«Talento Tech»

Videojuegos

Clase 09

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 9 | Conceptos básicos

Temario:

- Ejercicio de repaso de clases anteriores

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Repaso

Hoy **integraremos** todos los **conceptos** aprendidos hasta ahora para crear un obstáculo que se desplace de un lado a otro y que, al entrar en contacto con él, haga que nuestro personaje pierda vida.

Agregando Movimiento

La idea de esta clase es tener algo parecido a un juego completo, podemos unificar todo lo que vimos en clase para tener un producto prototípico.
Vamos a reutilizar el código de la clase 7 para el movimiento del jugador; reutilizar código es una práctica muy común en desarrollo de videojuegos que nos permite economizar tiempo.

```
    public float velocidadDeMovimiento;
    public float velocidadDeRotacion;
    public float fuerzaDeSalto;
    private float z;
    private bool estaEnSuelo;
```

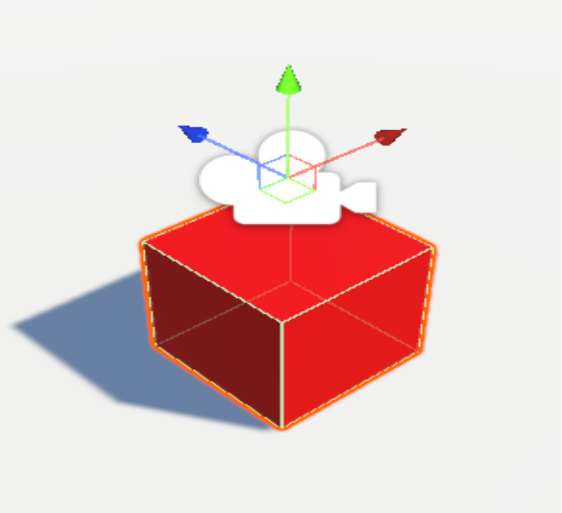
```
private void Update(){
    if (Input.GetKeyDown(KeyCode.Space) && estaEnSuelo){
        GetComponent<Rigidbody>()AddForce(Vector3.up * fuerzaDeSalto,
ForceMode.Impulse);
    }
    // Movimiento para adelante y para atrás
    if (Input.GetKey(KeyCode.W))
        z = 1f;
```

```
    else if (Input.GeyKey(KeyCode.S))
        z = -1f;
    else
        z - 0;

    //Rotación
    if (Input.GetKey(KeyCode.D))
        transform.Rotate(0f, velocidadDeRotacion * Time.deltaTime, 0f);
    if (Input.GetKey(KeyCode.A))
        transform.Rotate(0f, -velocidadDeRotacion * Time.deltaTime,
0f);

    //Aplicamos fuerza para ir para adelante o para atras
    GetComponent<Rigidbody>().AddForce(transform.forward *
velocidadDeMovimiento * z);
}
```

Con esto ya tenemos armado el movimiento. En la jerarquía, podemos agregar y alinear la cámara a nuestro jugador para hacer una vista en primera persona.



Hagamos un nivel de prueba para que podamos movernos alrededor y probar el movimiento de nuestro personaje.

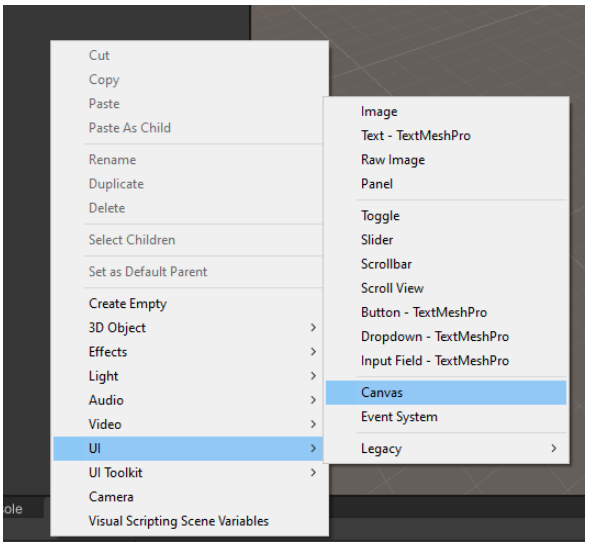
No nos olvidemos de agregar el rayCast para detectar si estamos en el suelo:

```
private void FixedUpdate(){
    RaycastHit hit;
                        if      (Physics.Raycast(transform.position,
transform.TransformDirection(Vector3.down), out hit, 1f)){
        estaEnSuelo = true;
        transform.SetParent(hit.transform);
    } else {
        estaEnSuelo = false;
        transform.SetParent(null);
    }
}
```

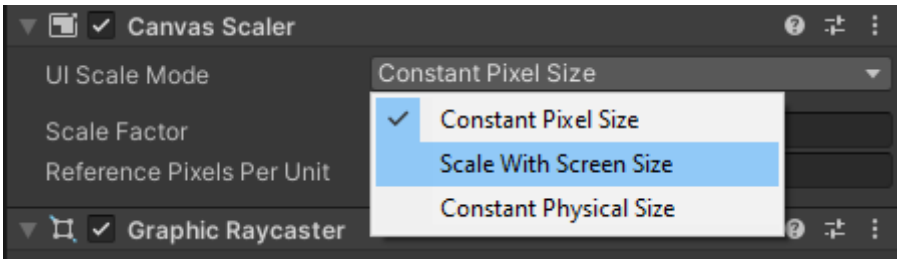
Buenos Aires aprende
Agencia de Habilidades para el Futuro

Implementando UI

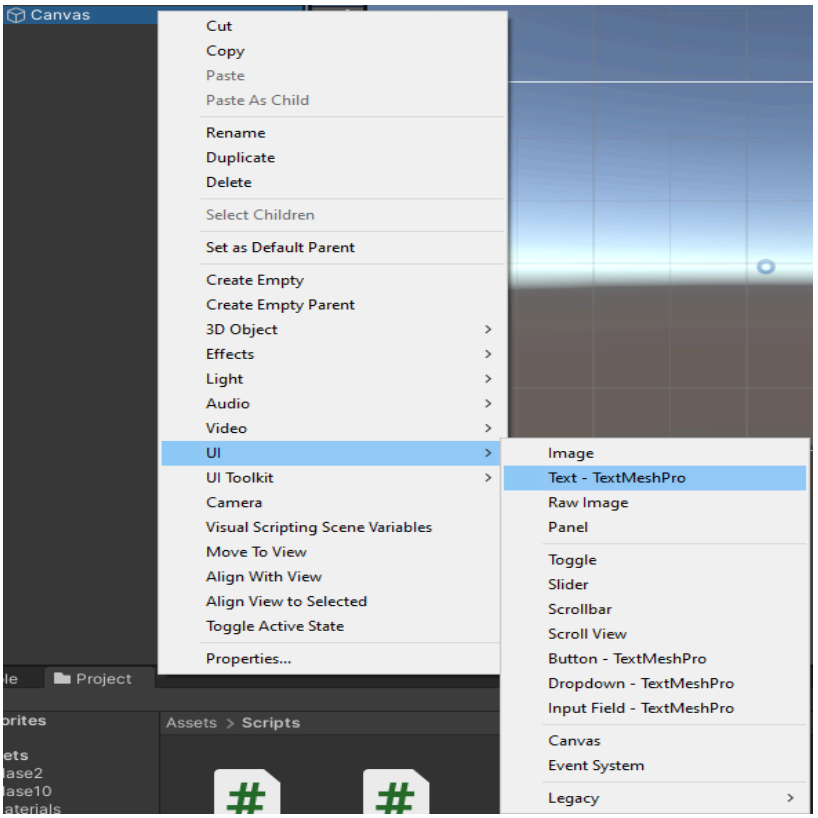
Como en la clase 6, tenemos que agregar desde la jerarquía un objeto de tipo **UI** → **Canvas** que va ser donde agreguemos la interfaz.



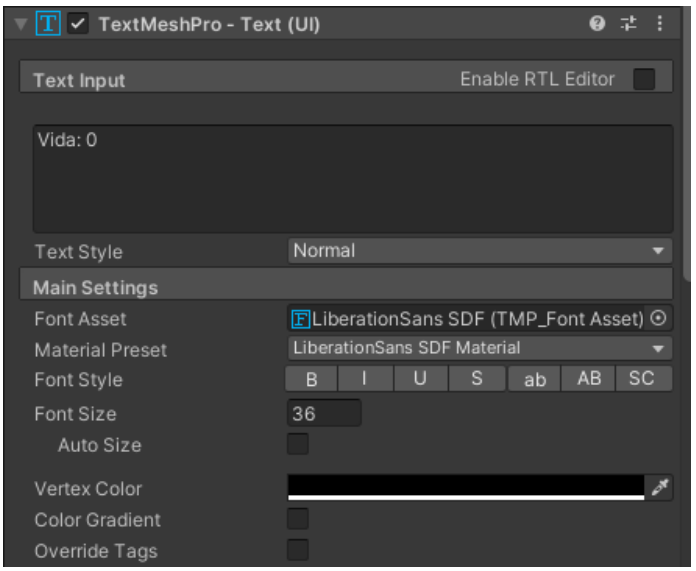
Acuérdense de modificar el **"UI Scale Mode"** a **"Scale With Screen Size"** para que los elementos de interfaz mantengan una proporción acorde al tamaño de la pantalla.



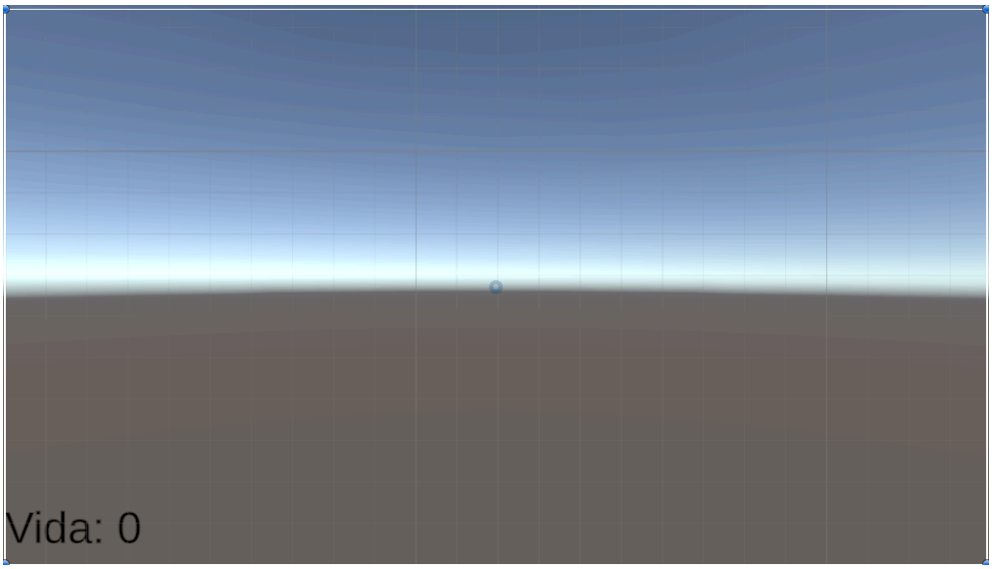
Agregamos un texto haciendo clic derecho sobre el canvas y seleccionando **"Text - TextMeshPro".**



Vamos a cambiarle el color para poder ver mejor el texto. La idea es tener una variable que se traduzca al texto en la interfaz. Podemos escribir lo siguiente en el campo de texto.



Y vamos a acomodar el texto para que no esté centrado, podemos elegir una esquina para no tapar la vista del jugador, en este caso la esquina derecha inferior, aunque esto queda a discreción de cada diseñador/a.



Creando un sistema de vida

Tenemos el movimiento de nuestro personaje, y tenemos una interfaz con un número que va a representar la vida actual de nuestro personaje, pero nos faltan algunas cuestiones importantes aún. Primero tenemos que crear una variable que represente la vida en el código, y después tenemos que vincularlo con el número que aparece en pantalla. Vamos al código.

```
Using UnityEngine;
Using TMPro;

public class MovimientoJugador : MonoBehaviour{
    public float velocidadDeMovimiento;
    public float velocidadDeRotacion;
    public float fuerzaDeSalto;
```

```
        private float z;
        private bool estaEnSuelo;

        private int vida;
        public TextMeshProUGUI textoVida;
    }
```

En la línea 2, agregamos la biblioteca **“TMPro”** que nos permite utilizar las clases de texto para poder modificarlo y crear la variable de tipo **“TextMeshProUGUI”** en la línea 14 al cual vamos a nombrar **“textoVida”**. Por último, en la línea 13 vamos a hacer una variable pública de tipo número entero llamado **“vida”** y que, como dijimos anteriormente, va a representar la cantidad de vida actual.

Dentro de la función de **Update(),** vamos a refrescar todos los frames, el número en pantalla, y que refleje el valor de la “variable” vida en todo momento. Esto vamos a lograrlo añadiendo la siguiente línea de código dentro de **Update()**:

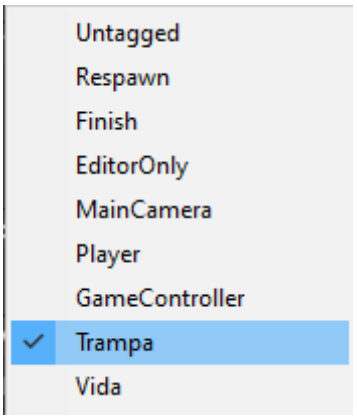
```
    textoVida.text = "Vida: " + vida.ToString();
```

Acordémonos que “textoVida” representa el texto en pantalla, y que podemos modificar su campo de texto con el “.text”. Luego hacemos una concatenación de tipos de variables, es decir, mezclamos un string con un int gracias al símbolo “+” y la función **ToString()** para que el valor de “vida” pase a ser un string.

Para terminar este apartado, es importante que **“textoVida”** esté referencia dentro del editor de **Unity.** Tenemos que arrastrar desde la jerarquía el texto, hacia la variable dentro del Inspector.

PickUps que interactúan con la vida

Vamos a hacer que nuestro sistema de vida tenga sentido gracias a unos pickUps que van a afectar la variable “vida” gracias al uso de los tags y las colisiones como vimos en la clase 4. Añadiremos dos tags: “Vida” y “Trampa”. Una va a sumar vida cuando la agarramos y la otra nos va a restar.



En el código, vamos a usar la función **OnTriggerEnter()** para detectar los objetos según los tags, y dependiendo si es “Vida” o “Trampa”, le vamos a restar a “vida” o sumarle. Además a destruir el objeto agarrado con “Destroy(other.gameObject)”.

```
private void OnTriggerEnter (Collider other){
    if (other.gameObject.tag == "Vida"){
        vida += 1;
        Destroy(other.gameObject);
    }
    if (other.gameObject.tag == "Trampa"){
        vida -= 1;
        Destroy(other.gameObject);
    }
}
```

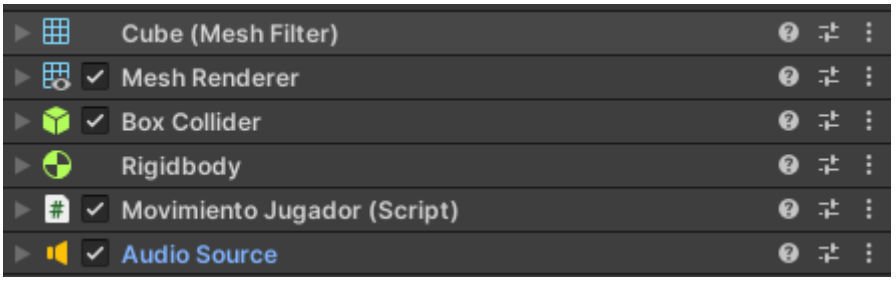
Por último, como en la clase anterior. Vamos a crear un sistema de sonido. Agregando un **AudioSource**, un sonido, y usando la función **“PlayOneShot()”**

cada vez que agarramos un objeto. A cada situación le agregamos un sonido distinto.

```
    private AudioClip sonidoTrampa;
    private AudioClip sonidoVida;
```

```
private void OnTriggerEnter(Collider other){
    if (other.gameObject.tag == "Vida"){
        vida += 1;
        Destroy(other.gameObject);
        GetComponent<AudioSource>().PlayOneShot(sonidoVida);
    }
    if (other.gameObject.tag == "Trampa"){
        vida -= 1;
        Destroy(other.gameObject);
        GetComponent<AudioSource>().PlayOneShot(sonidoTrampa)
    }
}
```

No nos olvidemos de agregar el AudioSource a nuestro jugador.



Buenos Aires aprende
Agencia de Habilidades para el Futuro

Para los pickUps, hagamos unos cubos, hagamos que sean triggers, les cambiamos el color con un material, y le asignamos los correspondientes tags para que podamos hacer un nivel.

Si todo nos funciona bien, es momento de ponernos más creativos/as y empezar a hacer nuestros propios niveles.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Desafío N° 9:

Aprovechá para ponerte al día. Seguí la clase y tratá de llegar al final, hasta implementar las plataformas móviles.

Crear un nivel con las plataformas que tenga un principio y un fin. No tengas miedo en ponerte creativo/a.

No te olvides de **guardar** todos los cambios.

Tomá una **captura de pantalla** del personaje con el nuevo material

Subilas al espacio correspondiente del Desafío 9.


Buenos Aires
aprende
Buenos
Aires
Ciudad