Help Práctica 1 UF2_M7

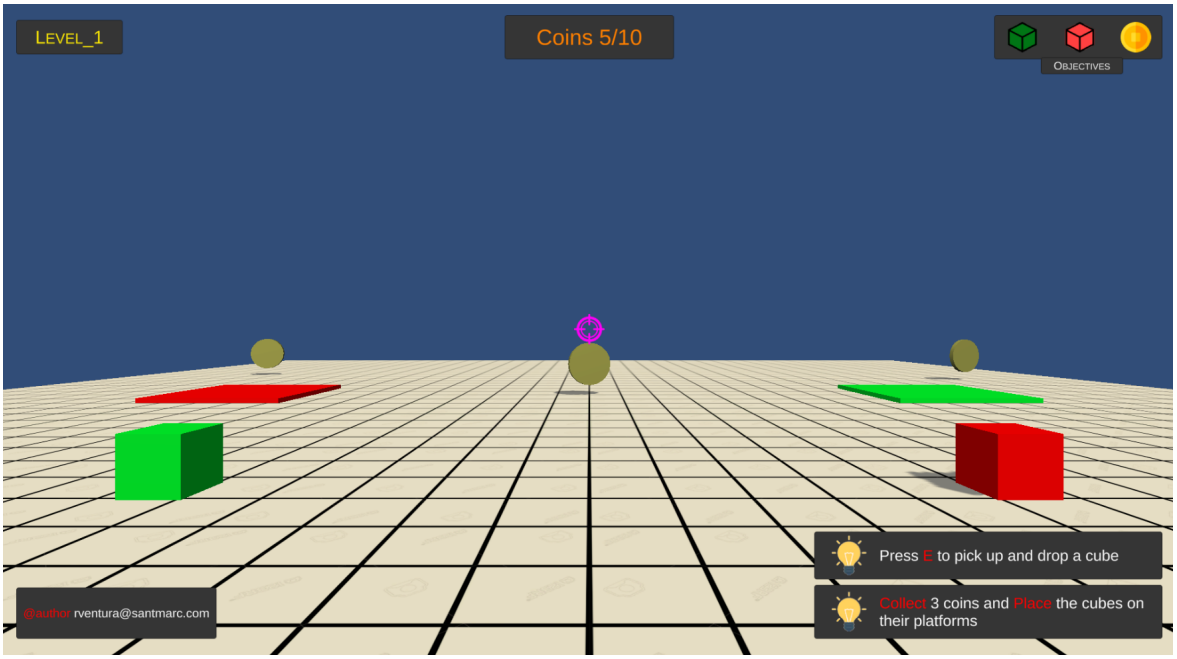
Para crear la build proporcionada se ha creado este documento con el fin de orientar al alumno con los pasos a seguir.

1. Crear un proyecto 3D de Unity (puede ser proyecto 3D normal o URP)
2. Creamos una escena donde trabajar por ejemplo con el nombre de Level_1
3. Creamos lo props que tendremos en el juego los cuales son:

    - Personaje (cápsula)
    - Cubo rojo
    - Cubo verde
    - Plataforma roja
    - Plataforma verde
    - Moneda (coleccionable)
    - Portal
    - Suelo

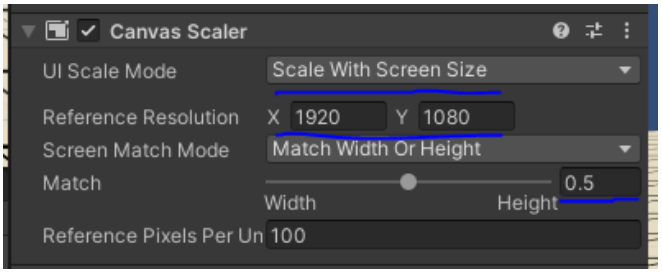
4. Montamos el level Design de nuestra escena

5. Trabajamos en la UI del juego con el fin de dejarlo como la build o la imagen de muestra

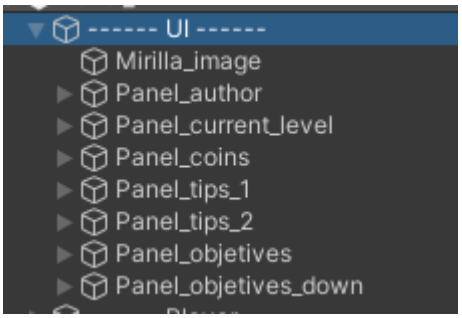


Tips UI

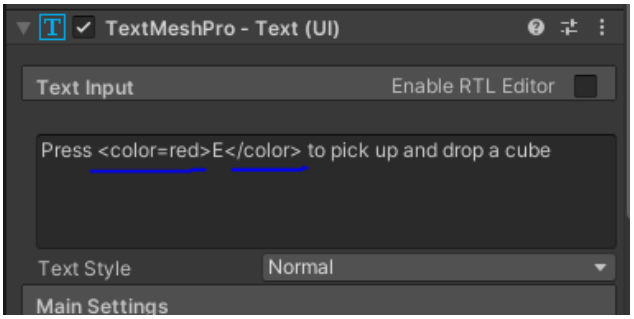
– Recordar setear los valores del canvas tal y como se muestra en la imágen

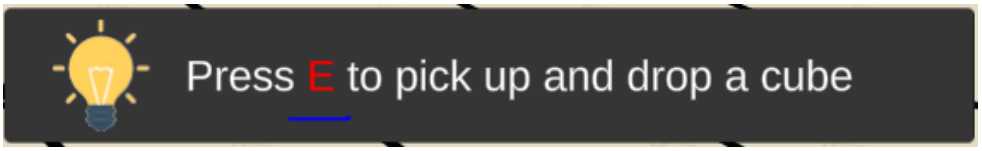


– Organizar todo por paneles para tener un mayor orden



– Para añadir color en una palabra de un texto se realiza de la siguiente manera





Creación de scripts

6. Creamos el script de **GameManager.cs**

Este script tiene que ser un Singleton (se puede acceder a su información desde otros scripts) y no tiene que destruirse entre escenas, por lo que deberéis crear una estructura principal como la que se muestra de ejemplo.

EJEMPLO

```
public class GameManager : MonoBehaviour
{

    public static GameManager instance;


    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }

}
```

Este script lo incluimos dentro de un empty en nuestra jerarquía de Unity (por tener un buen orden podemos llamar al empty con el mismo nombre que el script)

** Luego tendréis que añadir más variables públicas en función de las necesidades que tengáis como por ejemplo una variable para guardar el tiempo transcurrido.

7. Creamos el script de **LevelManager.cs**

Este script tiene que tener el control de todos los elementos tanto de la UI como los prefabs que están (activos o no activos) dentro de la escena. Por lo que tendréis que tener varias variables públicas, para tener referencia de todos los elementos que queréis modificar o acceder a su información.

Por otra parte en el método Start() {} tendréis que indicar al script **GameManager.cs** quien es el LevelManager activo en ese momento (esto lo hacemos ya que la información que tiene el **GameManager.cs** es accesible desde otros scripts, por lo tanto si tiene información del level activo podremos acceder también a información de dicho nivel aunque en este caso sólo tengamos uno, para futuros proyectos os puede interesar).

EJEMPLO

```
public class LevelManager : MonoBehaviour
{
    [Header("---- LEVEL 1 ----")]
    [SerializeField] private GameObject check_icon_cube_green;
    [SerializeField] private GameObject check_icon_cube_red;
    [SerializeField] private GameObject check_icon_cube_coins;

    public GameObject portal;
    public bool isGreenCubeOK;
    public bool isRedCubeOK;

    public TMP_Text text_counter_coins;
    public int maxCoins;
    public int currentCoins;

    private void Start()
    {
        GameManager.instance.currentLevelManager = this;
        GameManager.instance.coins = currentCoins;
        text_counter_coins.text = "<color=orange>Coins " + currentCoins + "/"
+ maxCoins + "</color>";

    }
}
```

Este script lo incluimos dentro de un empty en nuestra jerarquía de Unity (por tener un buen orden podemos llamar al empty con el mismo nombre que el script)

Este script deberá tener diferentes métodos los cuales tendrán que gestionar las siguientes lógicas.

- Comprobar si los objetivos están completados con el fin de mostrar un check en la UI

El check es un elemento de la UI por lo tanto es un **GameObject** que podéis tener como variable publica en el script, de esta manera podréis activarlo o desactivarlo según si el objetivo está completado o no

EJEMPLO VARIABLE

```
public GameObject check_icon_cube_green;
```

EJEMPLO ACTIVAR/DESACTIVAR UN OBJETO

```
check_icon_cube_green.SetActive(true); // Activar
check_icon_cube_green.SetActive(false); // Desactivar
```

Para comprobar si los objetivos están completados mi consejo es que lo verifiquéis esta pregunta en un If con el uso de bools

EJEMPLO

```
if (isGreenCubeOK)
{
    check_icon_cube_green.SetActive(true);
}
else
{
    check_icon_cube_green.SetActive(false);
}
```

La otra verificación tendrá que ser si se ha alcanzado el número de monedas como objetivo del nivel

- Tendréis que crear una lógica que si los bools están activos y las monedas están conseguidas active el **GameObject** que es el prefab del **Portal**

EJEMPLO

```
if (isGreenCubeOK && isRedCubeOK && currentCoins == maxCoins)
{
    portal.SetActive(true);
} else
{
    portal.SetActive(false);
}
```

- Crear un método para aumentar el contador de monedas del nivel y actualizar la información en el GameManager

** Tendréis que añadir más variables públicas en función de las necesidades que tengáis.

8. Creamos el script de **Platform.cs**

En este script tendremos que crear la lógica para detectar si los cubos están encima de la plataforma que toca.

Para ello tendremos que saber que tag es el que esperamos que esté colisionando con la plataforma, para luego revisarlo en los métodos **OnTriggerEnter() {} OnTriggerExit() {}**

Os dejo un breve ejemplo de cómo plantear el script sin la lógica completa.

EJEMPLO

```
public class Platform : MonoBehaviour
{
    public string target_tag;

    private void OnTriggerEnter(Collider other)
    {
        if (other.tag.Equals(target_tag))
        {
            // Lógica si es el tag…
        }
    }


    private void OnTriggerExit(Collider other)
    {
        if (other.tag.Equals(target_tag))
        {
            // Lógica si es el tag…
        }

    }
}
```

** Tendréis que añadir más variables públicas en función de las necesidades que tengáis.

9. Creamos el script de **Coin.cs**

En este script tendréis que crear la lógica que sucederá cuando el jugador choque con una moneda. La cual es:

- Verificar que colisiona con el Player (mediante el método OnTriggerEnter() y revisando el tag del Player)
- Incrementar el contador de monedas del Level Design (llamando al método que se ha creado en el script de **LevelManager.cs**)
- Destruir la moneda (Con el método Destroy())

Por otra parte la moneda tiene que estar rotando tal y como se muestra en la build

10. Creamos el script de **Portal.cs**

En este script tendréis que crear la lógica que sucederá cuando el jugador choque con el portal, una vez completados los objetivos

- Verificar que colisiona con el Player (mediante el método OnTriggerEnter() y revisando el tag del Player)
- Parar el contador del GameManager
- Enviar al jugador a la otra escena donde se muestra el tiempo transcurrido desde iniciar el juego.

EJEMPLO TRANSFORMAR FLOAT TIEMPO EN TEXTO

```
        totalTime = GameManager.instance.timeToCompleteGame;

        minutes = (int) (totalTime / 60);
        seconds = (int) (totalTime - minutes * 60);
        cents = (int) ((totalTime - (int) totalTime) * 100f);

        timer_txt.text = string.Format("{0:00}:{1:00}:{2:00}", minutes,
seconds, cents);
```

11. Script de **PlayerController.cs**

Este script se proporciona en el .zip de la práctica pero lo dejo como apuntes en este documento

```
using UnityEngine;

public class PlayerController : MonoBehaviour
{

    [Header("Character Variables")]
    public float speed;
    public Camera playerCamera;
    public Transform targetPoint;
    public Transform respawn;

    [Space(10)]
    [Header("Pickups Variables")]
    public bool haveAPickup;
    private GameObject pickup;
    public float rayCastRange;
    public LayerMask pickupMask;


    // Private Variables
    private float horizontalInput;
    private float verticalInput;



    void Update()
    {
        // Accedemos a los axis horizontales y verticales
        horizontalInput = Input.GetAxis("Horizontal");
        verticalInput = Input.GetAxis("Vertical");

        // Lógica para mover al jugador
        transform.Translate(Vector3.forward * (speed * verticalInput *
Time.deltaTime));
        transform.Translate(Vector3.right * (speed * horizontalInput *
Time.deltaTime));



        // Lógica para controlar que el jugador no caiga de cierta altura y
```

```
vuelva a un respawn
        if (transform.position.y <= -0.5f)
        {
            transform.position = respawn.transform.position; // Movemos la
posición del jugador a la de un respawn
        }


        // Si el jugador pulsa la letra E realizaremos la siguiente lógica
        if (Input.GetKeyDown(KeyCode.E))
        {
            // Primero miramos si nuestra variable bool está a False, indicando
que no tenemos un objeto en ese momento
            if (!haveAPickup)
            {
                // Creamos una variable de tipo RaycastHit
                RaycastHit hit;

                // Con la función Physics.Raycast no devolverá un true o false con
lo parámetros que le pasamos
                // En caso que el resultado sea true entraremos en el IF
                if (Physics.Raycast(playerCamera.transform.position,
playerCamera.transform.forward, out hit, rayCastRange, pickupMask))
                {

                    haveAPickup = true; // Indicamos que tenemos un objeto por si
pulsamos de nuevo la E que la lógica sea diferente
                    hit.transform.position = targetPoint.transform.position; // El
objeto que hemos impactado pasa a una posición indicada
                    hit.transform.parent = targetPoint; // Indicamos que el objeto
que hemos impactado ahora es hijo nuestro
                    hit.rigidbody.isKinematic = true; // Activamos isKinematic del
objeto impactado para evitar que le afecte la gravedad
                    pickup = hit.transform.gameObject; // Guardamos en nuestra
variable Pickup la información del objeto con el que impactamos.
                }
            }
            else
            {
                haveAPickup = false; // Indicamos que ya no tenemos un objetos
por si pulsamos de nuevo la E volvamos a la lógica anterior
                pickup.transform.parent = null; // Él objetos que teníamos ya no es
nuestro hijo por lo que no seguirá nuestro movimiento
                pickup.GetComponent<Rigidbody>().isKinematic = false; //
Desactivamos la propiedad isKinematic del objeto para que le afecte la
```

```
gravedad de nuevo
          }

       }

   }

}
```

12. Script de **CameraLook.cs**

Este script se proporciona en el .zip de la práctica pero lo dejo como apuntes en este documento

```
using UnityEngine;

public class CameraLook : MonoBehaviour
{

    // Variables
    public float mouseSensiblity = 80f;
    public Transform playerBody;

    private float mouseX;
    private float mouseY;
    private float xRotation = 0;


    void Update()
    {
        // Accedemos a los axis del mouse
        mouseX = Input.GetAxis("Mouse X");
        mouseY = Input.GetAxis("Mouse Y");

        // Lógica para controlar el movimiento de la cámara de arriba-abajo
        xRotation -= mouseY;

        xRotation = Mathf.Clamp(xRotation, -70f, 70f); // Definimos lo que podemos subir máximo y lo que podemos bajar máximo

        transform.localRotation = Quaternion.Euler(xRotation,0 ,0); // Aplicamos esa rotación a la camara

        // Rotamos al cuerpo de nuestro jugador de izquierda y derecha
        // Como nuestro cuerpo es padre de la cámara se moverá también
        playerBody.Rotate(Vector3.up * mouseX * mouseSensiblity * Time.deltaTime);

    }
}
```