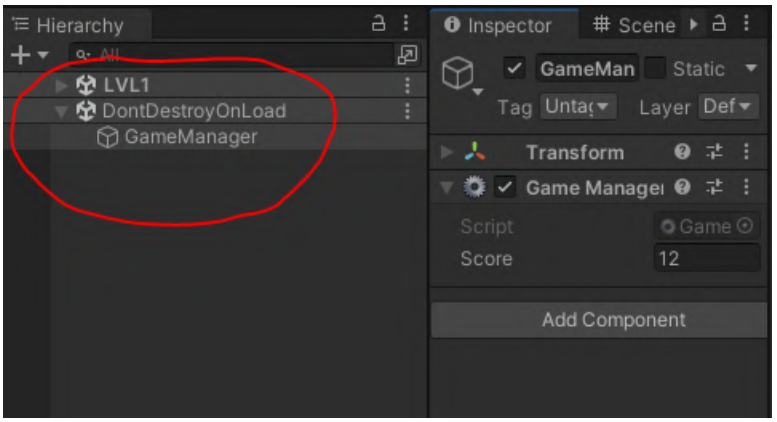
*La función **Awake()** en Unity es un método especial que se llama **una vez** en el ciclo de vida de un GameObject cuando se carga en la escena, justo antes de que el juego comience a ejecutarse. Es una de las funciones de inicialización que Unity ofrece y pertenece al ciclo de vida de los scripts.*

Una vez hecho esto, crearemos un emptyObject llamado “GameManager” en nuestro menú y le agregaremos nuestro Script. Veremos que al cambiar la escena, nuestro GameManager sigue estando.

Objeto GameManager en el Menú



Objeto GameManager en la escena LVL1



Podrán notar que el objeto no desaparece y conserva el valor de la variable.

Ejercicios prácticos:

DontDestroyMisObjetitos:

1) Realizar el menú de su juego, utilizando un “SceneManager” o “ManagerEscenas”.

2) Crear un GameManager con DontDestroyOnLoad, que tendrá aquellos datos que crean pertinentes conservar entre niveles/escenas.

3) Crear pantalla de Ganar o Perder según lo amerite el juego


Buenos Aires
aprende
Agencia de Habilidades para el Futuro
Buenos
Aires
Ciudad