«Talento Tech»

# Videojuegos

Clase 07

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

# Clase N° 7 | Conceptos básicos

## Temario:

- Función salto
- Movimiento de plataformas
- Raycast

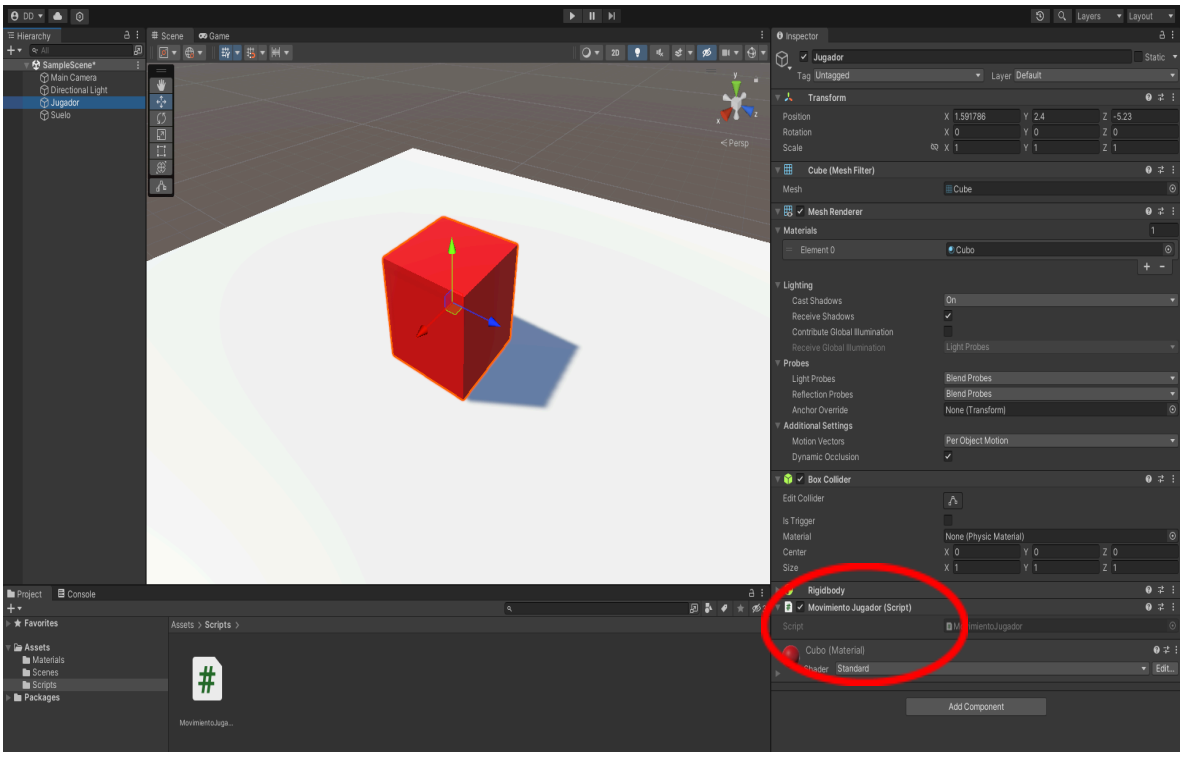
Buenos Aires aprende
Agencia de Habilidades para el Futuro

## Función de salto con AddForce()

En la Clase 3 hicimos una función para el movimiento de un objeto utilizando la función ***Addforce()***. En esta clase vamos a profundizar en esta función pero para lograr que nuestro personaje salte utilizando el motor de físicas de Unity.

Con nuestro objeto en escena vamos a asegurarnos primero que este posea un **Rigidbody**, recuerden que este componente es el que le da las propiedades físicas a los objetos.

Vamos a crear un script que se va a llamar **"MovimientoJugador"** y vamos a escribir todo el código que tenga que ver con el movimiento. Acordémonos de agregar este código al objeto que vamos a mover. En la siguiente imagen se puede ver el **Rigidbody** el script que recién creamos dentro de un cubo al cual llamamos "Jugador".




**Ahora pasemos a nuestro script:** Primero vamos a hacer tres variables de tipo float:

```
public float velocidadDeMovimiento;

public float velocidadDeRotacion;

public float fuerzaDeSalto;
```

Como dicen sus nombres, vamos a usar tres datos para el movimiento:

**Punto 1:** La velocidad a la que nos vamos a mover.

**Punto 2:** La velocidad de rotación

**Punto 3:** La fuerza de salto.

A estas variables les vamos a dar un valor en Unity y según el *feeling* que quieran conseguir, sus valores van a cambiar.

## Implementando el salto

Para implementar el salto podemos hacer una función, para que se entienda mejor, vamos a escribir todo el código dentro del **Update()** y después lo podemos trasladar a una función. Acuérdense que todo lo que sea registro de inputs lo tenemos que hacer dentro del **Update()**. Usamos el **Input.GetKeyDown()** dentro de un condicional para registrar una tecla y luego ejecutar una acción.

En este caso, usamos el **AddForce()** como en la clase 3: multiplicamos nuestra fuerza de salto por "**Vector3.up**", que es lo mismo que escribir **(0, 1, 0)**, es decir, solo tomamos el eje Y en positivo y lo multiplicamos por el valor de **"fuerzaDeSalto"**. Con una coma**(",")**, podemos elegir otro parámetro de AddForce que es un "ForceMode" y podemos elegir el modo de fuerza "Impulse" para que la fuerza que se le aplica al objeto sea de tipo impulso.

```
    private void Update(){
```

```
        if (Input.GetKeyDown(KeyCode.Space)){

                    GetComponent<Rigidbody>().AddForce(Vector3.up * fuerzaDeSalto, ForceMode.Impulse);

        }

    }
```

## Implementando el movimiento

Para implementar el movimiento, vamos a reutilizar lo que aprendimos en la clase 3. Vamos a crear una variable que va a adoptar un valor según qué tecla presionemos y después vamos a multiplicar ese número (positivo, negativo o nulo) por una velocidad de movimiento.

```
    public float velocidadDeMovimiento;

    public float velocidadDeRotacion;

    public float fuerzaDeSalto;

    private float z;
```

Esta variable se va a llamar “z”, porque en 3D es el eje que representa “adelante” y “atrás”, y va ser privado porque no queremos modificarlo en el editor de Unity.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Dentro de **Update()** vamos a agregar lo siguiente:

```
private void Update(){

    if (Input.GetKeyDown(KeyCode.Space)){

        GetComponent<Rigidbody>().AddForce(Vector3.up * fuerzaDeSalto,
ForceMode.Impulse);

    }

    // Movimiento para adelante y para atrás

    if (Input.GetKey(KeyCode.W))

        z = 1f;

    else if (Input.GetKey(KeyCode.S))

        z = -1f;

    else

        z = 0;

    //Aplicamos fuerza para ir para adelante o para atras

    GetComponent<Rigidbody>().AddForce(transform.forward *
velocidadDeMovimiento * z);

}
```

Como dijimos, este código es similar al de la clase 3: **“z”** adopta un valor positivo, negativo o cero si no apretamos ninguna tecla, y luego se la multiplicamos a **“velocidadDeMovimiento”** para que produzca un movimiento en el eje Z gracias a **“transform.forward”** que es lo mismo que escribir **“Vector3(0f, 0f, 1f),** es decir que siempre detecta cuál es el “adelante” del objeto.

## Implementando de rotación

Para la rotación vamos a hacer algo más simple, según qué tecla apretemos (en este caso “A” o “D”) vamos a utilizar “transform.Rotate()” para rotar nuestro objeto. Esta función nos pide un Vector3, es decir, que le pasemos que eje queremos rotar, como sólo queremos rotar el eje **Y** vamos a dejar la **X** y la **Z** en cero. En un caso vamos a sumar a la rotación, y en el otro caso vamos a restarle para que vaya hacia la izquierda:

```
private void Update(){

    if (Input.GetKeyDown(KeyCode.Space)){

        GetComponent<Rigidbody>().AddForce(Vector3.up * fuerzaDeSalto,
ForceMode.Impulse);

    }

    // Movimiento para adelante y para atrás

    if (Input.GetKey(KeyCode.W))
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

```
        z = 1f;
    else if (Input.GetKey(KeyCode.S))
        z = -1f;
    else
        z = 0;
    //Rotación
    if(Input.GetKey(KeyCode.D))
    transform.Rotate(0f, velocidadDeRotacion * Time.deltaTime, 0f)
    else 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);
}
```

## Physics.Raycast()

Si prueban, saltar van a notar que podemos saltar en el aire, porque la única condición que tenemos para saltar, es que apretemos espacio, lo cual queremos corregir para nuestro juego. Necesitamos tener alguna noción de que estamos en el suelo. Para eso vamos a utilizar un Raycast y una variable booleana que vamos a llamar **"estaEnSuelo"**, y en nuestro código de saltar vamos a agregar una nueva condición con los signos "&&".

```
public float velocidadDeMovimiento;

public float velocidadDeRotacion;

public float fuerzaDeSalto;

private float z;

private bool estaEnSuelo;
```

```
if (Input.GetKeyDown(KeyCode.Space) && estaEnSuelo){

GetComponent<Rigidbody>().AddForce(Vector3.up * fuerzaDeSalto,
ForceMode.Impulse);

}
```

Raycast es una función en Unity que se utiliza para lanzar un rayo desde un punto (de origen) en una dirección específica y ver si ese rayo golpea (colisiona) algo en su camino.

Aunque para entender este concepto necesitamos entender tres puntos:

**Origen *(origin)*;** El punto desde el cual comienza el rayo. Desde donde apuntas.
**Dirección *(direction)*:** Es la dirección en la que se lanza el rayo. Hacia donde apuntas.
**Colisión *(hit)*:** Si el rayo golpea algo, obtenemos información sobre ese impacto, como la posición donde golpeó y el objeto que fue alcanzado.

Dentro del FixedUpdate, vamos a escribir lo siguiente:

```
private void FixedUpdate(){

    RaycastHit hit;

    if (Physics.Raycast(transform.position,
transform.TransformDirection(Vector3.down), out hit, 1f)){

        estaEnSuelo = true;

    }

    else {

        estaEnSuelo = false;
```

```
        }

}
```

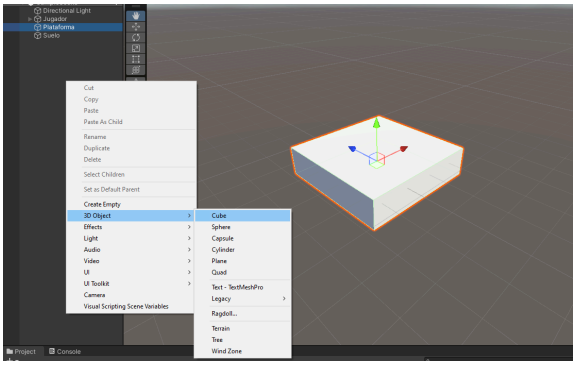
Primero creamos una variable local de tipo **RaycastHit,** que va a representar con lo que choca el rayo, si por ejemplo, hace contacto con el suelo, la variable “hit” va ser el suelo en el código.

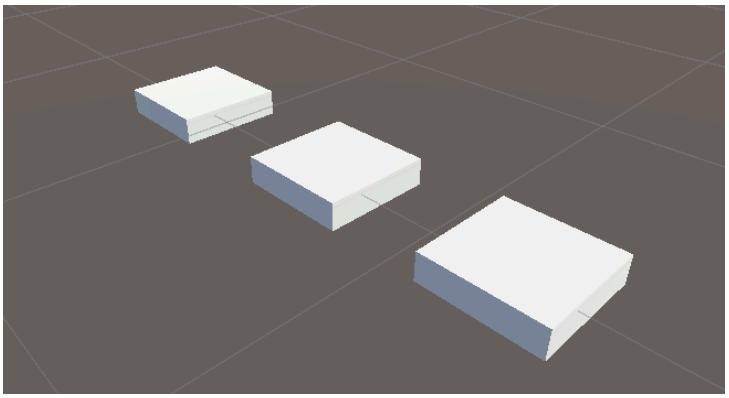
Luego, dentro de un condicional, vamos a utilizar la función **“Physics.Raycast()”** que dentro de los paréntesis nos pide varias cosas:

El punto de origen, la dirección, la variable que va a almacenar la colisión y por último, cuánto va a medir el rayo desde el punto de origen. Si se cumple esta condición, “estaEnSuelo” va a ser verdadero y si no va a ser falso. Con esto ya arreglamos el problema de que nuestro personaje saltaba en el aire, porque solo va a poder saltar si el raycast detecta que está en el suelo.

## Agregando plataformas

Vamos a agregar un elemento que le dé sentido a nuestra mecánica de saltar. Vamos a añadir plataformas para que nuestro jugador pueda saltar sobre ellos. Podemos crear un cubo y cambiarle un poco los números en “scale” para darle la forma deseada, y vamos a llamarlo “Plataforma”. Luego podemos apretar Ctrl + D para duplicar las plataformas.



## Moviendo las plataformas

Ya tenemos algo más interesante, pero podemos mejorar estas plataformas, podemos agregarle movimiento con un poquito de código. Creamos un script nuevo y lo llamamos Plataforma y lo abrimos.


Para lograr nuestro objetivo, tenemos que crear unas funciones primero:

```
public float velocidadDeMovimiento;

public float limite;

private float recorrido;

private bool yendo;
```

Un **número flotante público** para determinar la **velocidad de movimiento** de la plataforma.

Un **número flotante público** que va a ser un número que representa el **límite hasta donde llega** la plataforma antes de pegar la vuelta.

Un **número flotante privado** que va a representar el **trayecto recorrido** para saber si llegamos al “límite”.

Una **variable booleana** que va a determinar si la plataforma **está yendo** (sumando a “recorrido) **o no**.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Perfecto, ahora vamos a agregar lo siguiente en el **Update():**

```
    void Update(){

        if (recorddio <= 0)

            yendo = true;

        else if (recorrido >= limite)

            yendo = false;

    }
```

Acá estamos estableciendo como va a funcionar la lógica de nuestra plataforma. Vamos a tener una coordenada “0” y un límite, y “recorrido” va a ser cuando ha recorrido la plataforma. Si el recorrido es menor o igual a cero, entonces vamos a hacer que la plataforma vaya, y si es mayor o igual al límite vamos a hacer que venga. Ahora va a tener un poco más de sentido cuando le agreguemos el siguiente código:

```
void Update(){

    if (recorddio <= 0)

        yendo = true;
```

```
    else if (recorrido >= limite)

        yendo = false;



    if (yendo){

                transform.Translate(new  Vector3(velocidadDeMovimiento  *
Time.deltaTime, 0f, 0f));

        recorrido += velocidadDeMovimiento * Time.deltaTime;

    } else {

                transform.Translate(new  Vector3(-velocidadDeMovimiento  *
Time.deltaTime, 0f, 0f));

        recorddio -= velocidadDeMovimiento * Time.deltaTime;

    }

}
```

Si estamos yendo, vamos a usar el **"transform.Translate()"** que vimos en la clase 3. Y a "recorrido" le vamos sumar la velocidad a la que se mueve más el tiempo transcurrido, y se lo vamos a restar cuando "yendo" sea falso. Con esto vamos a hacer que la plataforma se mueva en **X** ida y vuelta según cuando valga "límite". Asegúrense de agregar el script "Plataforma" a los objetos que quieran que tengan este comportamiento.

## Solucionando un error

En principio ya tenemos lo más básico de nuestro juego plataformero. Pero si lo prueban un poco van a notar algo molesto; cuando nos subimos a una plataforma, nuestro jugador no sigue el movimiento de la misma. Gracias a que implementamos un raycast, la solución a nuestro problema es bastante sencillo y solo va a requerir dos líneas de código extra en la parte donde declaramos el raycast:

```
    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

Cuando entremos en contacto con algo, vamos a usar la función **“SetParent()”** para hacer que el jugador sea un objeto hijo del objeto sobre el que está parado, haciendo que imite los movimientos de la plataforma sobre la que está parado. Si estamos en el aire, vamos a hacer que no tengan ningún objeto padre **(null).** Y con esto ya vamos a tener un sencillo y divertido plataformero.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

# Desafío N° 7:

**Con la lógica de salto que armamos en clase:** pensar y escribir un código que nos permita hacer realizar un doble salto.

**A tener en cuenta:** No existe UNA sola forma de realizar la consigna, tratá de hacerlo por tu cuenta. No te olvides de usar los condicionales y no tengas miedo de experimentar distintas soluciones.

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

Tomá una **captura de pantalla** del código.

**Subilas** al espacio correspondiente del Desafío 7.


Buenos Aires
aprende
Buenos
Aires
Ciudad