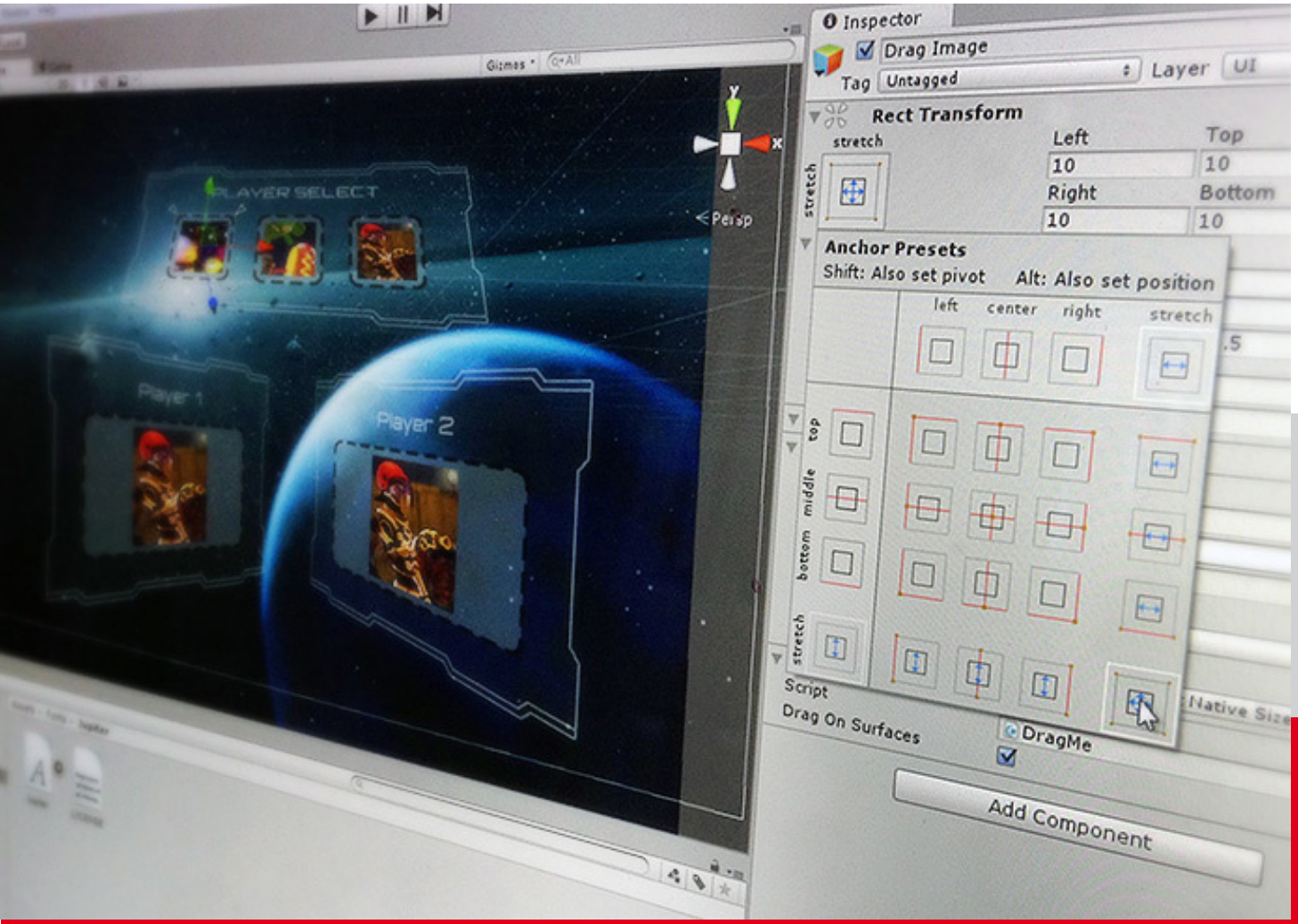
Álvaro Holguera
CIFP José Luis Garci
(Madrid)

UD 04
USER INTERFACE

Desarrollo de Entornos
Interactivos Multidispositivo

Álvaro Holguera
CIFP José Luis Garci
(Madrid)



CREANDO MENÚS & HUDS

En este unidad aprenderemos a crear menús de navegación y elementos gráficos para nuestros videojuegos

USER INTERFACE

La UI (User Interface) o "Interfaz de Usuario" es la forma que tiene el juego de comunicarse con el jugador, darle información (como estado, vida, mensajes, etc.), y también permite recabar información a partir de botones, campos de texto, etc.

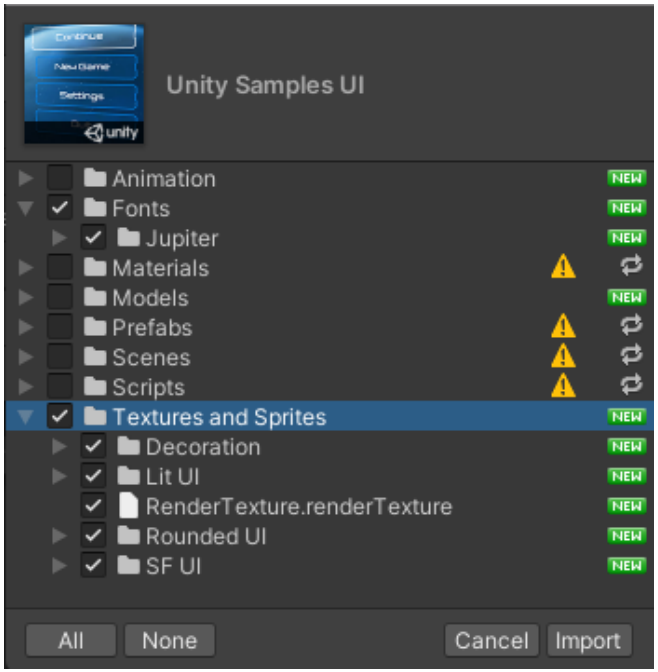
Puedes encontrar más información en:

https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html

Para los siguientes ejercicios, puedes usar los assets que ofrece Unity de forma gratuita: https://assetstore.unity.com/packages/essentials/ui-samples-25468

Para importarlos, añádelos a tu cuenta de Unity y descárgalos, ya sea desde la web o en la herramienta de Window>Package Manager (seleccionando Packages: My Assets). Una vez descargados, podrás importarlos a tu proyecto.

IMPORTANTE: cuando importamos unos assets descargados a través de la Asset Store, podemos seleccionar qué queremos importar. En este caso, importaremos solo las fuentes y los sprites. También debemos asegurarnos de que lo que importamos es compatible con la versión del editor en la que estamos trabajando (especialmente con los scripts) y que no sobreescribe nada de nuestro proyecto.



Librerías adicionales

Para utilizar las funcionalidades que vamos a ver a continuación, y por primera vez, tendremos que importar librerías adicionales a las que vienen por defecto.

Gestión de escenas

Para gestionar estenas mediante código, deberemos importar la librería correspondiente.

Añade esta línea de código al comienzo de tu script, junto a las otras librerías, o no podrás acceder a las variables y las clases necesarias:

```
using UnityEngine.SceneManagement;
```

Acceso a los elementos de la UI

Por su lado, tendremos que usar otra librería para poder acceder a los elementos de la User Interface, como textos, imagenes, etc.

```
using UnityEngine.UI;
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
//Librería para gestión de escenas
using UnityEngine.SceneManagement;
//Librería para gestión de la UI
using UnityEngine.UI;
```

En esta imagen puedes ver cómo quedarían las dos librerías importadas

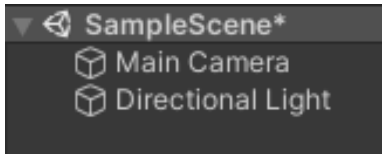
GESTIONANDO ESCENAS

Antes de ver cómo crear menús en nuestro juego, y ayudas al usuario mediante HUD's, entendamos cómo se construyen las escenas y cómo se gestionan.

Una escena es lo que contiene todos los elementos de nuestro juego (personajes, terrenos, menús...). Es como si cada escena fuese un nivel

Podemos crear tantas escenas como queramos (Ctrl. + n) y guardarlas en la carpeta del proyecto que queramos (es importante guardar la escena una vez creada y cada vez que realicemos cambios).

Unity por defecto crear una escena al crear un proyecto nuevo, que aparecerá como "Sample Scene" y que podemos verla en nuestra ventana de jerarquía.



NOTA: el asterisco indica que está sin guardar, algo peligroso.

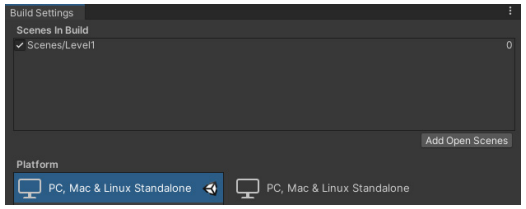
TRUCO: Es posible incluso editar varias escenas a la vez, mediante la herramienta "Multi-Scene editing", pulsando con el botón derecho en una escena del proyecto y pulsar en "Open Scene Additive"

Añadir las escenas a la compilación final

Todas las escenas que creemos deberán ser añadidas en la compilación del proyecto (la llamada "Build"), si no, no aparecerán en el juego final.

Para añadirlas, debemos ir a File>Build Settings y aparecerá la ventana de configuración, y en la parte superior las escenas añadidas al juego.

Se puede añadir la escena abierta en ese momento, o arrastrarlas directamente desde la ventana del proyecto



Veremos que se les asigna automáticamente un número de índice a la derecha, comenzando por 0. Este nº sirve para identificar a cada escena en el código, además de por su nombre, muy útil si queremos cambiarle el nombre a la escena en algún momento

Gestionar la escena mediante código

Ahora que tenemos incorporada la librería SceneManagement podremos, entre otras cosas, tener acceso al método "SceneManager", que te permite por ejemplo cambiar de escena en el juego mediante LoadScene() al que debemos pasar el nombre de la escena o el número de índice el BuildSettings. Aquí tienes un ejemplo:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneManagement : MonoBehaviour
{
    void LoadSceneName()
    {
        //Cargamos la escena indicando el nombre
        SceneManager.LoadScene("Level1");
    }

    void LoadSceneNumber()
    {
        //Cargamos la escena indicando el número
        SceneManager.LoadScene(1);
    }
}
```

Puedes ver la librería cargada y dos funciones que cargan la escena de dos formas distintas. No hace falta decir que la escena se la podemos pasar en una variable a la función, así una misma nos valdría para cargar cualquier escena. Como siempre, revisa la documentación

Algo muy práctico es obtener el índice de la escena en la que estamos (que es un número entero) mediante GetActiveScene().buildIndex y de esa forma podemos indicar que cargue siempre la escena siguiente. Aquí tienes el ejemplo:

```
void LoadNextScene()
{
    int sceneID = SceneManager.GetActiveScene().buildIndex;
    int nextScene = sceneID + 1;
    SceneManager.LoadScene(nextScene);

}
```

CREANDO UN LOADER

Aunque esto es "para nota", vamos a ver cómo se crea la tipica escena de carga entre un nivel y otro.

Cargando escenas de forma asíncrona

El problema del método LoadScene es que cierra la escena actual y comienza la carga de la siguiente. Si es ligera, el tiempo de espera es corto, pero si la siguiente escena tiene un tiempo de carga largo la pantalla aparecerá en negro.

La solución, cargar la siguiente escena de forma asíncrona sin salir de la actual, mediante LoadSceneAsync: a diferencia de LoadScene, este método carga la escena en segundo plano y avisa cuándo se ha terminado de cargar. Esto es muy útil para crear escenas de carga o "loaders"

Si queremos obtener más datos o realizar más acciones sobre la escena cargándose, deberemos usar el método AsyncOperation, que entre otras cosas nos puede dar el % completado del progreso carga (muy útil si lo vinculamos a una barra de estado mediante sliders, que veremos más tarde).

En el siguiente código se carga una escena de forma asíncrona mediante una corrutina. Muy útil para pantallas de "loading":

```
void iniciarCarga()
{
    //Iniciamos la corrutina que gestiona el loader
    StartCoroutine(LoadYourAsyncScene());
}

IEnumerator LoadYourAsyncScene()
{

    AsyncOperation asyncLoad = SceneManager.LoadSceneAsync("Scene2");

    //Esperamos a que la escena se haya cargado
    while (!asyncLoad.isDone)
    {
        //Variable con el progreso en %
        float progress = asyncLoad.progress * 100;
        print("Porcentaje cargado: " + progress);

        //Si la carga ha superado el 99% cambiamos de escena
        if (asyncLoad.progress >= 0.9f)
        {
            //Permitimos que se active la escena cargada
            asyncLoad.allowSceneActivation = true;
        }
        //Mientras se está cargando la corrutina se ejecuta cada fotograma
        yield return null;
    }
}
```

Una vez que tenemos el progreso, es fácil trasladarlo a un texto o a un slider en la pantalla.

PARA NOTA: MoveGameObjectToScene permite mover el GameObject indicado a la escena de destino. Solo funciona con GameObjects del raíz (no se puede mover hijos), y la escena tiene que estar cargada en el background. También podemos acudir al método DontDestroyOnLoad(), que permite indicar qué un objeto no sea destruido al cargar la nueva escena, algo muy útil para conseguir que la música siga sonando entre una escena y otra.

NOTA: si lo que queremos es salir del juego, debemos usar el método Application.Quit(); que cierra el programa (no lo podremos probar hasta compilar el juego ya que en la previsualización no funciona).

VARIABLES ENTRE ESCENAS

EN POO vimos las variables estáticas, pero algo que tenemos que conocer es que mantienen su valor en todas las instancias de la clase, incluso al pasar de una escena a otra, algo realmente util, si no necesario

Game Manager

Algo muy útil, es crear una clase que contenga las variables que vamos a necesitar a lo largo de todas las escenas, como vidas, dificultad, datos de configuración del juego introducidos por el usuario (como nombre, volumen de la música, dificultad, etc.)

Estos atributos de la clase, como son estáticos, pueden ser llamados en cualquier momento, sin necesidad de instanciar la clase no agregarla como componente a un objeto de la escena

Por ejemplo, si creamos una clase llamada "GameManager", en la que añadimos variables públicas estáticas como la energía del usuario en el atributo "playerEnergy", ó "lives", podemos llamarla desde cualquier script como atributo de la clase

Curiosidad: si creamos un script llamado "GameManager", Unity le cambiará automáticamente el Icono

Veamos ejemplos de uso:

1. Creamos nuestro script llamado GameManager, en el que añadimos las variables estáticas que consideremos oportunas. Como el número de vidas del jugador, o el volumen de la música configurado por el usuario.

2. En una función, que podemos cargar al pulsar un botón de inicio de juego, ponemos las vidas a 3 y lanzamos el primer nivel.

3. Tenemos otra función que se lanza cuando chocamos con un obstáculo. Al hacerlo, resta una vida y comprueba si nos hemos quedado a cero, si es así, carga la escena de inicio, si no, vuelve a cargar la misma escena, obteniendo su índice (así este método nos vale para cualquier escena). Eso sí, el número de vidas ahora es uno menos, y ese valor se mantiene entre escenas.

IMPORTANTE: las variables estáticas deben ser públicas para poder acceder a ellas desde otros scripts, y no es necesario acceder a un componente que tenga el script asociado.

```
void Chocar()
{
    GameManager.lives--;
    if(GameManager.lives == 0)
    {
        SceneManager.LoadScene("MenuInicial");
    }
    else
    {
        int currentScene = SceneManager.GetActiveScene().buildIndex;
        SceneManager.LoadScene(currentScene);
    }
}
```

1

```
public class GameManager : MonoBehaviour
{
    public static int lives;
    public static float musicVolume;

}
```

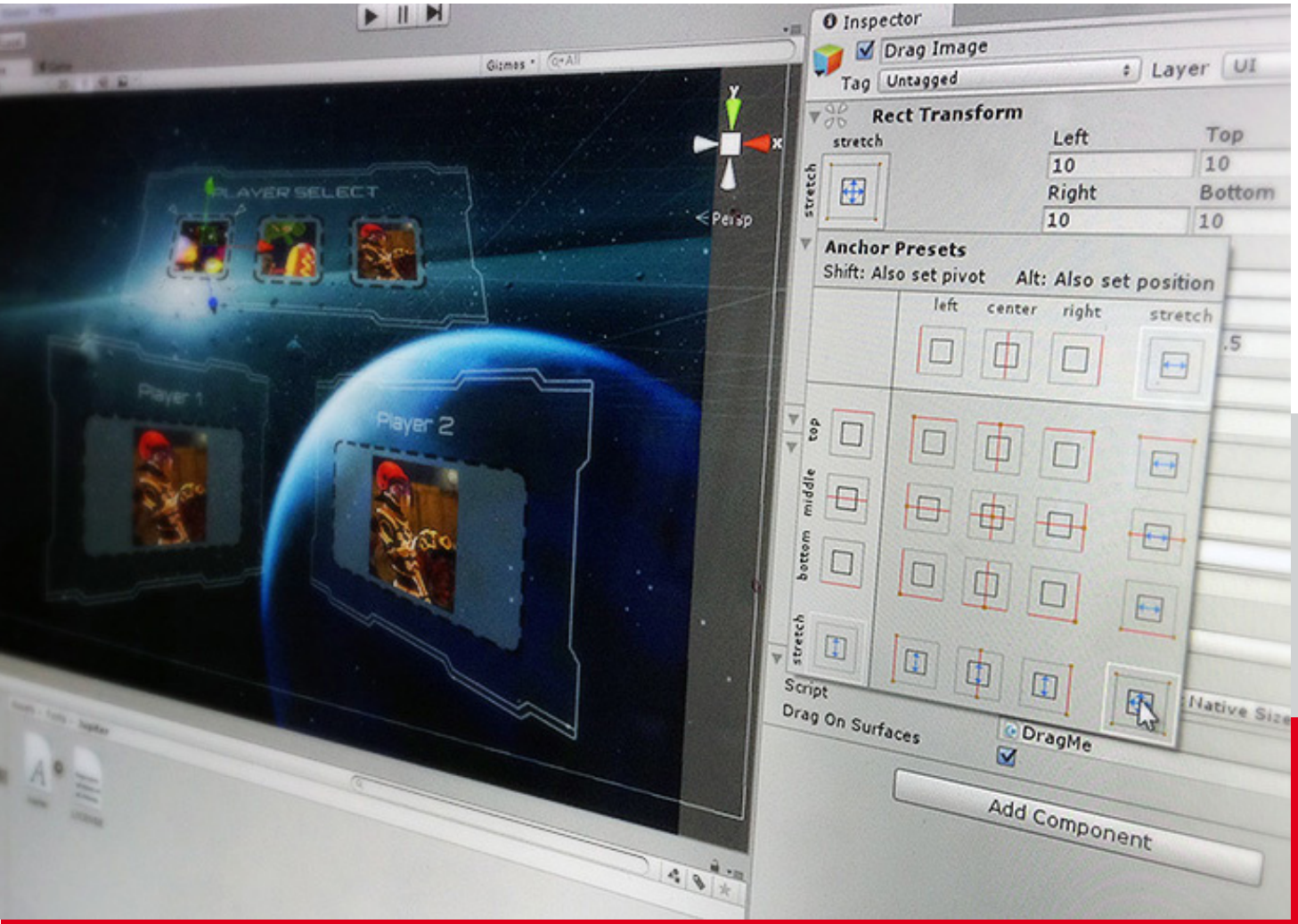
2

```
private void IniciarJuego()
{
    GameManager.lives = 3;
    SceneManager.LoadScene("Level1");
}
```

3

PARA NOTA: si queremos crear una clase que se instancie solo una vez y sea accesible globalmente, debemos crear los denominados "singletons".

Álvaro Holguera
CIFP José Luis Garci
(Madrid)

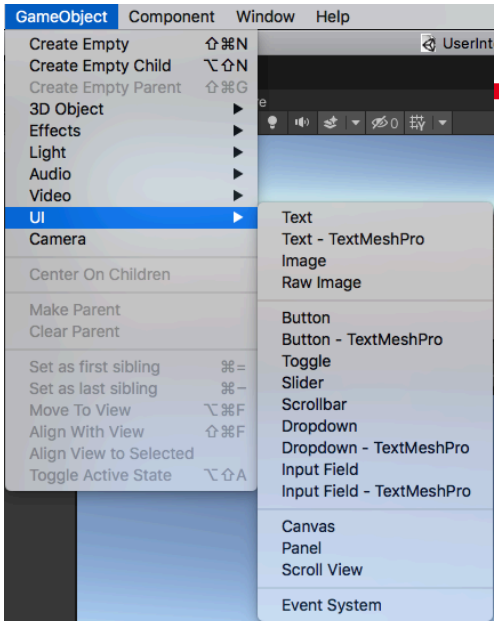


CANVAS

Antes de añadir los elementos de la UI, veamos cómo se organizan dentro de un “lienzo” o “canvas”.

CREANDO UN CANVAS

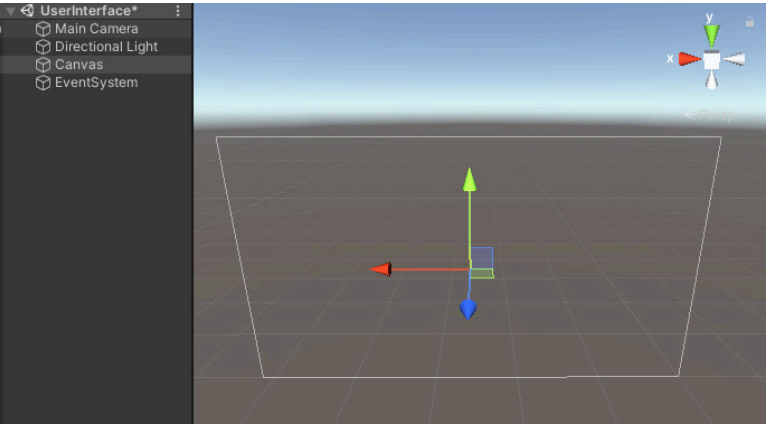
Ahora que sabemos gestionar las escenas, vamos a gestionar nuestra UI. En el menú Game Object > UI tenemos todos los elementos disponibles



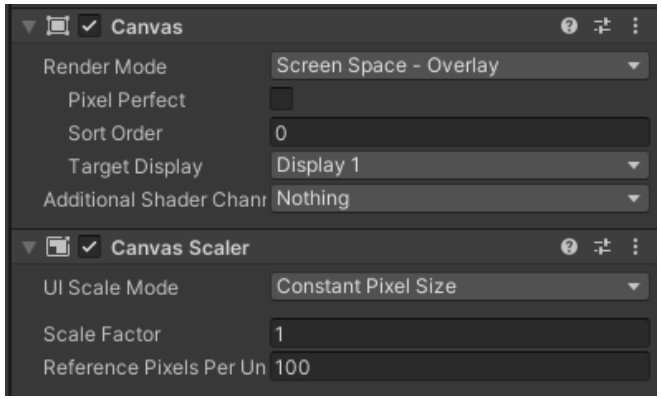
Canvas

Todos los elementos de un UI tienen que ser hijos de un elemento padre de tipo canvas, el cual podemos crear vacío, que dibujará un cuadrado en el escenario. Al insertar un elemento de UI, ya sea un texto, una imagen o un botón, se crea automáticamente.

Podemos hacer un canvas invisible (desactivándolo en el inspector) y haciéndolo visible mediante el método SetActive(), por ejemplo para mostrar una pantalla de Game Over sobreimpresionada.



Entre otros componentes, tendrá dos por defecto que veremos a continuación:



- Render mode: cómo se adapta a la vista del juego, es decir, a la cámara.
- Canvas scaler: determina en caso de que la pantalla de juego cambie de tamaño, cómo debe comportarse el canvas.

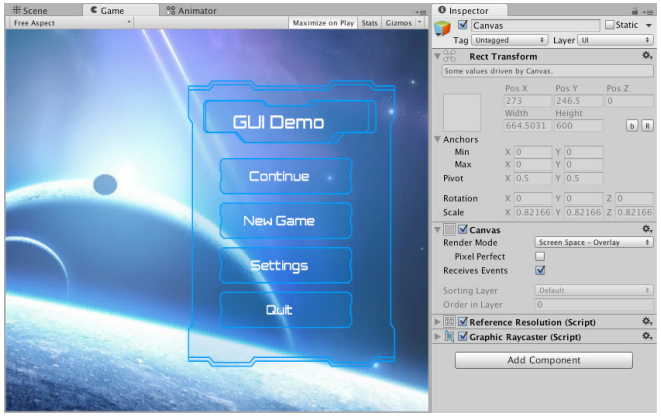
Veámoslos con más detalle.

RENDER MODE

Tiene 3 opciones:

Screen Space - Overlay (Superposición)

Este modo de renderizado, coloca elementos UI en la pantalla mostrada en la parte superior de la escena. Si el tamaño de la pantalla es modificada o cambia la resolución, el Canvas va a automáticamente cambiar el tamaño para que coincida.

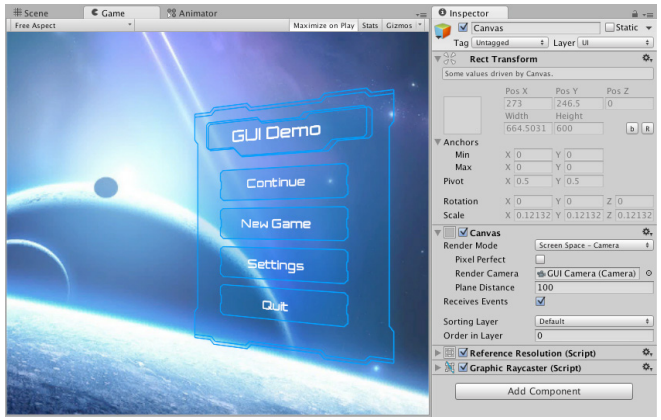


Screen Space - Camera

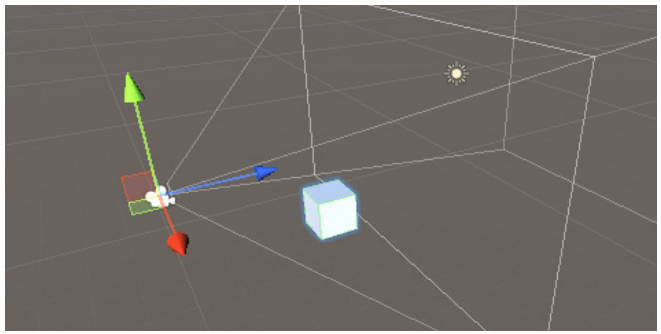
Este modo es similar a Screen Space - Overlay, pero en este modo de renderizado el Canvas se coloca a una distancia dada (Plane Distance) delante de una Camera especificada.

La configuración de la cámara afecta la apariencia de la interfaz de usuario. Si la cámara está configurada en Perspectiva, los elementos de la interfaz de usuario se mostrarán con perspectiva, y si la pantalla cambia de tamaño, cambia la resolución, el canvas cambiará automáticamente el tamaño para que coincida también.

Esta técnica nos permite poner elementos entre la cámara y la UI, creando efectos interesantes.



En la siguiente imagen podemos ver cómo el canvas se adapta al plano de visiónd e la cámara, pero permite colocar objetos entre ambos, que se renderizarán por delante de él:

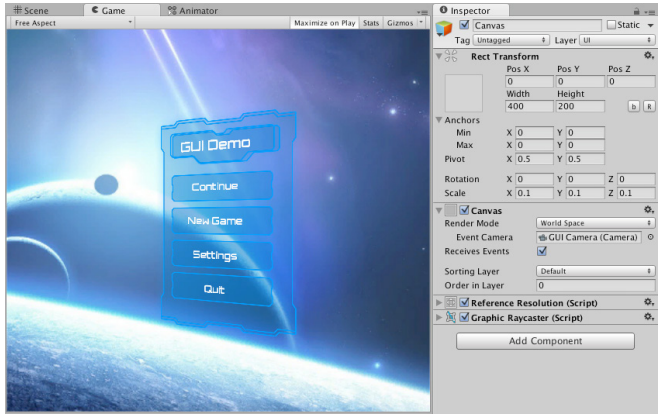


World Space

En este modo de renderizado, el Canvas se va a comportar como cualquier otro objeto en la escena. El tamaño de este Canvas puede ser configurado manualmente utilizando su Rect Transform (que en los otros dos estaban desactivados), y los elementos UI van a renderizar al frente o detrás de otros objetos en la escena basados en una colocación 3D.

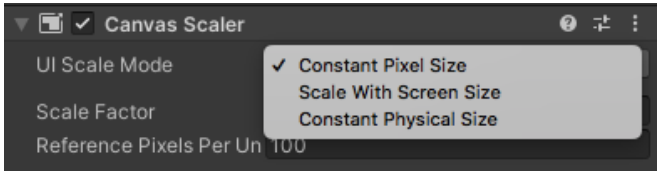
Esto es útil para UIs que están destinados a ser parte del mundo. Esto también es conocido como “diegetic interface”.

Debemos especificar una cámara para facilitar el renderizado en el espacio 3D.



CANVAS SCALER

Es importante indicar cómo se comportarán los elementos dentro del canvas en caso de que la pantalla cambie de tamaño (UI Scale Mode). Tenemos estas tres opciones:



Constan Pixel Size

Los elementos que pongamos en el canvas mantendrán su tamaño original. Es decir, que si ponemos una imagen de 100pxx100px ocupará siempre eso en la pantalla. Eso es un problema si esta va a cambiar de tamaño, pero si no es así, nos permite siempre visualizarlo al tamaño adecuado y garantizar así un correcto flujo de trabajo.

Por ejemplo, si quiero hacer una imagen que ocupe toda la pantalla en una pantalla en HD, la hago a 1920pxx1080px y me aseguro que no se va a escalar. Eso sí, conviene que previsualice en ese tamaño el juego.

Nos aparecerá en el componente las opciones de Scale Factor, que nos permite escalar todo el canvas y sus componentes, y qué referencia en tamaño usa dentro del mundo 3D (por defecto, 100 píxeles equivalen a una unidad).

Scale With Screen Size

Si la pantalla se hace más grande o más pequeña, el canvas se escala con ella. Para qué sepa cómo adaptarse, debemos dar un tamaño de referencia a nuestro canvas (Reference Resolution) y luego decirle cómo comportarse cuando las proporciones no concuerdan: o bien ajusta alto y ancho indicando cuál de los valores debe prevalecer, o bien lo expande para que nunca sea menor que la resolución de referencia o lo encoje para que nunca sea mayor que el de referencia.

Constan Physical Size

Algo más complicado, ya que mantendrá un tamaño físico (medido en milímetros, centímetros, puntos, etc.) que dependerá de la resolución de la pantalla, la cual podemos indicar.

Para más información sobre cómo diseñar para juegos con diferentes resoluciones de pantalla, consultar la documentación de Unity:

https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/HOWTO-UIMultiResolution.html

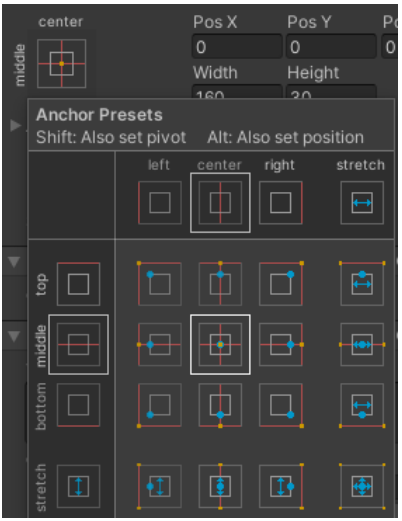
Previsualizar correctamente

Ya que vamos a trabajar con elementos que se distribuyen por la pantalla del juego, es bueno trabajar a la resolución final en nuestro panel de Game (o por lo menos, a la proporción correcta), usando un preset o creando una resolución nueva:

UBICANDO LOS ELEMENTOS

Al colocar una imagen, un texto o cualquier otro elemento en el canvas tendremos que ubicar su posición.

Podemos usar los parámetros del componente transform ("Rect Transform"), que está presente en todos los elementos de la UI, para moverlo o escalarlo de forma precisa:

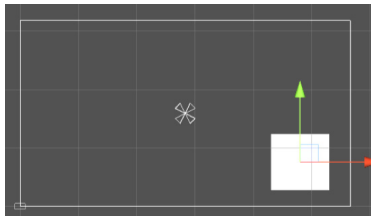


Y si hacemos click en la imagen que aparece arriba a la izquierda, se abrirá la herramienta de "Anchor Presets" que determina respecto a qué punto de la pantalla se establece la posición

Si pinchamos en la posición del elemento, podemos indicar el punto de anclaje respecto al canvas en el que está incluido.

Los datos de posición se hacen siempre relativos al punto de anclaje del elemento respecto a la posición del "Anchor Presets". Esto permite ubicar los elementos correctamente independientemente del tamaño de la pantalla.

No cambia la posición del elemento, sino su punto de anclaje (representado por un puntero blanco). Tendremos que modificar la posición del elemento a 0,0,0 para desplazarlo a ese punto.



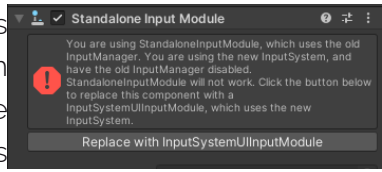
Ejemplo: si le decimos que ubique nuestro elemento de forma relativa a la esquina superior izquierda, en las coordenadas 0 – 0, hará que el punto de anclaje de ese elemento esté siempre centrado en esa esquina, sin importar qué tamaño tiene la pantalla.

RECUERDA: es recomendable usar la herramienta de "Rec Tool" para mover y escalar los elementos 2D de la UI en caso de que queramos hacerlo manualmente.

EventSystem

Antes de ver los elementos que podemos ubicar en el canvas, fijémonos en un elemento que se ha creado en nuestra escena, llamado EventSystem.

Este elemento, que no debemos borrar y que SOLO DEBE HABER UNO, gestiona el comportamiento general de la UI.



IMPORTANTE: entre otras cosas, el EventSystem gestiona la interactividad que usará para moverse por los menús (por defecto, los ejes creados en Unity en el Input Manager). Si hemos instalado el paquete del nuevo InputSystem, nos saltará un error indicando que debemos ir a nuestro EventSystema a reemplazarlo (pulsando el botón "Replace with InputSystem…")

Los aspectos más importantes que tenemos que tener en cuenta son:

- First Selected: qué elemento del canvas está seleccionado al lanzarse la escena. Es importante indicarlo, por si no usamos ratón.
- Deselect On Background: cuando un menú se desactiva o pasa a segundo plano el botón seleccionado se desactiva, algo importante si queremos evitar que se pulse por error.
- Los Inputs que usará para interactuar con los elementos del Canvas.

Álvaro Holguera
CIFP José Luis Garci
(Madrid)

Fuente: https://assetstore.unity.com/packages/2d/gui/sci-fi-buttons-and-panels-pack-142568

ELEMENTOS DEL USER INTERFACE

Ahora es el momento de colocar botones, textos, imágenes y demás elementos que componen nuestrs menús y HUD's

INSERTANDO TEXTOS

Vamos a ver algunos, no todos, de los elementos que podemos añadir a nuestro canvas.

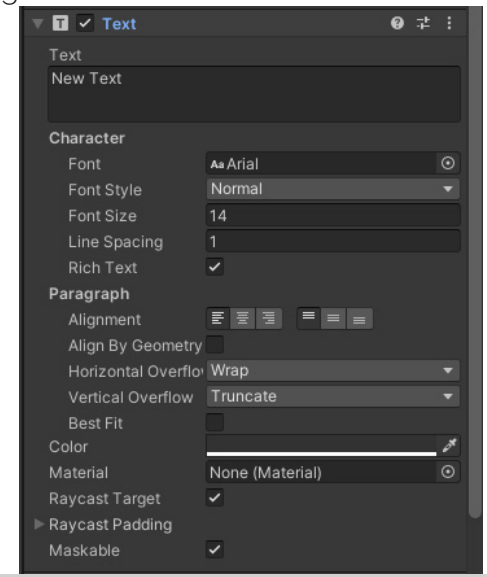
IMPORTANTE: los elementos del canvas se renderizan en orden de aparición, de arriba abajo, por lo que los elementos inferiores se muestran por encima

Textos

La opción más básica es añadir textos a nuestro Canvas. Es importante usar esta forma para añadir información textual, ya que permite cambiarla de forma dinámica, ya sea para el desarrollo del juego o para un proceso de I18n.

En el inspector podremos modificar sus parámetros básicos (recuerda que luego puedes cambiarlos mediante código):

- **Text**: el texto que aparecerá en pantalla. Puedes dejarlo vacío y cambiarlo mediante un script.
- **Fuente y estilos**: podemos cargar en nuestro proyectos tipografías, siempre respetando las licencias de uso.



Podemos importar a nuestro proyecto fuentes externas, copiándolas directamente a la carpeta de Assets, y estarán disponibles como "Font"

- **Tamaño de fuente e interlineado**.
- La opción de "**Rich Text**" nos permitirá añadir estilos mediante código
- **Parámetros del párrafo**: alineación, o cómo se comporta con la caja que lo contiene en caso de que el texto desborde (overflow)
- **Color**. Recuerda que mediante código los colores se indican mediante el método Color.
- **Material**. Pensado para añadir texturas a los textos.
- **Raycast** Target: indicar si es un objetivo para el raycasting, y en caso de serlo el padding.

IMPORTANTE: Si tienes que agrandar un texto, hazlo aumentando el cuerpo del texto, NUNCA escalándalo, o se pixelará. Si es necesario, ajusta el "overflow" o el tamaño de la caja de texto para que no desborda.

Realizar cambios mediante código

En scripting, podemos crear en un script asociado al canvas la variable pública de tipo Text, asignarle el texto en Unity, y mediante código modificar sus parámetros, como el texto que contiene-

Si creamos una variable pública (o serializada) de tipo Text, podremos arrastrar al campo el texto desde el panel de Jerarquía, y a partir de ese momento cambiar el contenido, o el color, o cualquier parámetro. Ejemplo de código:

```
public Text myVar;

//Arrastraremos el elemento texto en Unity

myVar.text = "hola mundo";

myVar.color = Color.white;
```

IMPORTANTE: cualquier texto que ponemos en un UI de tipo texto, debemos convertirlo en string mediante el método ToString(); o bien concatenándolo con una cadena de texto.

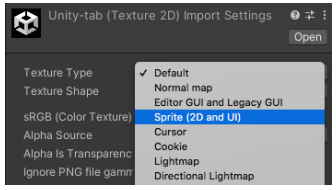
INSERTANDO IMÁGENES

Otro de los elementos básicos de un UI son las imágenes.

Si insertamos una imagen (Game Object > UI > Image) creará un "placeholder" a la espera de asignarle una imagen (sprite).

Cuestiones a tener en cuenta:

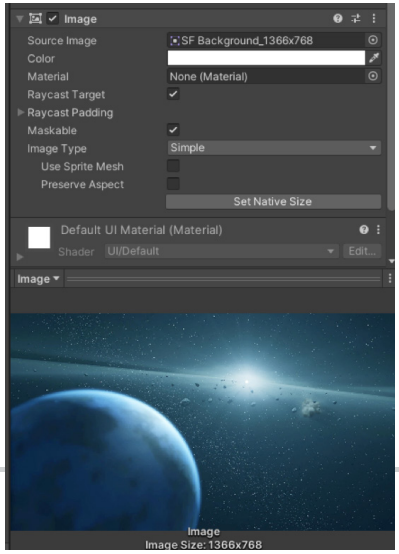
1. Siempre que traemos una imagen a Unity, por defecto entiende que es una textura. Si queremos que lo interprete como un Sprite para UI, debemos seleccionarla en el proyecto y cambar la opción de "Texture Type" a "Sprite(2D and UI)". Debemos pulsar el botón de "Apply"



2. Si arrastramos un sprite directamente a la escena, no lo reconocerá como una imagen de UI, sino como un elemento más de la escena.

Ahora ya podemos seleccionarla como imagen del Sprite:

- **Source Image**: el sprite que tenemos en nuestro proyecto. Recuerda, mediante código podemos crear variables tipo "Sprite" y asociarlas con imágenes de nuestro proyecto, para luego asignárselas a este parámetro.



- **Color**: si es blanco no afecta, pero podemos tintar la imagen o bajar su opacidad en caso de quererla semitransparente.
- **Material**: se puede aplicar un shader al renderizado de la imagen.
- **Raycast Target**: indicar si es un objetivo para el raycasting, y en caso de serlo el padding.
- **Maskable**: permite añadir un canal alpha, para ello tendremos que usar un componente "Mask" y una imagen hija que actúe como canal alpha.
- **Image Type**: cómo se muestra la imagen en el caso de que cambiemos su tamaño y no se corresponda con la imagen original. Las opciones son: simple / sliced / Tiled / Filled.

  - Use sprite mesh: en el proyecto podemos indicar cómo es la malla de la imagen, y si tiene bordes.

  - Preserve Aspect: importante marcarlo para evitar deformarla.

Set Native Size: nos permite devolver la imagen a su tamaño original en el canvas. Si hemos seguido un correcto flujo de trabajo, este es nuestro botón.

Cambiar imágenes mediante código

Cualquier imagen del Canvas tiene el prámatro "sprite" que podemos cambiar por código. Para ello, crearemos una variable de tipo "Sprite" (recuerda que tienes que importar la librería correspondiente), que contendrá el sprite de nuestro proyeto, y otra variable de tipo Image, que contendrá la imagen de nuestro Canvas:

```
[SerializeField] Sprite mySprite;
[SerializeField] Image myImage;
```

De esta forma, podemos asignar al atributo sprite de la imagen, la variable Sprite que hemos creado, y por tanto, su contenido:

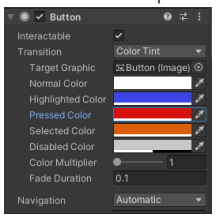
```
void Start()
{
      myImage.sprite = mySprite;
}
```

AÑADIENDO BOTONES

Unity permite añadir un gran nº de elementos interactivos al canvas. El más básico de todos es el botón.

Al añadirlo, mediante GameObject>UI>Button, veremos en el panel de Jerarquía que añade 2 elementos:

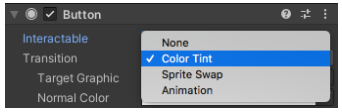
1. El botón en sí, que permite añadir imágenes y colores de fondo. En el componente "Image" del elemento Button, podemos poner una imagen como botón, tintarlo, mantener sus proporciones en caso de escalado, etc.
2. El texto, que se comporta como una caja de texto normal. Si el botón se compondrá solo de una imagen, podemos incluso eliminar ese elemento.



Es buen momento para recordar que un botón puede estar en varios estados:

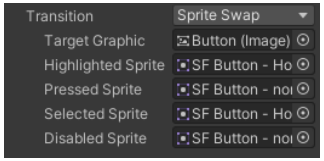
- **Normal**: su estado por defecto
- "Highlighted": cuando pasamos el cursor por encima
- "**Pressed**": al hacer clilck sobre él.
- "**Selected**": tenemos que saber qué botón está seleccionado en cada momento
- "**Disabled**": botones desactivados.

Interactuar con el botón



Si marcamos la casilla de "Interactable" en el componente "Button", podemos modificar cómo se comporta al interaccionar con él, con varias opciones:

- "**None**": el botón es interactivo pero no se produce ningún feedback al navegar por él.
- **Cambio de color** ("Color tint"): dependiendo del estado del botón lo tintamos de un color u otro (blanco lo deja como está).
- **Sprite Swap**: permite cambiar la imagen de fondo dependiendo del estado del botón. Da mucho más juego que la anterior.



- **Animation**. Si pulsamos "Auto Generate Animation", guardará un Animator Controller con tantas animaciones como estados tiene el botón. Podremos ir a cada animación y modificar sus parámetros en la timeline.

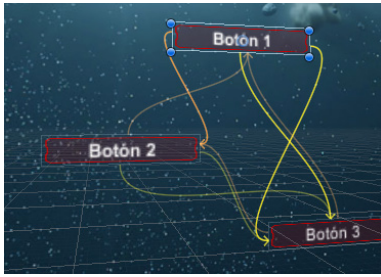
Para ver el resultado, tendrás que dar al play y comprobar cómo queda en el panel de Game.

NOTA: la opción "Target Graphic" indica sobre qué imagen se ejecutan estas interacciones, por defecto el propio botón, aunque podemos cambiarlo por otra imagen de la UI.

En la opción de "Navigation" determinaremos cómo se comportan los cursores del teclado o el gamepad cuando tenemos varios botones en pantalla.

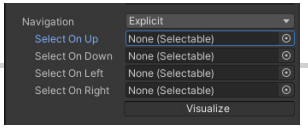
Podemos dejarlo que Unity lo configure de forma automática, o podemos indicar a qué botón se accede cuando usamos los cursores.

Si pulsamos el botón para visualizar, veremos cómo está configurado mediante flechas amarillas:



En esta imagen podemos ver cómo se comporta la navegación entre botones, por ejemplo si pulsamos el cursor izquierdo teniendo seleccionado el botón 1 iremos al botón 2..

Podemos configurarlo manualmente si en el desplegable de Navegación deseleccionamos todas las opciones y elegimos "Explicit".



INTERACTUAR CON LOS BOTONES

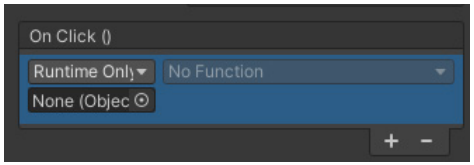
Podemos llamar a métodos de código al hacer click en los botones, que es lo mínimo que tenemos que poder hacer.

RECUERDA: cómo nos movemos a través de un menú se establece en el Event System, y no creas que todo el mundo usa un ratón cuando va a jugar. Y de nuevo: elige un botón seleccionado al lanzar la escena.

Añadir código a un botón

Podemos llamar a métodos de código al hacer click en los botones, que es lo mínimo que tenemos que poder hacer.

En el componente "Button" tienes la posibilidad de añadir eventos a la acción de "On Click". Si pulsamos en el icono de "+" se añadirá uno.



Nos permite adjuntar GameObjects, que si tienen scripts asociados nos dará la posibilidad de llamar a funciones públicas que incluyan esos scripts.

En el desplegable de funciones, ahora aparecerá el script asociado a ese GameObject y dentro las funciones disponibles, habrá una con la clase creada.

Vamos a ver un ejemplo:

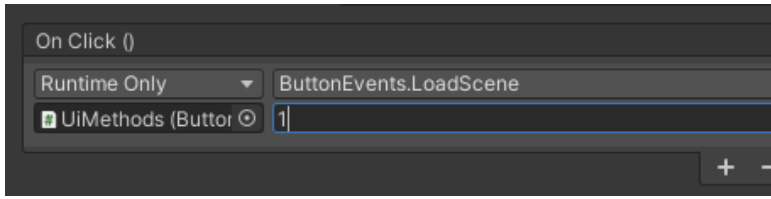
1. Hemos creado un Empty Object, llamado "UiMethods", y le hemos asociado un script llamado "ButtonEvents"

2. En ese script, hemos creado un método público que permite cargar una escena. Fíjate que para ello hemos cargado la librería necesaria y, además, hemos hecho que al método haya que pasarle el nº de escena a cargar:

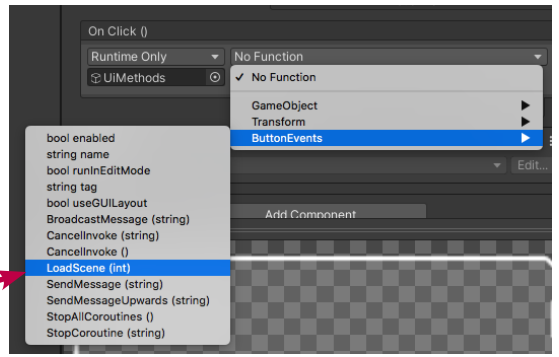
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class ButtonEvents : MonoBehaviour
{
    public void LoadScene(int scene)
    {
        SceneManager.LoadScene(scene);
    }
}
```

3. Ahora, arrastramos ese Empty Object llamado "UiMethods" a la casilla del botón, y verás que en el desplegable de funciones aparecerá el script asociado a ese Game Object, y entre los método heredados aparece el que hemos creado nosotros (por ser público)

4. Como es un método que pide variables (la escena a cargar), podemos usarlo en varios botones, y cada uno que cargue una escena diferente:



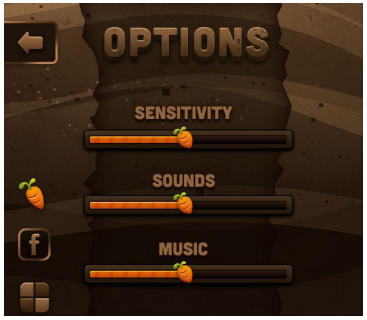
PARA NOTA: Desde código podemos añadir listeners a los botones o a cualquier elemento del UI, lo que nos da mayor control sobre la interactividad, aunque de forma más compleja



LOS SLIDERS

Una herramienta disponible en la UI que nos puede ofrecer muchas utilidades son los Sliders (o barras de desplazamiento)

Un slider puede ser usado para:



- Barra de estado, por ejemplo de carga de una escena por ejemplo
- Modificar parámetros del juego por el usuario (volumen, brillo, etc)
- Barras de vida

Si desplegamos el Slider en la ventana de Jerarquía veremos los diferentes elementos sobre los que podemos actuar, cada uno con sus propiedades en el inspector:

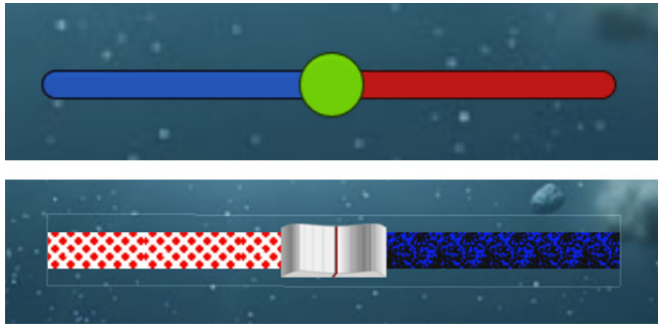
1. Slider: elemento padre y el que contiene la configuración básica de su comportamiento: si es horizontal o vertical, cuales son sus valores máximos y mínimos, y si son números enteros ("Whole Numbers")
2. Background: el fondo del slider cuando que muestra la zona no rellenada. Se puede cambiar por una imagen o cambiar el color.
3. Fill Area: delimita la zona hasta la que llegará el relleno. Si te fijas en el inspector, solo tiene su comomente "Rect Transform", en el que entre otras cosas marca una posición de 5 por la izquierda y un área de seguridad por la derecha de 15. Puedes cambiar esos valores si customizas tu slider.
4. Fill: el color o la imagen que se usará para indicar la zona rellenada del slider. Si te fijas, al igual que el "Background" tiene una Image Type de "Sliced" para adaptarse al área de relleno.
5. Handle Sider Area: área por la que se moverá el Hndle (o asidero).
6. Handle: imagen y color que se usará para el asidero.

Para que lo veas mejor, a la derecha se muestra un slider con un 50% de valor de relleno, al que se le han cambiado los colores que vienen por defecto, y a continuación el mismo al que se le ha cambiado el fondo, el relleno y el asidero por imágenes.

Recuerda que si customizas tu slider, las imágenes que uses las debes configurar como Sprites para UI después de importarlas.

TRUCO: si quieres usar imágenes con patrones que se repiten en el fondo y en el "fill área", en el componente Image de esos elementos deberás poner el Image Type como "Tiled" y así no se deformarán al cambiar el valor del slider.

En la siguiente diapositiva veremos cómo acceder a los valores del Slider.



LOS SLIDERS (2) y CAJAS DE TEXTO

Acceder a los valores del slider mediante código

Mediante código, podemos leer y escribir el estado del Slider para actuar en consecuencia. Para ello, vamos a seguir estos pasos:

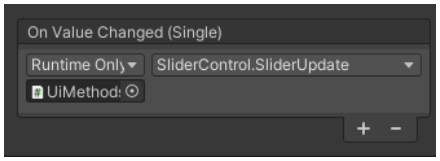
1. Debemos indicar en el componente Slider del elemento slider los valores mínimos y máximos y, sobre todo, si serán números enteros o con decimales, ya que eso determina el tipo de variable.
2. Creamos una variable de tipo Slider serializada o pública en nuestro script (acuérdate de importar la librería UntyEngine.UI). Ahora podremos vincularla con un slider arrastrándolo.
3. Podemos cambiar y/o leer su valor usando el atributo value.

En este ejemplo, nos manda a consola el valor que tiene el Slider en el Start, y automáticamente lo cambia a 50 (admite valores decimales entre 0 y 100):

```
[SerializeField] Slider mySlider;

void Start()
{
    print(mySlider.value);
    mySlider.value = 50f;
}
```

IMPORTANTE: debemos desactivar la casilla "Interactable" si vamos a controlar su valor externamente y no queremos que el usuario lo cambie, por ejemplo, para una barra de energía.

Pero si lo que queremos es que el slider sea interactivo, y que cada vez que el usuario lo toque lancemos una función, debemos ir al inspector del Slider, y en la herramienta de "On Value Changed" añadir un nuevo comportamiento vinculado a un Game Object que contenga el script con la función pública que se lanzará, igual que hicimos con el botón:



En el siguiente ejemplo, veremos que cada vez que movemos el slider, nos muestra en consola el valor modificado.

```
public class SliderControl : MonoBehaviour
{
    [SerializeField] Slider mySlider;

    public void SliderUpdate()
    {
        print(mySlider.value);
    }

}
```

Campos de entrada de datos

Otro tipo de elemento son los campos de entrada de datos (o input fields), diseñados para recoger información directamente del usuario

Al igual que los botones, permite añadir transiciones para sus diferentes estados, o incluso añadir imágenes de fondo.

Otra opción es vincularlo al componente de texto que Unity crea automáticamente, y que recogerá los datos que escriba el usuario. Esta son algunas de sus opciones:

- Limitar el nº de caracteres
- Definir el tipo de texto que se va a introducir (Standard, email, password, etc.)
- Permitir introducir textos de varias líneas
- Placeholder: texto semitransparente que aparece por defecto en la caja, etc.

Desde el código, podemos obtener en cualquier momento lo que está escrito en la caja de texto, creando una variable de tipo Text, vinculada al objeto, y obteniendo su propiedad "text" (es recomendable convertirla en variable "String")

```
public InputField Username_field;
//lo vinculamos en Unity
string userID = Username_field.text.ToString();
```

Como los botones, podemos vincularlo con scripts en la escena y lanzar métodos que se ejecutan cuando cambia algo el contenido de la caja (On Value Changed), o al salir el usuario de la caja indicando que ha terminado (On End Edit)

ALMACENANDO INFORMACIÓN

Ya hemos visto anteriormente cómo mediante variables estáticas podemos guardar valores de una escena a otra. Pero a menudo ocurre que necesitamos conservar esos valores entre una ejecución y otra del programa.

Existen varias formas de conseguir eso, pero la más sencilla es usando el método PlayerPrefs():

Su funcionamiento es bastante sencillo: crea variables (o más bien claves "keys") de datos en el archivo de configuración del programa (la ubicación de ese archivo dependerá del sistema operativo).

Veamos algunos de sus usos y métodos:

- **SetFlloat() – SetInt() – SetString():** establece un valor de tipo decimal, número entero o cadena de texto, para una clave en concreto. Hay que pasarle dos valores: el nombre de la clave y el valor.
- **GetFloat() – GetInt() – GetString():** recupera ese valor. Sólo hay que indicar el nombre de la clave con el que se ha guardado.
- **HasKey():** le indicamos una clave y nos devuelve una booleana que indica si existe (por ejemplo, para comprobar si el usuario ha guardado ya ese dato anteriormente)
- **DeleteKey():** borra una clave guardada.

NOTA: cuando establecemos valores en PlayerPrefs, estos se guardan automáticamente al salir de la aplicación mediante ApplicationQuit(), pero si el programa se cuelga o se crashea, los datos se pierden. Si queremos evitarlo, podemos usar el método PlayerPrefs.Save().

En el siguiente ejemplo vamos a ver cómo al lanzar el juego comprobamos si el usuario ya ha guardado previamente el volumen del juego, y si es así lo ponemos al slider, y si no, ponemos un valor de 50. Y cada vez que el usuario cambia ese slider, lo cambiamos en las preferencias.

```
[SerializeField] Slider mySlider;

void Start()
{
    //Comprobamos si el usuario ha guardado antes el volumen
    if(PlayerPrefs.HasKey("GameVolume"))
    {
        mySlider.value = PlayerPrefs.GetFloat("GameVolume");
    }
    else
    {
        mySlider.value = 50f;
    }

}

public void SliderUpdate()
{
    PlayerPrefs.SetFloat("GameVolume", mySlider.value);
}
```

PARA NOTA: cuando necesitamos guardar información mucho más compleja del juego (pensemos en un RPG), este sistema se queda pequeño. Necesitareos almacenarla con archivos de intercambio de datos, como JSON.

Otros elementos de la UO

Aunque aquí no los veremos, es bueno conocer que existen más elementos disponibles en un Canvas:

- "Toggle": casilla de verificación
- Scrollbar: similar al slider pero diseñado para hacer desplazamientos
- DropDown: desplegable con múltiples opciones
- Panel: funciona como una imagen pero semitransparente, muy útil para destacar elementos por encima del fondo.

NOTA: los elementos que aparecen como "MeshPro" incluyen opciones avanzadas de renderizado y configuración, pero requieren importar elementos de Unity al proyecto.

EJERCICIO

Vamos a dotar a nuestro juego Zaxxon de todo lo neccesario para que contenga una UI y unos menús completos, utilizando para ello todo lo aprendido.

Textos

Ahora que sabes poner textos en pantalla y cambiarlos mediante código, añade a tu juego ZAXXON un texto que indique la distancia.

Ya sabes que la nave no se mueve, pero sí que tenemos un dato importante: la velocidad. Como ya sabes, el espacio recorrido es igual a la velocidad multiplicada por el tiempo transcurrido. ¿Recuerdas cómo podemos saber el tiempo transcurrido?

Ten en cuenta que esa velocidad puede variar, así que mantén el cálculo actualizado.

Es posible que tengas que redondear la cifra para no mostrar demasiados decimales.

```
Mathf.Floor(kmts * 100) / 100
```

Puedes incorporar tus propias tipografías al juego, siempre respetando los derechos de autor, ya lo sabes.

Imágenes

¿Recuerdas que en nuestro juego de ZAXXON habíamos creado una variable estática en el GameManager que gestionaba el número de vidas? Ahora vamos a hacer que eso se traduzca en una imagen en el HUD.

Lo ideal es crear un array de Sprites, que contenga todas las imágenes, y usar el n.º de vidas para elegir el elemento del array.

Puedes trabajar con varias imágenes. Más adelante aprendermos a editar hojas de sprites.

Menús

Vamos a seguir desarrollando nuestro juego de Zaxxon añadiendo todo lo que hemos visto. Además del contador de distancia y de vidas que hemos añadido a nuestro HUD, vamos a crear los menús de juego. Al menos estos:

- Un menú inicial que además será la primera escena que se abra al lanzar el juego. En él, además de elementos gráficos y textos, añadiremos 3 botones:

  Jugar / Configuraciones /Salir

- Un menú de configuración donde el jugador podrá ajustar el nivel de la música (ya aprenderemos más adelante cómo se regula, de momento guarda el dato en una variable), el nivel de dificultad y el nombre del jugador. Todos estos datos deberán estar disponibles en todo el juego.

- Un menú de Game Over que se muestra cuando nos chocamos, dando la opción de volver a jugar o de volver al menú principal. Puedes usar el mismo canvas que el HUD, y hacer los elementos dependientes de un padre que por defecto no aparece y se muestra al chocarnos.

ENLACES A VÍDEOS

Aquí podrás acceder a los vídeos que explican y/o profundizan sobre los conceptos vistos, por si te son de utilidad.

User Interface

Introduccion a la gestión de escenas, el canvas y la introducción de textos:

https://www.youtube.com/watch?v=rokhR6vCy2M

Botones y su interactividad

https://www.youtube.com/watch?v=TyM6IKxIb7g

Sprites

Editar hojas de Sprites (lo veremos más adelante) y gestionar imágenes:

https://www.youtube.com/watch?v=1V9HQyYtz6Q

Imáenes y variables estáticas:

https://www.youtube.com/watch?v=rbPS7jKBo9Y

Otros

En este vídeo se puede ver cómo usar el método DontDestroyOnLoad y cómo construir los singletons:

https://www.youtube.com/watch?v=g1fbpipqg3E

En este vídeo puedes ver cómo almacenar dats complejos:

https://youtu.be/aSNj2nvSyD4