Unity 2020
Scripts

Introducción

Cuadro 1

1 Banda sonora
2 Premios
3 Ocultar cubos de opciones

Cuadro 2

1 Colisionador
2 Temporizador

Cuadro 3

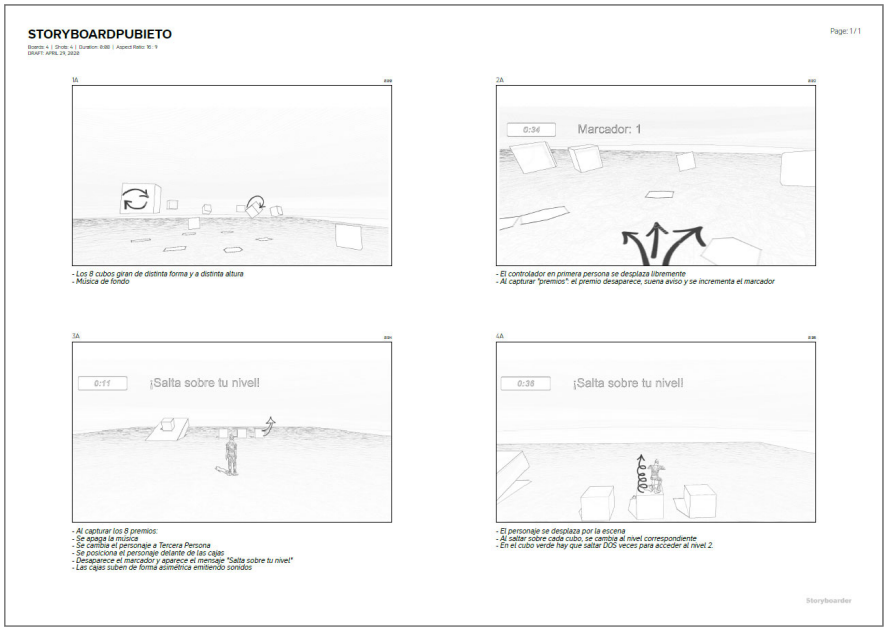
1 Acciones desde el controlador
2 Acciones en las Opciones

Cuadro 4

1 Detectar presencia
2 Cambio de escena
3 Build Settings

0 Introducción

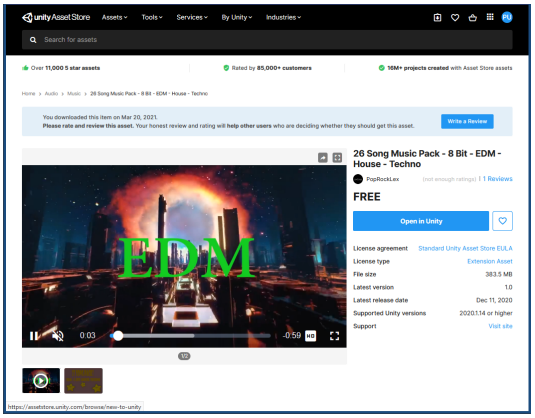
- En este tema vamos a definir una serie de Scripts que añadiremos a los GameObjects para darles nuevas funcionalidades.
- Tomamos como referencia el *storyboard* definido en el tema anterior:



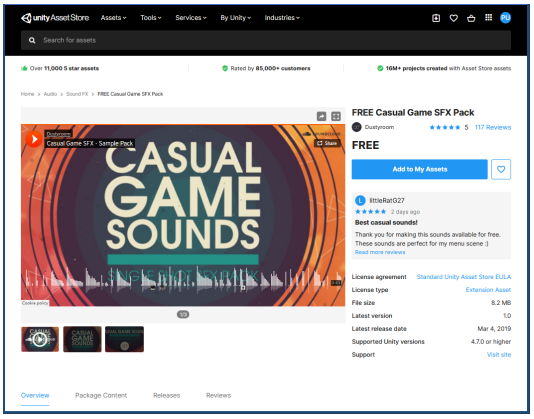
0 Introducción
Recursos Utilizados

- En los scripts hemos utilizado los siguientes recursos de Audio (se puede utilizar cualquier otro)

  – Para la banda sonora:
  26 song Music Pack



  – Para los sonidos de la aplicación (captura premio, aparición de opciones)
  Casual Game Sounds



Unity - Scripts

Cuadro 1

- Directrices del cuadro 1:
  1. Música de fondo. Definimos la banda sonora a la aplicación. Añadimos también un script con conmutador, para poder activar/desactivar la banda sonora.
  2. Añadimos los premios, con posiciones y movimientos aleatorios.
  3. No está en el *storyboard*, pero debemos ocultar los cubos de las opciones

Cuadro 1. Banda Sonora

1.1 Añadir banda sonora

- Para añadir la banda sonora a la aplicación:

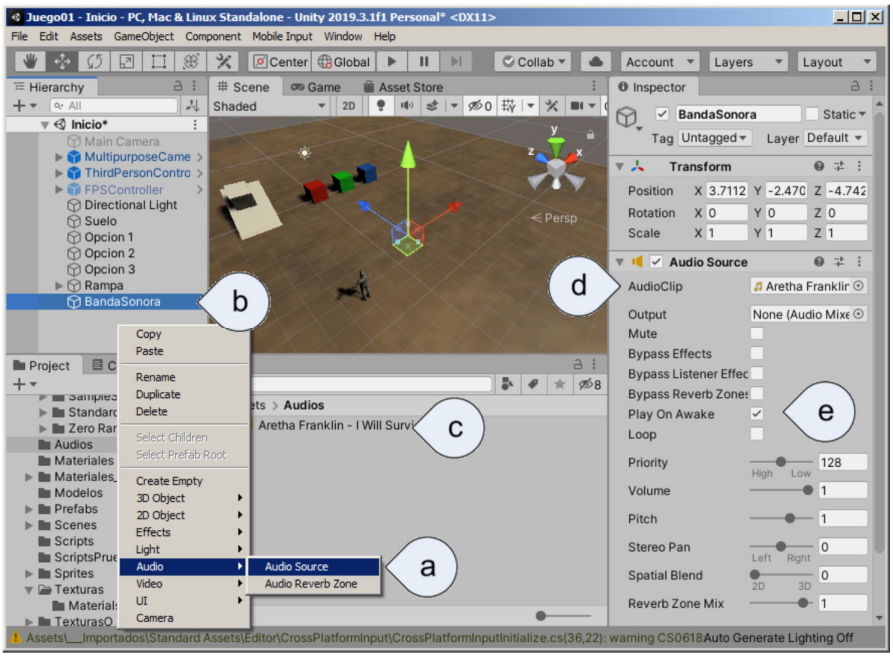
a) Crear un *Game Object* nuevo de tipo ***Audio Source***

b) Renombrarlo como *BandaSonora*

c) Buscar un archivo de audio en el proyecto
   - Es necesario tener el permiso de reproducción

d) Arrastrar el fichero de audio al control ***Audio Clip*** del componente ***Audio Source***

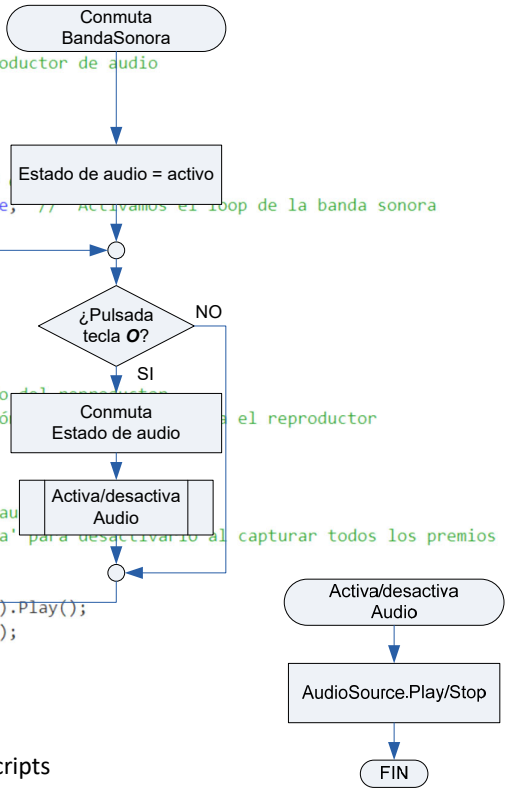
e) Activar el control ***Play On Awake***
   - Para que se empiece a reproducir al iniciar la aplicación



Cuadro 1. Banda Sonora

1.2 Script para conmutar la banda sonora

```
4
5   public class BandaSonoraConmuta : MonoBehaviour
6   {
7       private bool audioOn;  // Contiene el estado del reproductor de audio
8
9       // Start is called before the first frame update
10      void Start()
11      {
12          audioOn = true;   //  Por defecto, el reproductor
13          gameObject.GetComponent<AudioSource>().loop = true;  //  Activamos el loop de la banda sonora
14      }
15
16      // Update is called once per frame
17      void Update()
18      {
19          if (Input.GetKeyDown(KeyCode.O))
20          {
21              audioOn = ! audioOn;  //  Conmutamos el estado
22              ActivaSonido(audioOn);//  Llamamos a la funció                el reproductor
23          }
24      }
25
26      //  Función que activa o desactiva el reproductor de au
27      //  Es pública, ya que se accede desde 'PremiosCaptura'                capturar todos los premios
28      public void ActivaSonido (bool estado)
29      {
30          if (estado) gameObject.GetComponent<AudioSource>().Play();
31          else gameObject.GetComponent<AudioSource>().Stop();
32
33          audioOn = estado;
34
35      }
36  }
```



Cuadro 1. Premios

2.1 Añadir cubos *Premio*

- Para añadir los Premiso a la escena

a) Crear un cubo (objeto 3D) y renombrarlo a *Premio*

b) Poner los datos de ***Transform*** de la imagen

c) Definirlo como ***prefab*** (arrastrarlo a la carpeta del proyecto para prefabs

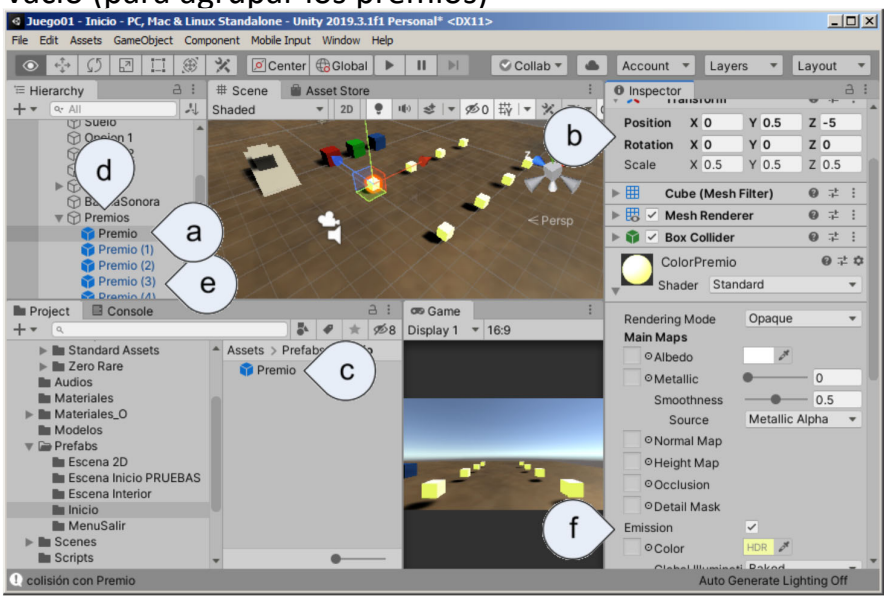
d) Crear un *GameObject* vacío (para agrupar los premios)

   – Su Transform debe ser lo más neutro posible

e) Copiar y posicionar los ocho premios

f) Asignar un material emisor de luz a los premios

   – Para que sean fácilmente visibles en la escena del laberinto



Cuadro 1. Premios

2.2 Definir TAGS

- Los TAGS son etiquetas que se pueden asignar a los *Game Objects* para hacer más fácil su identificación y búsqueda desde los scripts.

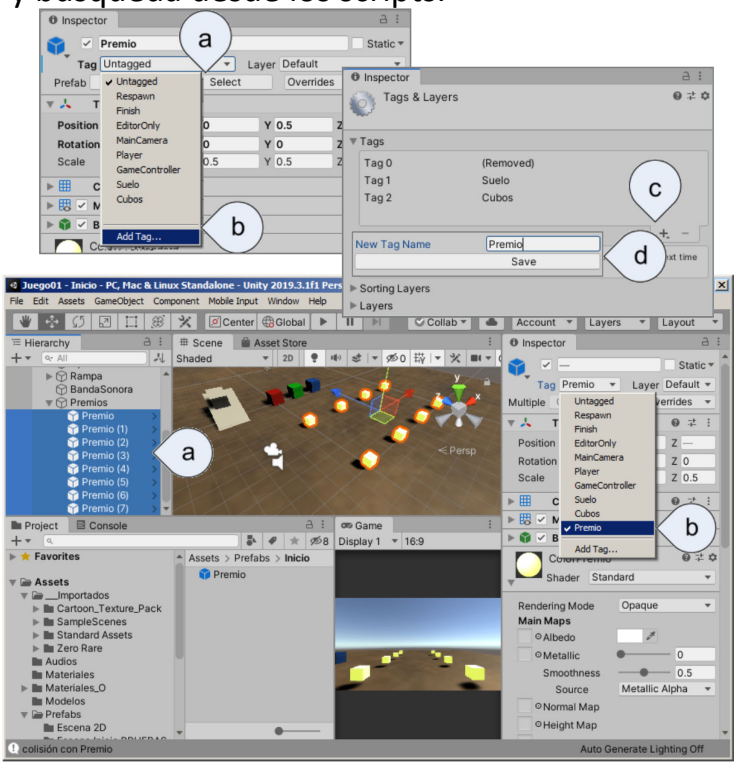
- Primero hay que crear el Tag:

a) Desplegar la lista de Tags

b) Pinchar en la opción Add Tag…

c) Pinchar en el + para añadir Tag

d) Añadir el nombre del Tag



- Para asignar el Tag a los elementos

a) Seleccionar los elementos

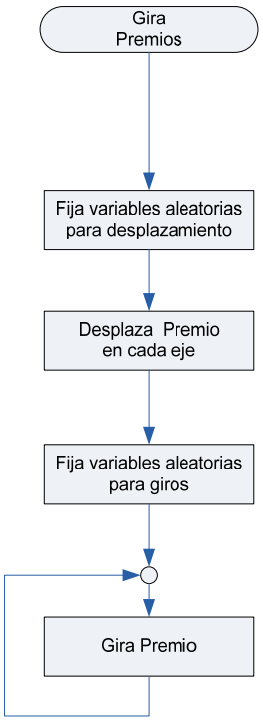
b) Elegir el Tag de la lista desplegable

Generalmente, es mejor asignar el Tag al Prefab, en vez de a sus instancias

Cuadro 1. Premios

2.3 Script para diferenciar Premios

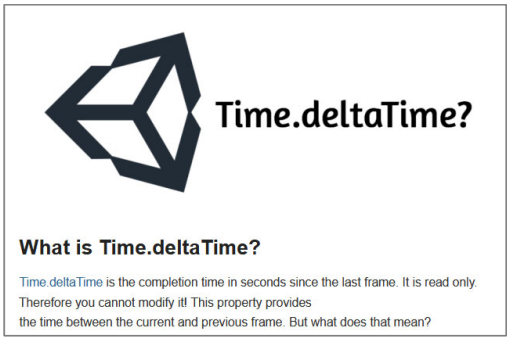
```
5   public class PremiosGira : MonoBehaviour
6   {
7       float posx, posy, posz;
8       float girox, giroy, giroz;
9
10      // Start is called before the first frame update
11      void Start()
12      {   //  Defino los desplazamientos random para cada eje
13          posx = Random.Range(-1.0f, 1.0f);
14          posy = Random.Range(0.0f, 1.0f);
15          posz = Random.Range(-2.0f, 2.0f);
16
17          //  Cambio la posición en cada uno de los ejes
18          gameObject.transform.position += Vector3.left * posx;
19          gameObject.transform.position += Vector3.up * posy;
20          gameObject.transform.position += Vector3.forward * posz;
21
22          //  Defino los giros random respecto de cada eje
23          girox = Random.Range(0.0f, 5.0f);
24          giroy = Random.Range(5.0f, 15.0f);
25          giroz = Random.Range(0.0f, 3.0f);
26      }
27
28      // Update is called once per frame
29      void Update()
30      {
31          //  Giro el cubo respecto a cada uno de los ejes
32          gameObject.transform.Rotate(girox, giroy, giroz);
33      }
34  }
35
```



Cuadro 1. Premios

2.4 Ajustar Script a la velocidad del dispositivo

```
1   /*********************************************/
2   /* Premios Gira                              */
3   /* Explicado en Cuadro 1. Punto 2.3)         */
4   /*********************************************/
5   using System.Collections;
6   using System.Collections.Generic;
7   using UnityEngine;
8
9   public class PremiosGira : MonoBehaviour
10  {
11      private float posX, posY, posZ;
12      private float giroX, giroY, giroZ;
13
14      public int VelGiro = 10; // Factor de multiplicación para regular velocidad
15                               // Lo hacemos público para poder cambiarlo interactivamente
16
17
18      // Start is called before the first frame update
19      void Start()...
37
38      // Update is called once per frame
39      void Update()
40      {
41          // Añado Time.deltatime al factor de multiplicación de la velocidad
42          // Así nos adaptamos a la velocidad del dispositivo
43          float Rotacion = Time.deltaTime * VelGiro;
44
45          //  En cada frame: giro el cubo respecto a cada uno de los ejes
46          gameObject.transform.Rotate(giroX * Rotacion, giroY * Rotacion, giroZ * Rotacion);
47
48      }
49  }
```



Cuadro 1. Ocultar Opciones

3.1 Script para Ocultar Opciones

- Para ocultar las opciones en el cuadro inicial, vamos a optimizar un poco la escena:

a) Creamos un *Empty Object*, llamado *Opciones*, del que colgar las tres opciones

b) Definimos el cubo como *prefab*

c) Añadimos el script al *prefab*

d) Copiamos y situamos las tres opciones

- El script únicamente modifica la altura de la *Opción*:

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

public class OpcionesPresenta : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        gameObject.transform.position += Vector3.down * 1.5f;
    }

    // Update is called once per frame
    void Update()
    {
    }
}
```


Cuadro 2

- Directrices del cuadro 2:
  1. El controlador se desplaza libremente por la escena colisionando con los premios.
  2. Al colisionar con los premios, los captura:
     - El premio desaparece, suena un aviso acústico y se incrementa el marcador
  3. Se visualiza un cronómetro
     - Cuando pase un tiempo, se indica cambiando a color rojo

Cuadro 2. Colisionador

1.1 Eliminar premio capturado

- Para detectar las colisiones con el controlador en primera persona, utilizamos un método propio todo *Collider*: *OnControllerColliderHit*
  - Esta función se ejecuta cada vez que el controlador colisiona con algo

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


public class PremiosCaptura : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
    }

    void OnControllerColliderHit(ControllerColliderHit loColisionado)
    {

        string etiqueta = loColisionado.collider.gameObject.tag;  // Guardamos en 'etiqueta' el tag del objeto que ha colisionado


        if (etiqueta == "Premio")  //  Actuación en caso de colisión con un Premio
        {
            Debug.Log("colisión con " + etiqueta); //  Aviso para debug. Se puede desactivar

            Destroy(loColisionado.collider.gameObject);
        }
    }
}
```


Cuadro 2. Colisionador

1.2 Emitir sonido al colisionar

- Al colisionar se activa el *Audio Source* del controlador
  - El clip de sonido se guarda en una variable pública, accesible desde el Inspector

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

public class xPremiosCaptura : MonoBehaviour
{
    public AudioClip pickupSound; //  Variable en la que meter el nombre del fichero de sonido
                                  //  Como es public, aparece un campo editable en la ventana de Inspector
                                  //  También se podía poner [SerializeField] y definirla como private
                                  //  [HideInInspector] permite ocultar variables public en el Inspector
    // Start is called before the first frame update
    void Start() { }
    // Update is called once per frame
    void Update() { }

    void OnControllerColliderHit(ControllerColliderHit loColisionado)
    {
        string etiqueta = loColisionado.collider.gameObject.tag;  // Guardamos en 'etiqueta' el tag del objeto que ha colisionado

        if (etiqueta == "Premio")  //  Actuación en caso de colisión con un Premio
        {
            Debug.Log("colisión con " + etiqueta); //  Aviso para debug. Se puede desactivar

            /***  Emitimos sonido al capturar premio   ***/
            gameObject.GetComponent<AudioSource>().clip = pickupSound;  //  Leemos el fichero indicado en el componente
            gameObject.GetComponent<AudioSource>().volume = 1.0f;       //  Fijamos el volumen (de 0 a 1)
            gameObject.GetComponent<AudioSource>().Play();              //  Activamos el sonido
        }
    }
}
```

Cuadro 2. Colisionador
1.3 Incrementar el marcador

- Como marcador usamos un variable privada de tipo *int*
  - Cada vez que colisiona con un premio, se incrementa el valor de la variable
  - Creamos una variable tipo string en la que guardar el texto del mensaje

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

public class xPremiosCaptura : MonoBehaviour
{
    int iMarcador;
    string msgMarcador; // Mensaje de texto para presentar en pantalla

    // Start is called before the first frame update
    void Start()
    {
        iMarcador = 0;  //  Ponemos marcador a cero
    }

    // Update is called once per frame
    void Update() { }

    void OnControllerColliderHit(ControllerColliderHit loColisionado)
    {
        string etiqueta = loColisionado.collider.gameObject.tag;  // Guardamos en 'etiqueta' el tag del objeto que ha colisionado

        if (etiqueta == "Premio")  //  Actuación en caso de colisión con un Premio
        {
            /***  Actualizamos marcador   ***/
            iMarcador = iMarcador + 1;              // Incrementamos iMarcador. También se puede poner iMarcador++;
            msgMarcador = "Marcador: " + iMarcador; // Generamos mensaje a presentar en el marcador de la interfaz

            Debug.Log(msgMarcador); //  Aviso para debug. Valor del marcador
        }
    }
}
```


Cuadro 2. Cronómetro
2.1 Control del tiempo

- Para controlar el tiempo de juego, se utiliza *Time.deltaTime*
  - Devuelve el tiempo transcurrido desde la última *frame*

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

public class GeneralCronómetro : MonoBehaviour
{
    float tiempo, tiempoAnt;
    int minutos, segundos;
    string elTiempo, sSegundos;

    // Start is called before the first frame update
    void Start()
    {   tiempo = tiempoAnt = 0.0f;
    }

    // Update is called once per frame
    void Update()
    {
        tiempo = tiempo + Time.deltaTime; // Al tiempo acumulado, le sumamos el tiempo que ha pasado desde la última frame.

        if (tiempo > (tiempoAnt + 1))
        {   tiempoAnt = tiempo;
            minutos  = (int)tiempo / 60;
            segundos = (int)tiempo % 60;

            sSegundos = "" + segundos;
            if (segundos < 10) { sSegundos = "0" + segundos; }

            elTiempo = minutos + ":" + sSegundos;
            Debug.Log("Tiempo: " + elTiempo);
        }
    }
}
```

Cuadro 3

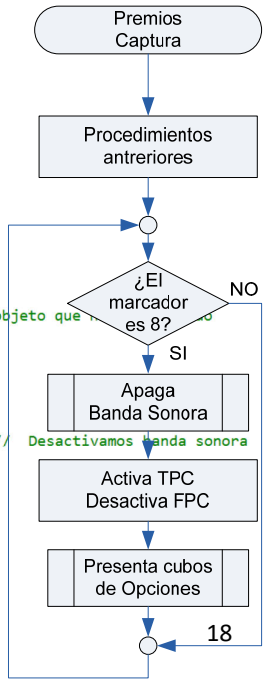
- Directrices del cuadro 3:
  1 Al capturar los ocho premios:
    - Se apaga la banda sonora
    - Se cambia el personaje a tercera persona
    - Se posiciona el personaje delante de las cajas de las opciones
    - Las cajas suben de forma asimétrica emitiendo un sonido
    - Desaparece el marcador y aparece el mensaje "*Salta sobre tu nivel*"

Cuadro 3. Acciones desde el controlador

1.1 Acciones al capturar los 8 premios

- Al capturar el último (octavo) premio, se realizan las siguientes acciones desde el script del controlador:

```
5   public class PremiosCaptura : MonoBehaviour
6   {
7       int iMarcador;
8       public GameObject ControlIni, ControlFin;  //  Variables con los dos controladores
9
10      // Start is called before the first frame update
11      void Start()
12      {   iMarcador = 0;  //  Ponemos marcador a cero
13      }
14
15      // Update is called once per frame
16      void Update() { }
17
18      void OnControllerColliderHit(ControllerColliderHit loColisionado)
19      {   string etiqueta = loColisionado.collider.gameObject.tag;  // Guardamos en 'etiqueta' el tag del objeto que
20
21          if (etiqueta == "Premio")  //  Actuación en caso de colisión con un Premio
22          {   iMarcador ++;          // Incrementamos iMarcador ;
23
24              if (iMarcador == 8)
25              {   GameObject.Find("BandaSonora").GetComponent<BandaSonoraConmuta>().ActivaSonido (false); //  Desactivamos banda sonora
26
27                  ControlFin.SetActive(true);   // Activamos controlador en tercera persona
28                  ControlIni.SetActive(false);  // Desactivamos controlador en primera persona
29
30                  GameObject.Find("Opcion 1").GetComponent<OpcionesPresenta>().Presenta();
31                  GameObject.Find("Opcion 2").GetComponent<OpcionesPresenta>().Presenta();
32                  GameObject.Find("Opcion 3").GetComponent<OpcionesPresenta>().Presenta();
33              }
34          }
35      }
```



Cuadro 3. Acciones desde el controlador

1.2 Tratar conjuntos de *Game Objects*

- Vamos a ver como tratar un conjunto (matriz) de *Game Objects*.
  - Para *acortar* el proceso de captura de premios durante los vídeos, voy a reducir el número de premios a capturar a 2 de los 8, eliminando de forma automática los 6 restantes
  - Vamos a capturar todos los premios (con *TAG* = "Premio") en una matriz
  - Vamos a recorrer la matriz eliminando uno a uno los *Game Objects*

```
73  if (iMarcador == 2)
74  {
75
76      GameObject[] matrizPremios;  //  Definimos una variablo como matriz de GameObjects
77
78      matrizPremios = GameObject.FindGameObjectsWithTag("Premio"); //  Buscamos todos los GameObjects que tienen el tag "Premio"
79
80      foreach (GameObject unPremio in matrizPremios)  //  Analizamos, uno a uno, los GameObjects de la matriz matrizPremios
81      {
82          Destroy(unPremio);  //  Borramos el GameObject que tiene el tag "Premios"
83      }
84
85
86      GameObject.Find("BandaSonora").GetComponent<xBandaSonoraConmuta>().ActivaSonido (false); //  Desactivamos banda sonora
87
88
89      ControlFin.SetActive(true);
90      ControlIni.SetActive(false);
```


Cuadro 3. Acciones en las Opciones

2 Presentar opciones

```
5   public class xOpcionesPresenta : MonoBehaviour
6   {   bool aparece, haysonido;
7
8       float maxY, incY, elpitch;
9       float posx, posy, posz;
10      public AudioClip pickupSound; //  Variable en la que meter el nombre del fichero de sonido
11
12      // Start is called before the first frame update
13      void Start()
14      {   gameObject.transform.position += Vector3.down * 1.5f;  //  Bajamos el cubo por debajo de la superficie
15          aparece = haysonido = false;
16          maxY = 0.5f; //  Altura máxima a la que sube el cubo (0.5 para que esté sobre el suelo)
17          incY = Random.Range(0.01f, 0.05f);  //  Calculamos un incremento aleatorio (para dar varias velocidades)
18          elpitch = Random.Range(0.5f, 3f);  //  Calculamos un pitch aleatorio (para dar varios tonos)
19      }
20
21      // Update is called once per frame
22      void Update()
23      {
24          if (aparece)
25          {   posy = gameObject.transform.position.y;
26              if (posy < maxY)
27              { transform.position += Vector3.up * incY; }
28              else
29              {   aparece = false;
30                  gameObject.GetComponent<AudioSource>().Stop(); }           // Paramos el sonido
31
32              if (! haysonido) //  Emitimos sonido mientras sube
33              {   gameObject.GetComponent<AudioSource>().clip = pickupSound;  //  Leemos el fichero indicado en el componente
34                  gameObject.GetComponent<AudioSource>().volume = 1.0f;       //  Fijamos el volumen (de 0 a 1)
35                  gameObject.GetComponent<AudioSource>().pitch = elpitch;     //  Cambiamos timbre (de 0 a 1)
36                  gameObject.GetComponent<AudioSource>().loop = true;         //  Activamos loop, por si sube muy despacio
37                  gameObject.GetComponent<AudioSource>().Play();              //  Activamos sonido
38                  haysonido = true;  //  Para que no empiece en cada loop
39              }
40          }
41      }
42
43      public void Presenta ()
44      {   aparece = true;
45      }
46
```

Cuadro 4

- Directrices del cuadro 4:
  1. El personaje se desplaza por la escena
  2. Debe saltar sobre los cubos de las opciones para cambiar de escena
     - En el cubo verde debe saltar dos veces para cambiar de escena

Cuadro 4. Detectar presencia

1.1 Definir *Collider*

- Para detectar que el jugador ha saltado sobre uno de los cubos de opciones, hay que definir una superficie que detecte su presencia

a) Editamos el prefab de las opciones

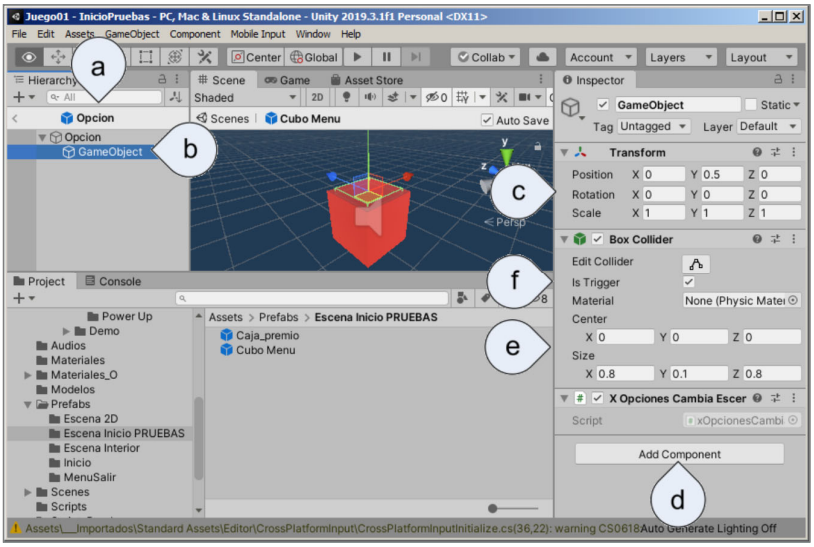
b) Creamos un empty Game Object

c) Lo posicionamos en la parte superior del cubo

d) Añadimos componente ***Box Collider***

e) Definimos la superficie del *collider* un poco más pequeña que el cubo
   - Si no, se activaría al tocar lateralmente

f) Activar el control ***Is Trigger***



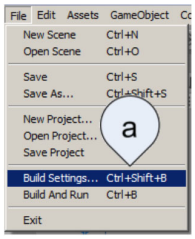
Cuadro 4. Cambio de escena

1.2 Script para cambiar de escena

```
using UnityEngine.SceneManagement;  //  para cambiar de escena

public class xOpcionesCambiaEscena : MonoBehaviour
{   string nombreEscena;
    int saltos, saltosNecesarios;
    bool contado;

    // Start is called before the first frame update
    void Start()
    {   saltos = 0;
        contado = false;
    }

    private void OnTriggerEnter(Collider loColisionado)
    {   if (loColisionado.tag == "Player")
        {   Debug.LogWarning(gameObject.transform.parent.name); //  resentamos en el Debug el nombre del objeto "Padre" para comprobar

            if (gameObject.transform.parent.name == "Opcion 1")  //  Guardamos los datos según la opción en la que estemos
            { saltosNecesarios = 1; nombreEscena = "InicioPruebas"; }

            else if (gameObject.transform.parent.name == "Opcion 2")
            { saltosNecesarios = 2; nombreEscena = "Interior"; }

            else if (gameObject.transform.parent.name == "Opcion 3")
            { saltosNecesarios = 1; nombreEscena = "Exterior"; }

            if (!contado)
            {   saltos++;   //  Incrementamos el número de saltos dados
                contado = true; //  Activamos la variable contado, para no volver a contar saltos hasta que no salga del collider
            }

            if (saltos >= saltosNecesarios)
            {   SceneManager.LoadScene(nombreEscena); // Si hemos datos los saltos necesarois, cambiamos de escena
            }
        }
    }

    private void OnTriggerExit(Collider _col)
    {   if (_col.tag == "Player")
        {   contado = false; // Si salimos del collider, desbloqueamos la opción de contar saltos
        }
    }
}
```


Cuadro 4. Build Settings

1.3 Definir las escenas que forman la aplicación



- Es necesario definir la lista de escenas que van a formar parte de nuestra aplicación:

a) Menu ***File*** → ***Build Settings***.

b) La ventana permite definir los parámetros de la aplicación a generar (Build)

c) Seleccionamos por orden las escenas del proyecto

d) Arrastramos las escenas a la ventana de ***Build Settings***

- Se pueden desplazar para cambiar el orden
- Con el menú contextual se pueden eliminar, añadir…

e) La ventana se cierra sin la necesidad de generar (Build)