

# Unity

## Proyecto: “Rodando sin límites”

Escenas usadas y contenido.....	2
Escena 1: MainMenu.....	2
Escena 4: EscenaFinal.....	3
Escena 2: ‘Juego’.....	4
1. Pelota.....	4
2. Obstáculos.....	7
3. Paredes.....	8
4. LimiteFinal.....	8
Escena 3: ‘Nivel2’.....	9
• Coche.....	10
• Valla altas.....	10
• Elementos de circuito.....	10
• Rampa.....	10
• Otros elementos de construcción y circuito.....	10
Mecánica del Juego.....	11
Entorno:.....	11
Reglas:.....	11
Objetivo:.....	11
Sistema de Puntuación:.....	12
Instrucciones de manejo del Juego.....	12
Elementos utilizados.....	12
Problemas Surgidos en el Proceso.....	13

El objetivo del juego es guiar una pelota que avanza automáticamente, recogiendo monedas y esquivando obstáculos hasta llegar a la meta.

Características principales:

1. Dos niveles de juego:

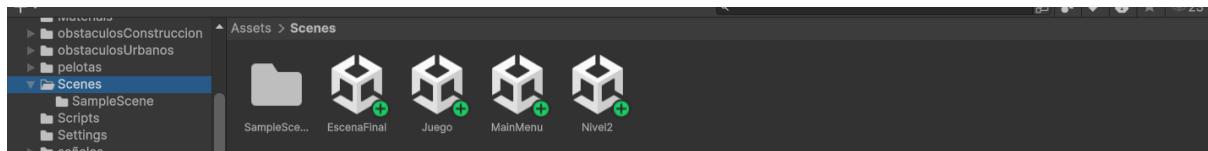
- Nivel 1: Un recorrido sencillo diseñado para que el jugador se familiarice con los controles

- Nivel 2: Un circuito más complejo, con obstáculos y salto.
- 2. Controles del jugador:
  - Tecla ‘A’: Desplaza la pelota hacia la izquierda
  - Tecla ‘D’: Desplaza la pelota hacia la derecha
- 3. Movimiento continuo
  - La pelota avanza automáticamente hacia adelante
  - La velocidad aumenta de forma progresiva, añadiendo dificultad al juego

El jugador debe llegar a la meta en cada nivel, esquivando obstáculos y evitando chocar con objetos, mientras recoge la mayor cantidad de monedas.

## Escenas usadas y contenido

Las escenas que vayamos a usar por cuestión de organización deberán de situarse en una carpeta dentro de nuestros Assets con el nombre Scene en nuestro caso.



### Escena 1: MainMenu

Esta escena está compuesta por una imagen de fondo y dos botones, un primer botón que nos llevará al juego y otro botón con el que podremos salir del juego.

- Para colocar la imagen de fondo deberemos de:
  1. Importar la imagen en una carpeta dentro de nuestro juego.
  2. Crea un Canvas (GameObject > UI > Canvas).
  3. Añade un Raw Image al Canvas y asigna la imagen importada, ajustando su tamaño.
- La creación de los botones se llevarán a cabo de la siguiente manera:
  1. Dentro del canvas creamos los botones y fuera del canvas creamos un GameObject
  2. Crearemos un script que tenga el siguiente contenido

The screenshot shows the Unity Editor interface. On the left, the Project Explorer displays a folder structure for a project named 'JUEGOUNITY'. Inside 'Assets' are folders like '.plastic', '.vscode', 'Assets', 'Azerilo', 'carretera', 'Casas\_Pared', 'circuito', 'Concrete Barrier', 'Materials', 'obstaculosConstrucion', 'obstaculosUrbanos', 'pelotas', 'Scenes', and 'Scripts'. Scripts listed include FollowCamera.cs, MainMenu.cs (selected), Obstacle.cs, PlayerMove.cs, Salir.cs, and SceneController.cs. On the right, the code editor shows the 'MainMenu.cs' script:

```

using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections;
using System.Collections.Generic;

public class MainMenu : MonoBehaviour
{
    void Start()
    {
    }

    void Update()
    {
    }

    public void EscenaJuego(){
        SceneManager.LoadScene("Juego");
    }

    public void Salir(){
        Application.Quit();
    }
}

```

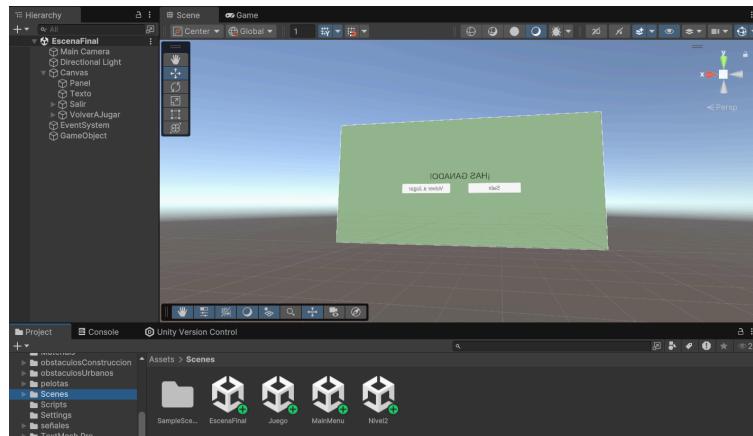
3. Arrastra el script al GameObject desde el inspector
4. Configurar OnClick de cada botón
  - a. Haz clic en él '+' para añadir un evento
  - b. Arrastra el GameObject que contiene el script al campo vacío.
  - c. En el desplegable en el botón de jugar seleccionamos el método 'EscenaJuego' y en el botón Salir el método 'Salir'
- Explicación de métodos: 'EscenaJuego()' carga la escena especificada, en este caso, la escena 'Juego', usando SceneManager.LoadScene("nombre\_escena") y 'Salir' contiene Application.Quit(), cierra el juego cuando se ejecuta.

Para el texto en el Canvas, crea un Text - TextMeshPro, ajusta su posición, tamaño y color según se deseé.

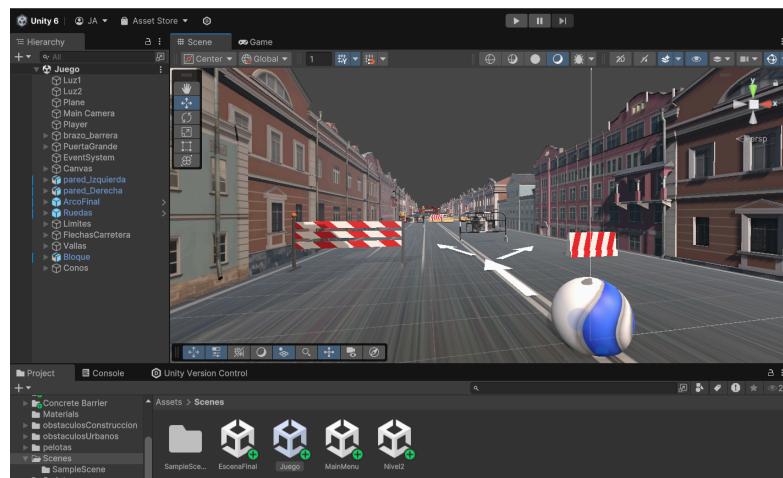


## Escena 4: EscenaFinal

En la escena final, asigna al botón 'Volver a jugar' el mismo script que lleva al primer nivel. Al botón 'Salir', asigna el script con el método 'Salir' (igual que en la escena 1).



## Escena 2: 'Juego'



Esta escena será el nivel 1 de nuestro juego, en la que se compone de:

### 1. Pelota

Es el objeto principal del juego, y se le asignan los siguientes componentes:

- Mesh Renderer: Activado para que la pelota sea visible en la escena.
- Sphere Collider: Detecta colisiones o interacciones físicas con otros objetos.
- RigidBody: Permite que la pelota se mueva, caiga por gravedad y choque de manera realista con otros objetos.
- Textura: Se le puede aplicar una textura personalizada.

Además, tiene un script asociado que contiene el código comentado para explicar su funcionamiento.

```

4  public class PlayerMove : MonoBehaviour
5  [
6      5 references
7      public Rigidbody rb; // Referencia al Rigidbody del jugador
8          1 reference
9      private float forwardForce = 350; // Fuerza hacia adelante
10         2 references
11     private float sideForce = 1000; // Fuerza lateral
12         1 reference
13     private float jumpForce = 6.0f; // Fuerza del salto
14         4 references
15     private bool isGrounded; // Comprobar si el jugador está en el suelo
16         1 reference
17     public float groundCheckDistance = 1f; // Distancia del raycast al suelo
18         // Referencia a la UI de puntuación
19         2 references
20     public Text puntuacionText;
21
22     // Start is called once before the first execution of Update after the MonoBehaviour es creado
23     0 references
24     void Start(){
25         rb.useGravity = true; // Gravedad activada al empezar
26     }
27     0 references
28     void Update(){
29         isGrounded = CheckIfGrounded();
30         rb.AddForce(0, 0, forwardForce * Time.deltaTime); // Movimiento hacia adelante
31
32         // Movimiento hacia los lados
33         if (Input.GetKey("d"))
34         {
35             rb.AddForce(sideForce * Time.deltaTime, 0, 0);
36         }
37         if (Input.GetKey("a"))
38         {
39             rb.AddForce(-sideForce * Time.deltaTime, 0, 0);
40
41             // Salto (aplica la fuerza solo si el jugador está en el suelo)
42             if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
43             {
44                 rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse); // Aplica la fuerza de salto
45             }
46             // Actualizar la puntuación basada en la posición en el eje Z
47             if (puntuacionText != null) // Verificar que la referencia no es null
48             {
49                 puntuacionText.text = "PUNTOS: " + Mathf.RoundToInt(transform.position.z).ToString(); // Mostrar la puntuación
50             }
51         }
52         // Método para detectar si el jugador está tocando el suelo usando Raycast
53         1 reference
54         private bool CheckIfGrounded(){
55             // Lanza un Raycast hacia abajo desde el centro del jugador y verifica si toca algo
56             return Physics.Raycast(transform.position, Vector3.down, groundCheckDistance);
57         }
58         // Usando OnCollisionEnter para verificar si tocamos el "Plane"
59         0 references
60         private void OnCollisionEnter(Collision collision){
61             // Verificar si el objeto con el que hemos colisionado es el "Plane"
62             if (collision.gameObject.name == "Plane")
63             {
64                 isGrounded = true;
65             }
66         }
67         0 references
68         private void OnCollisionExit(Collision collision)[]
69             if (collision.gameObject.name == "Plane")
70             {
71                 isGrounded = false;
72             }
73     }

```

Al comenzar el juego, activamos la gravedad y, en el método Update, añadimos el código para mover la pelota hacia adelante, hacia la izquierda con la tecla 'A', hacia la derecha con 'D', y saltar con la barra espaciadora (solo si la pelota está en el suelo)

- Tenemos otros métodos como CheckIfGrounded(), para verificar si la pelota está tocando el suelo lanzando un rayo desde su posición hacia abajo, si el rayo detecta una superficie dentro de una distancia específica devuelve true.
- El método OnCollisionEnter verifica si el objeto con el que colisionó se llama "Plane" (suelo). Si es así, establece la variable isGrounded en true, indicando que el jugador está tocando el suelo y el método OnCollisionExit cambia isGrounded a false si la pelota deja de tocar el suelo.
- Las fuerzas de salto, movimiento hacia adelante y movimiento lateral se inicializan con valores específicos para controlar la dinámica del jugador.

El código también incluye ‘public Text puntuacionText’, al que se le asigna el valor del eje Z de la pelota. Esta comienza en 0,0,0 (ejes X Y Z). Cuando avanza, su valor en el eje Z aumenta, incrementando la puntuación. El código se introduce en el método Update para actualizar la puntuación conforme la pelota avanza

```
if (puntuacionText != null) // Verificar que la referencia no es null
{
    puntuacionText.text = "PUNTOS: " + Mathf.RoundToInt(transform.position.z).ToString(); // Mostrar la puntuación
}
```

Para llevar a cabo el seguimiento de la pelota por parte de la cámara crearemos un script para asociarlo a la cámara

```
0 references
3 public class FollowCamera : MonoBehaviour
4 {
5     1 reference
6     public Transform player;           // Referencia al transform de la pelota (jugador)
7     3 references
8     public Vector3 offset;          // Distancia entre la cámara y la pelota
9
10    0 references
11    void Start()
12    {
13        // Si no has asignado un offset en el inspector, ajustamos un valor por defecto
14        if (offset == Vector3.zero)
15        {
16            offset = new Vector3(0, 5, -7); // Ajusta este valor según lo que necesites
17        }
18
19    0 references
20    void Update()
21    {
22        // Actualizar la posición de la cámara para que siga a la pelota
23        // La cámara se mueve con la pelota pero mantiene una distancia constante
24        Vector3 targetPosition = player.position + offset;
25
26        // Asignamos la posición calculada a la cámara
27        transform.position = targetPosition;
28    }
29 }
```

En el Start() colocamos la cámara en la posición deseada, en el Update(), actualizamos la posición de la cámara para que siga a la pelota, sumando el offset (desplazamiento) que le hayamos asignado previamente

### Recolección de Monedas

En el mapa estarán habrá monedas que el usuario debe de coger, estas dispondrán de una animación de giro

- Cada moneda tendrá asociado un script con el siguiente contenido con el fin de que cuando la pelota choque con cada moneda esta se elimine

```

0 references
public class Coin : MonoBehaviour
{
    // Referencia al ScoreManager para modificar la cantidad de monedas
    1 reference
    public ScoreManager scoreManager;

    0 references
    private void OnTriggerEnter(Collider other)
    {
        // Verificar si lo que colisiona es el jugador (usando el tag "Player")
        if (other.CompareTag("Player"))
        {
            // Llamar al método AddCoin para incrementar el contador de monedas
            scoreManager.Addcoin();

            // Destruir la moneda después de ser recogida
            Destroy(gameObject);
        }
    }
}

```

- En cada moneda le añadiremos tambien box collider y marcaremos la casilla de Is Trigger para detectarla pero no choque con ella alterando su dirección

Para contar el número de monedas que cogemos lo haremos de la misma forma que la puntuación, solo que al GameObject le asociaremos este Script que nos permite llevar un conteo de las monedas que cogemos

```

0 references
public class ScoreManager : MonoBehaviour
{
    // Variable para llevar el conteo de las monedas
    2 references
    public int coinCount = 0;
    // Referencia al Text de UI donde se mostrará el número de monedas
    1 reference
    public Text coinText;
    // Método para aumentar el contador de monedas
    0 references
    public void Addcoin()
    {
        coinCount++;
        UpdatecoinText(); // Actualiza el texto en pantalla
    }
    // Método para actualizar el texto en pantalla
    1 reference
    private void UpdatecoinText()
    {
        coinText.text = "MONEDAS: " + coinCount.ToString();
    }
}

```

## 2. Obstáculos

Todos los obstáculos como vallas, conos, ruedas... tendrán asociado un script que le hemos llamado Obstacle

```

Obstacle.cs X PlayerMove.cs SceneController.cs
Assets > Scripts > Obstacle.cs > ...
1  using UnityEngine;
2  using UnityEngine.SceneManagement;
3
4  0 references
5  public class Obstacle : MonoBehaviour
6  {
7      0 references
8      private void OnCollisionEnter(Collision collision)
9      {
10         if(collision.transform.CompareTag("Player")){
11             SceneManager.LoadScene(SceneManager.GetActiveScene().name);
12         }
13     }
14 }

```

Este código verifica si el objeto con el que colisiona el objeto que tiene este script tiene la etiqueta "Player". Si es así, vuelve a cargar la escena actual usando SceneManager.LoadScene(), lo que hace que la escena se reinicie. El objetivo es simular que cuando toca un obstáculo el jugador pierde y empieza de nuevo el recorrido

### 3. Paredes

Para reiniciar el juego al tocar las paredes que simulan las casas, he colocado una pared vertical en lugar de asignar un script a cada casa. Esta pared, ubicada a la misma altura en el eje X que las casas, tiene un script que reinicia el nivel al colisionar con el jugador. Además, se desactiva su Mesh Renderer para que esté oculta. Esto simplifica el código y mejora la eficiencia al usar un solo script en la pared, que actúa como detector de colisiones, simulando el contacto con las casas.



### 4. LímiteFinal

El límite final de la meta se configura de manera similar a las paredes

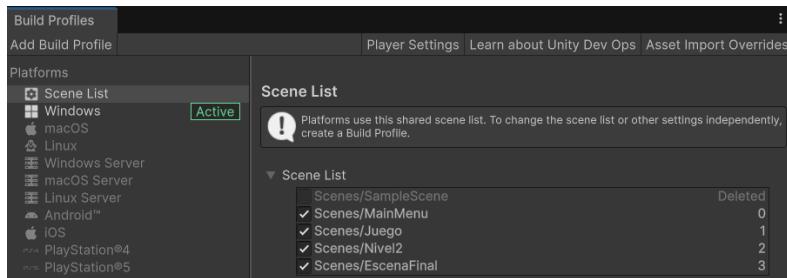
Sin embargo, este límite necesita un script diferente para cargar la siguiente escena al colisionar con el jugador.

El script que se utiliza es el siguiente:

```
Assets > Scripts > Victoria2.cs > ...
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine.SceneManagement;
5
6  public class Victoria2 : MonoBehaviour
7  {
8      private void OnCollisionEnter(Collision collision){
9          if(collision.transform.CompareTag("Player")){
10              SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex+1);
11          }
12      }
13  }
14
```

Este método detecta la colisión entre el límite final y el jugador. Al colisionar con el objeto que tiene el script, verifica si el objeto tiene la etiqueta "Player". Si es así, usa SceneManager.LoadScene para cargar la siguiente escena.

SceneManager.GetActiveScene().buildIndex obtiene el índice de la escena actual y '+1' indica a Unity que cargue la escena siguiente, es decir, el nivel 2



Esta es la estructura y permite que, al llegar al límite final, el jugador pase automáticamente al siguiente nivel del juego

### Escena 3: 'Nivel2'



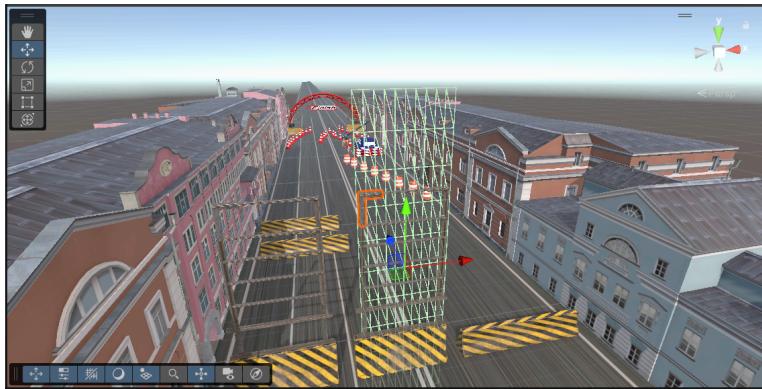
El nivel dos es similar en relación a su funcionalidad que el nivel 1, a diferencia que en el nivel 2 tenemos un poco más de complejidad con nuevos obstáculos como:

- Coche

Es un prefab que contiene todos los elementos a excepción de las ruedas, que son otro prefab que contiene otros elementos de diseño como los frenos, los discos... Uniendo ambos se formará el coche. Tienen asociado al igual que todos los obstáculos el Script Obstacle.

- Valla altas

Al principio, había una valla pequeña, pero al usar un Asset con materiales de hierro, la he reemplazado por una valla más alta. Esta valla no tiene asociado el script de Obstacle, ya que ese script lo lleva una pared invisible, que simula el choque con la valla, igual que ocurre con las paredes laterales



- Elementos de circuito

- Cono
- Señal

- Rampa

- No tiene el script Obstacle porque está en el camino correcto del circuito
- Contiene un Box Collider, que detecta colisiones y simula la subida de la rampa y la caída de la pelota debido a la gravedad al avanzar.

- Otros elementos de construcción y circuito

- Conos de punta
- Muro
- Barrera de parking
- Ruedas apiladas

El límite final de este nivel nos dará paso a la última escena ‘Victoria 2’. Esto se realizará mediante el Script ‘Victoria 2’

```

Victoria2.cs X PlayerMove.cs SceneController.cs ...
Assets > Scripts > Victoria2.cs > ...
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine.SceneManagement;
5
6  0 references
7  public class Victoria2 : MonoBehaviour
8  {
9      0 references
10     private void OnCollisionEnter(Collision collision){
11         if(collision.transform.CompareTag("Player")){
12             SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex+1);
13         }
14     }
15 }
```

El método OnCollisionEnter se activa automáticamente cuando el objeto al que está vinculado detecta una colisión con otro objeto. En este caso, realiza lo siguiente:

1. Identifica el objeto que colisiona:
  - Verifica si el objeto que ha colisionado tiene el tag "Player" mediante CompareTag
2. Avanza a la siguiente escena:
  - Si el objeto es el jugador, usa el SceneManager para cargar la escena siguiente, calculando el índice actual '+1' en los Build Settings (como vimos anteriormente)

Estamos en 'Nivel2' (escena 3) por lo que cambiaría a EscenaFinal (escena 4)

## Mecánica del Juego

Entorno:

- El juego se desarrolla en un escenario 3D, específicamente en una carretera con un ancho limitado. Al final del ancho de la carretera, aparecen casas a los lados a modo de pared.
- A lo largo del recorrido, hay obstáculos que el jugador debe evitar.

Reglas:

- El jugador controla una pelota que se mueve automáticamente hacia adelante en el eje Z.
- El desafío es esquivar los obstáculos situados sobre la carretera y evitar colisionar con las paredes laterales. Si la pelota choca con un obstáculo o las paredes, el juego se reinicia.
- El jugador debe avanzar lo más lejos posible en el eje Z para lograr una mejor puntuación.

Objetivo:

- El objetivo es alcanzar la línea de meta al final de cada nivel.
- Al llegar a la meta del Nivel 1, el jugador avanza al Nivel 2.
- Al completar el Nivel 2 y llegar a la meta, el jugador es dirigido a una pantalla final con dos opciones:
  - Jugar de nuevo, lo que lo regresará al Nivel 1.
  - Salir del juego, usando el botón Salir.

Sistema de Puntuación:

- La puntuación del jugador se basa en la distancia recorrida en el eje Z
- Se mostrará en pantalla el número de monedas recogidas

## Instrucciones de manejo del Juego

- No se necesita el uso de ratón para manejar la pelota
- La pelota se maneja mediante las teclas
  - ‘A’ para ir hacia la izquierda
  - ‘D’ para ir hacia la derecha
  - Barra espaciadora para saltar

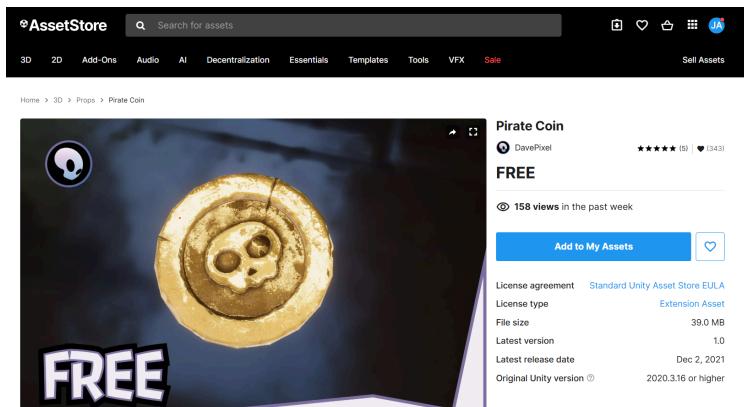
## Elementos utilizados

Para llevar a cabo el desarrollo de la interfaz del juego hemos descargados numerosos Assets de la propia Asset Store de Unity, algunos de ellos son:

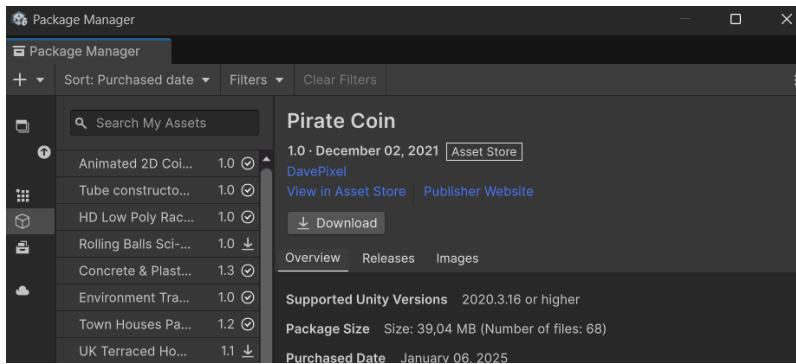
- HD Low Poly Racing Car
- Rolling Balls Sci-fi Pack Free
- Concrete & Plastic Barrier
- Environment Track Lowpoly
- Town Houses Pack
- 3D Cartoon Road Warning Signs

Para importarlos debemos de hacer los siguientes pasos:

1. Dirigirnos al Asset que queremos llevar hasta nuestro proyecto y al botón de Add to my Assets



2. Abrir unity e irnos a nuestros Assets
3. Seleccionar el añadido previamente y descargar



4. Importarlo en nuestro proyecto y aparecerá junto a los demás Asset importados y estaría listo para ser utilizado

## Problemas Surgidos en el Proceso

1. Inicialmente, configuré la cámara como hija de la pelota, lo que funcionaba bien sin gravedad, pero al activarla, la cámara giraba con la pelota, desorientando al jugador.

Para solucionarlo, creé un script que asocia la cámara a la pelota, permitiendo que la cámara avance según la posición de la pelota más el desplazamiento que le asignamos. De esta forma, la pelota gira mientras la cámara se mantiene estática y avanza simultáneamente

2. Tuve problemas con los colores de los objetos importados, ya que al arrastrarlos a la escena aparecían en color rosa y no me dejaba aplicar las texturas. La solución fue cambiar el shader, ya que venía configurado por defecto con uno inapropiado.

Lo cambié a Universal Render Pipeline/Lit, y desde entonces los colores de los materiales se aplicaron correctamente.

3. Al llegar a la meta, el juego cambiaba de escena sumando 1 al índice de la escena actual, pero al principio me llevaba a la escena incorrecta debido a un desorden en el orden de las escenas.

La solución fue ir a File > Build Settings > Scene List y organizar las escenas en el orden correcto para que se carguen sucesivamente de una a otra