TALLER DE METAVERSO: REALIDAD VIRTUAL

TELECO RENTA

PLAN DE
PROMOCIÓN DE LOS ESTUDIOS
DE TELECOMUNICACIÓN

Plan de Recuperación, Transformación y Resiliencia

Financiado por la Unión Europea
NextGenerationEU

UNICO I+D

ACTIVIDAD 1. HatchXr RECOLECTA DE DRONES

Bienvenidos al curso de TelecoRenta.

¿Estáis listos? Espero que sí, porque tenemos preparadas unas tareas muy interesantes.

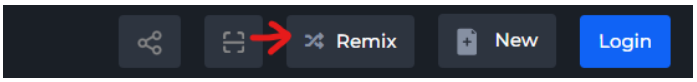
Hoy, nos adentraremos en el mundo del METAVERSO y algunas de las tecnologías asociadas. La primera de ellas es la Realidad Virtual. Este concepto puede ser familiar, pero ¿qué es la realidad virtual? La Realidad Virtual nos permite entrar en un mundo totalmente diferente. ¡Imaginaros explorando selvas, volando a través del espacio y todo esto sin salir de la sala! Suena emocionante, ¿verdad? Pues esto es posible gracias a la herramienta HatchXR.

Así que manos a la obra. Nos encontramos en la Universidad Politécnica de Catalunya y tenemos una misión: ¡recoger todos los cinco drones del dronlab!

- *Video recogiendo los drones:*
  https://drive.google.com/file/d/1s6KRIL_nBG_4_vthFXrQnx6PyFrUCw8l/view?usp=drive_link

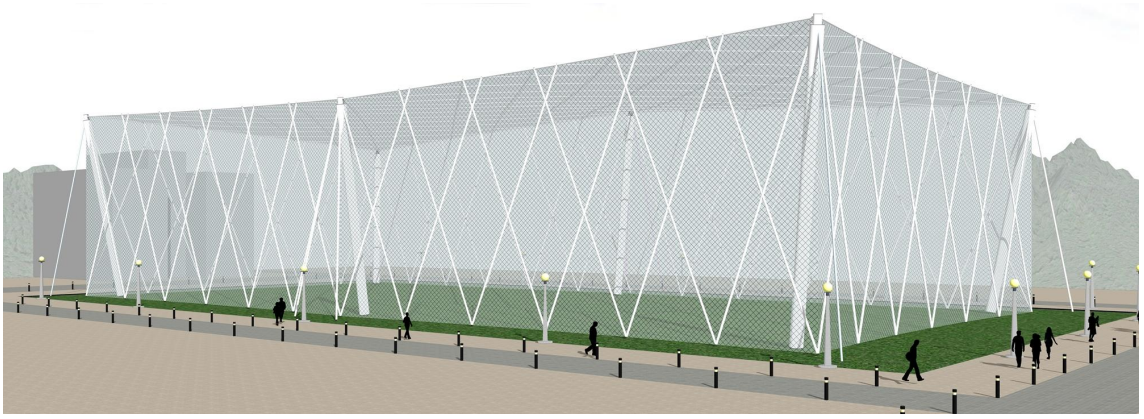
¡Logrado! ¿Quieres intentarlo? Pues es tu turno, pero primero debemos configurar el escenario. Vamos paso a paso:

- *Preparación del entorno:*
  - Acceder desde un navegador (preferible chrome) a la URL: *https://hatchxr.com/*
  - Crear una cuenta en la plataforma si no se dispone
  - Importar el proyecto base de la siguiente URL: *https://hatchxr.com/@ResearchProposals/Teleco-Renta*
  - Crear vuestra propia versión con los cambios dando clic a “Remix” arriba a la derecha:



- *Explicar los diferentes pasos para construir el recinto de los drones y añadir los drones en el metaverso de la EETAC:*

- Como hemos introducido anteriormente, nos encontramos en el campus del Baix Llobregat de la UPC. Así que 1r paso "**analizar el escenario**". Podemos identificar un edificio con tres módulos, el cual represente la universidad. En el lateral izquierdo encontramos un recinto, esto es el dronlab (https://pmt.es/ca/dronlab)



- En el siguiente vídeo veremos cómo se ha realizado el **montaje del dronlab virtual en el metaverso de le EETAC.**
    1. Crearemos las 4 paredes, y el techo. Es tan sencillo como ir a la barra de herramientas superior y en la primera opción, seleccionar el objeto "caja". Modificamos altura, el ancho y la profundidad, y orientación para obtener el recinto deseado.
    2. Para diseñar la puerta del dronlab nos las tenemos que ingeniar diseñando las paredes de tal manera que dejemos una entrada por la que el jugador podrá entrar.
    3. Para las columnas laterales, procederemos de la misma manera, pero la opacidad y color de los objetos será diferente.
    4. Por último, colocaremos los 5 drones. Para hacerlo, en la segunda opción de la barra de herramientas, podemos buscar diferentes objetos predeterminados. Una vez escogido el dron deseado, lo seleccionamos y arrastramos dentro del recinto.

Por último, pero no menos importante, la parte lógica, es decir, el código de las funciones.

- *Explicación de las funciones*:

```
// Variable para rastrear la cantidad de drones recolectados
var drones = 0;

// Función que se ejecuta cuando la escena se carga
Hatch.onSceneLoaded(function (event) {
  // Llama a la función moveDrone para mover cada dron en la escena
      moveDrone(Drone, 'timer1');
      moveDrone(Drone2, 'timer2');
      moveDrone(Drone3, 'timer3');
      moveDrone(Drone4, 'timer4');
      moveDrone(Drone5, 'timer5');
});
```

Esta función se ejecuta cuando se carga la escena del juego se carga.

En ese momento, llama repetidamente a la función "moveDrone" para iniciar el movimiento de cinco drones diferentes en la escena, asignándoles temporizadores únicos para controlar sus patrones de movimiento hacia arriba y abajo. La función “moveDrone” se ve a continuación.

```
// Función que detecta colisiones entre el jugador y los las paredes del Dronlab
PlayerBox.detectCollisionsWith([Dronlab1, Dronlab2, Dronlab3,
Dronlab4, Dronlab5, Dronlab6, Dronlab7], function
(collidedObject) {
  // Cambia el color del objeto con el que colisionó el jugador a rojo
    collidedObject.setColor('#d62222');
  // Crea un temporizador para restaurar el color original después de 2 segundos
  Hatch.createTimer('colorTimer', 2000, function () {
        collidedObject.setColor('#dedede');
        Hatch.removeTimer('colorTimer');
    });
});
```

Esta función **detecta colisiones entre el jugador y un conjunto de objetos** representados por Dronlab1, Dronlab2, ..., Dronlab7 en el juego, es decir, las paredes del DronLab.

Cuando se produce una colisión, cambia el **color** del objeto con el que el jugador colisionó a rojo.

Luego, crea un **temporizador** que restaurará el color original del objeto a su estado original después de 2 segundos.

Si se detectan múltiples colisiones en rápida sucesión, el temporizador asegura que cada objeto afectado vuelva a su color original sin problemas.

```
// Función que detecta colisiones entre el jugador y los drones
PlayerBox.detectCollisionsWith([Drone, Drone2, Drone3, Drone4, Drone5], function (collidedObject) {
  // Destruye el dron con el que colisionó el jugador
    Hatch.destroyObject(collidedObject);
  // Incrementa la cantidad de drones recolectados
    drones += 1;
  // Actualiza el texto que muestra la cantidad de drones recolectados
  TextDrones.setText('Drones: ' + drones + '/5');
});
```

Esta función detecta **colisiones entre el jugador y los 5 drones** (representados por Drone, Drone2, Drone3, Drone4 y Drone5 en el juego).

Cuando una colisión se produce, la función:

- **destruye el dron** con el que el jugador colisionó
- **incrementa un** contador de drones recolectados
- **actualiza un texto** que muestra la cantidad de drones recolectados.

Además, verifica si se han recolectado los 5 drones requeridos para completar un objetivo en el juego.

```
// Función para mover un dron arriba y abajo en un bucle
function moveDrone(drone, timer) {
    var moveDirection = 1;   //1 para mover hacia arriba, -1 para mover hacia abajo
    var posY = drone.getY();
  // Crea un temporizador que se ejecuta cada 50 milisegundos
    Hatch.createTimer(timer, 50, function () {
    // Mueve el dron en la dirección especificada (arriba o abajo).
        posY += moveDirection;
        drone.setY(posY);
    // Cambia la dirección si el dron alcanza los límites superiores o inferiores
     if (posY >= 50 || posY <= 0) {
            moveDirection *= -1;
        }
    });
}
```

La función "moveDrone" hace que un **dron se mueva hacia arriba y abajo en un bucle**.

Se inicializa la variable **moveDirection** en 1, lo que significa que el dron comenzará moviéndose hacia arriba. También se obtiene la posición vertical actual del dron (**posY**) utilizando **drone.getY()**.

A continuación, se crea un **temporizador** que se ejecuta cada 50 milisegundos. En cada ciclo de este temporizador, se actualiza la posición vertical del dron (**posY**) añadiéndole el valor de **moveDirection**. Esto hace que el dron se mueva gradualmente hacia arriba o hacia abajo.

El siguiente paso es verificar si el dron ha alcanzado los límites superior o inferior. Si la posición vertical (**posY**) supera los 50 píxeles o desciende por debajo de 0 píxeles, se cambia la dirección del movimiento multiplicando **moveDirection** por -1. Esto invierte la dirección del movimiento, haciendo que el dron cambie de dirección

¡Pues ya estaríamos! Y ahora viene la mejor parte, a jugar y disfrutar.

ACTIVIDADES PROPUESTAS PARA LOS ALUMNOS:

Ahora que hemos explorado la implementación lógica de la creación y actualización de un texto (como en el caso de los contadores de los drones), proponemos una actividad para los estudiantes: **crear visualmente un temporizador que comienza con un valor predefinido**.

Los estudiantes deberán desarrollar la función para esta tarea. Una vez que el contador llegue a cero, podrán realizar diversas acciones, como cambiar el color del texto del contador.

Para aquellos que busquen un desafío adicional, se sugiere que intenten modificar la función para detener el movimiento de los drones después de que el contador llegue a cero.

ACTIVIDAD 2. Unity CIRCUITO DRONES

¡Bienvenidos nuevamente a TelecoRenta!

En la actividad anterior, exploramos la herramienta HatchXR, que nos permitió sumergirnos en el mundo de la Realidad Virtual y recoger drones en un entorno virtual. Sin embargo, en el mundo laboral, aunque HatchXR es una excelente introducción a la Realidad Virtual, existen dos gigantes que dominan la industria: Unity[1] y Unreal Engine[2]. Estas son las plataformas más utilizadas para crear experiencias de Realidad Virtual y Videojuegos. Piensen en juegos populares como "Among Us" o "Fortnite". ¡Sí, fueron creados con una de estas plataformas! Así que, hoy, exploraremos Unity.

- *Video circuito de drones con Unity:* *https://drive.google.com/file/d/1rBBFFPV5IS2bSYpyFQWpaPbREBzz_agR/view?usp=drive_link*

En esta ocasión, nos sumergiremos en el desarrollo de un pequeño videojuego utilizando Unity. Imagina un emocionante circuito circular, en el que tomarás el control de un dron y tu misión será recolectar las monedas que están dispersas a lo largo de la pista. Este videojuego que vamos a ver en acción es un ejemplo simple de lo que puedes crear con Unity.

En las siguientes notas, encontrarás detalles sobre el código y la lógica detrás de este juego. Ahí, podrás ver cómo se ha programado cada aspecto del juego, recolección de monedas.

Para poner en marcha el juego debes utilizar el entorno de desarrollo Unity y descargar el proyecto base tal como te indicará tu instructor.

[1] https://unity.com/es
[2] https://www.unrealengine.com/es-ES

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

public class Bitcoin : MonoBehaviour
{
    public AnimationCurve myCurve;

    private float posX = 0;
    private float posZ = 0;

    void Start()
    {
        posX = transform.position.x;
        posZ = transform.position.z;
    }

    void Update()
    {
        transform.position = new Vector3(posX, myCurve.Evaluate((Time.time %
myCurve.length)), posZ);
    }

    void OnCollisionEnter(Collision collision)
    {
        Aircraft1 otherAircraft =
collision.gameObject.GetComponent<Aircraft1>();

        if (otherAircraft != null)
        {
            otherAircraft.coins += 1;
            Destroy(gameObject);
        }
    }
}
```

public AnimationCurve myCurve: Aquí se declara una variable pública llamada **myCurve**. Esta variable se utiliza para almacenar una curva de animación. Las curvas de animación son una herramienta en Unity que permite definir cómo cambian los valores con el tiempo. En este caso, **myCurve** se utilizará para controlar el movimiento vertical del objeto **Bitcoin** a lo largo del tiempo.

private float posX = 0; y **private float posZ = 0;**: Estas líneas declaran dos variables privadas llamadas **posX** y **posZ**, que se utilizan para almacenar las coordenadas X y Z iniciales del objeto **Bitcoin**.

void Start(): Esta función se llama automáticamente cuando se inicializa el objeto **Bitcoin**. En su interior, se asignan las coordenadas X y Z actuales del objeto a las variables **posX** y **posZ.** Esto permite recordar la posición inicial del objeto antes de que comience cualquier movimiento.

void Update(): La función **Update()** se ejecuta en cada fotograma del juego. Su propósito aquí es actualizar la posición vertical del objeto **Bitcoin** utilizando la curva de animación **myCurve**.

- **Time.time** representa el tiempo transcurrido desde que comenzó el juego.
- **myCurve.Evaluate(Time.time % myCurve.length)** calcula el valor de la curva en función del tiempo actual. **% myCurve.length** se usa para que la curva se repita en bucle, lo que significa que el objeto **Bitcoin** seguirá la curva una y otra vez en lugar de detenerse al final. La posición vertical del objeto se establece en función del valor de esta evaluación.

void OnCollisionEnter(Collision collision): Esta función se activa cuando el objeto **Bitcoin** colisiona con otro objeto en el juego.

- **Collision collision** representa la información de la colisión, como los objetos involucrados.
- **Aircraft1 otherAircraft = collision.gameObject.GetComponent<Aircraft1>();**: Aquí se intenta obtener una referencia al componente **Aircraft1** del objeto con el que colisionó el **Bitcoin**. Esto se hace para verificar si el objeto con el que colisionó es una aeronave (**Aircraft1**) en el juego.
- **if (otherAircraft != null)**: Se verifica si se pudo obtener una referencia válida a un objeto **Aircraft1** (es decir, si el **Bitcoin** colisionó con una aeronave).
- **otherAircraft.coins += 1;**: Si la colisión involucra a una aeronave, se incrementa la variable **coins** del objeto **Aircraft1**. Esto podría usarse para llevar un registro de las monedas recolectadas por el jugador.
- **Destroy(gameObject);**: Finalmente, el objeto Bitcoin actual se destruye, lo que simboliza que ha sido recolectado por la aeronave y ya no está presente en el juego.

¡Es hora de ponerse las gafas y experimentar! Colócatelas y sumérgete en el emocionante mundo de Unity. Controla el dron, recoge las monedas y disfruta de la experiencia de jugar un videojuego que tú mismo podrías crear en el futuro.

¡Manos a la obra y a disfrutar de esta inmersión en el mundo de Unity!

ACTIVIDADES PROPUESTAS PARA LOS ALUMNOS:

1. **Análisis de Unity**: Comenzaremos analizando el editor de Unity. Los estudiantes abrirán los archivos de código y se familiarizarán con el proceso general de un proyecto 3D en Unity

2. **Prueba de la Realidad Virtual**: Los estudiantes tendrán la oportunidad de probar la simulación del juego con las gafas de realidad virtual. Podrán experimentar en primera persona cómo se siente controlar un dron y recolectar monedas en un entorno virtual. Para que eso sea posible se deberá desplegar primero la aplicación en las gafas de realidad virtual.

3. **Desafío Adicional:** Para aquellos que busquen un desafío adicional, les sugerimos agregar un objeto 3D de tipo texto que esté dentro de la jerarquía del jugador. Este texto mostrará el número de monedas recolectadas por el jugador y deberá actualizarse dentro de la función OnCollisionEnter. Esta actividad les permitirá comprender cómo se pueden mostrar y actualizar datos en tiempo real en un juego de Unity.