Unity 3D

(fuente: wikipedia)

Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataformas (Véase la sección Plataformas objetivo). A partir de su version 5.4.0 ya no soporta el desarrollo de contenido para navegador a través de su plugin web, en su lugar se utiliza WebGL. Unity tiene dos versiones: Unity Professional (pro) y Unity Persona

Características Principales

Unity puede usarse junto con 3ds Max, Maya, Softimage, Blender, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks y Allegorithmic Substance. Los cambios realizados a los objetos creados con estos productos se actualizan automáticamente en todas las instancias de ese objeto durante todo el proyecto sin necesidad de volver a importar manualmente.

El motor gráfico utiliza Direct3D (en Windows), OpenGL (en Mac y Linux), OpenGL ES (en Android y iOS), e interfaces propietarias (Wii). Tiene soporte para mapeado de relieve, reflexión de mapeado, mapeado por paralaje, pantalla de espacio oclusión ambiental (SSAO), sombras dinámicas utilizando mapas de sombras, render a textura y efectos de post-procesamiento de pantalla completa.

Se usa ShaderLab language para el uso de shaders, soporta tanto programación declarativa de los programas de función fija de tuberías y shader GLSL o escritas en Cg. Un shader puede incluir múltiples variantes y una especificación declarativa de reserva, lo que permite a Unity detectar la mejor variante para la tarjeta de vídeo actual y si no son compatibles, recurrir a un shader alternativo que puede sacrificar características para una mayor compatibilidad.

El soporte integrado para Nvidia (antes Ageia), el motor de física PhysX, (a partir de Unity 3.0) con soporte en tiempo real para mallas arbitrarias y sin piel, ray casts gruesos, y las capas de colisión.

El scripting viene a través de Mono. El script se basa en Mono, la implementación de código abierto de .NET Framework. Los programadores pueden utilizar UnityScript (un lenguaje personalizado inspirado en la sintaxis ECMAScript), C# o Boo (que tiene una sintaxis inspirada en

Python). A partir de la versión 3.0 añade una versión personalizada de MonoDevelop para la depuración de scripts.

Unity también incluye Unity Asset Server - una solución de control de versiones para todos los assets de juego y scripts, utilizando PostgreSQL como backend, un sistema de audio construido con la biblioteca FMOD, con capacidad para reproducir audio comprimido Ogg Vorbis, reproducción de vídeo con códec Theora, un motor de terreno y vegetación , con árboles con soporte de billboarding, determinación de cara oculta con Umbra, una función de iluminación lightmapping y global con Beast, redes multijugador RakNet y una función de búsqueda de caminos en mallas de navegación.

Licencias

Hay dos licencias principales para desarrolladores: Unity personal y Unity Professional14 . Originalmente la version pro costaba alrededor de 200 dólares estadounidenses. La versión Pro tiene características adicionales, tales como render a textura, determinación de cara oculta, iluminación global y efectos de postprocesamiento. La versión gratuita, por otro lado, muestra una pantalla de bienvenida (en juegos independientes) y una marca de agua (en los juegos web) que no se puede personalizar o desactivar.

Tanto Unity como Unity Pro dan acceso a la documentación del motor y a tutoriales o vídeos de entrenamiento. La versión pro ofrece soporte a una version, ejemplo si ha comprado Unity 5 esta licencia le da acceso a todas las actualizaciones y soporte de las siguientes mejoras de la versión (Unity 5.x), al igual que le da acceso a las versiones beta.

Unity technologies ofrece la licencia pro como una suscripción o como un objeto de pago en una sola exhibición, esta puede ser configurada a la necesidad del desarrollador que puede incluir las plataformas a las que desee publicar, tales como Android Pro, IOS Pro, etc. La suscripción tiene un valor de 75 USD al mes durante el plazo que se quiera utilizar el motor.

Las licencias para el desarrollo en las plataformas, PlayStation 3, PlayStation 4, PlayStation Vita, Xbox 360, Xbox One, Wii, se negocian contactando con un gerente de cuentas regional.15

Las licencias educativas son proporcionados por Studica con la estipulación de que es para la compra y uso de las escuelas, exclusivamente para la educación.16

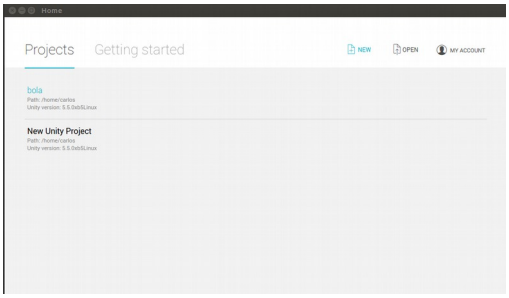
Desde la versión 4.0, un nuevo modelo de licencia se puso en marcha para organizaciones de juegos de azar. Deben ponerse en contacto con Unity directamente para obtener una licencia de distribución. Esta licencia se encuentra en el nivel de la distribución, no el nivel de desarrollador.

Creando un juego en Unity

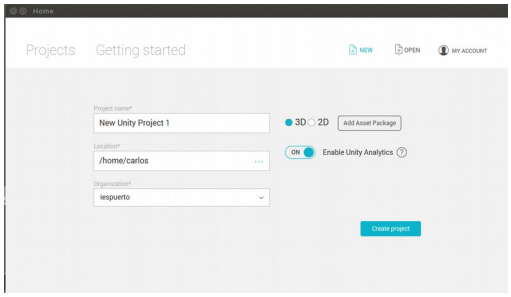
(basado en: https://unity3d.com/es/learn/tutorials/projects/roll-ball-tutorial Hay videotutorial en youtube inglés-subtítulos https://www.youtube.com/watch?time_continue=3&v=RFlh8pTf4DU)

Comenzando:

1. Pulsando en el botón New al comenzar Unity:

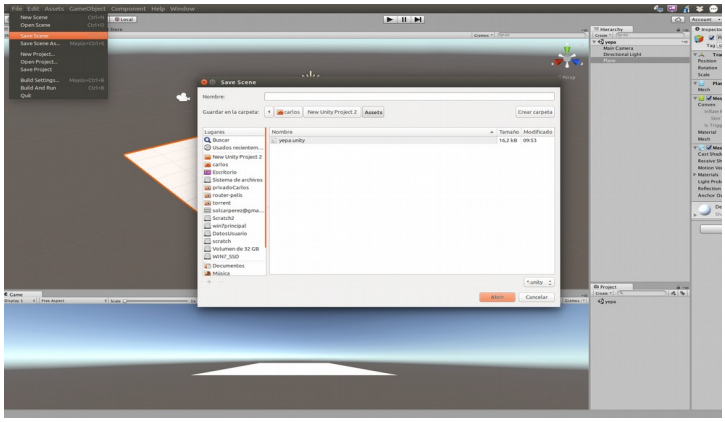


Otra opción, ahora dentro de Unity: File → New project
Nos mostrará la ventana:

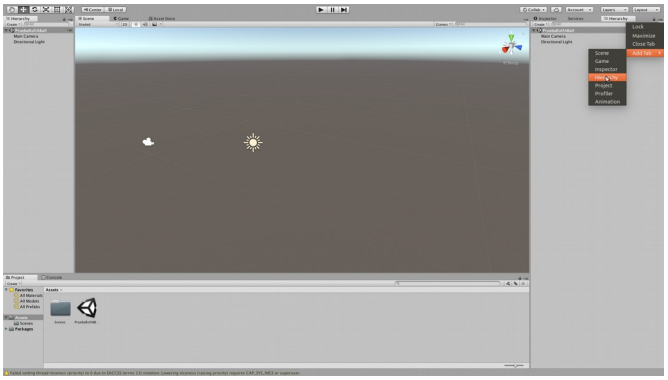


Ahí elegimos si es 3d o 2d, la ubicación y el nombre

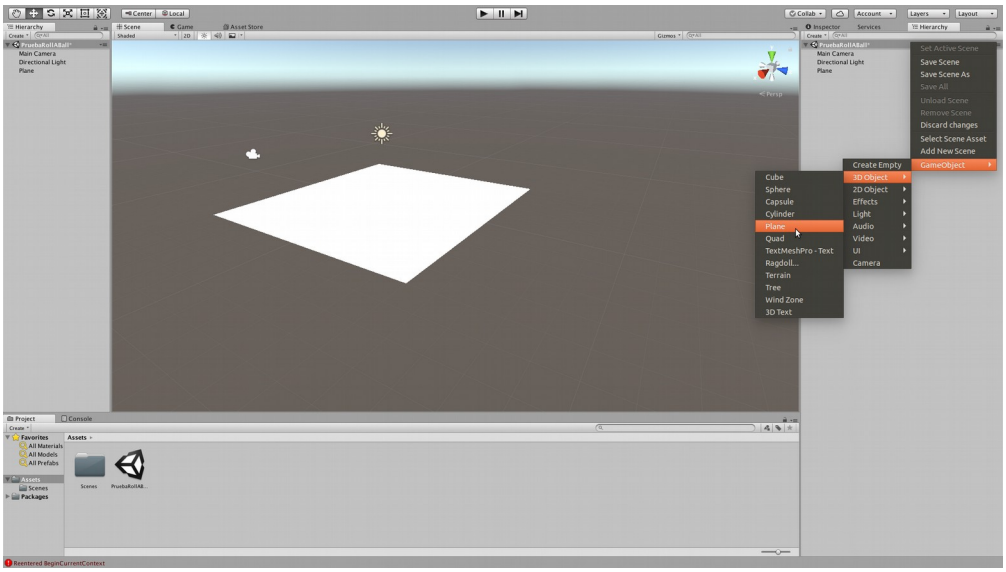
Lo primero que vamos a hacer es darle un nombre a la escena y guardarla para tener una base sobre la que trabajar y que esté convenientemente guardada



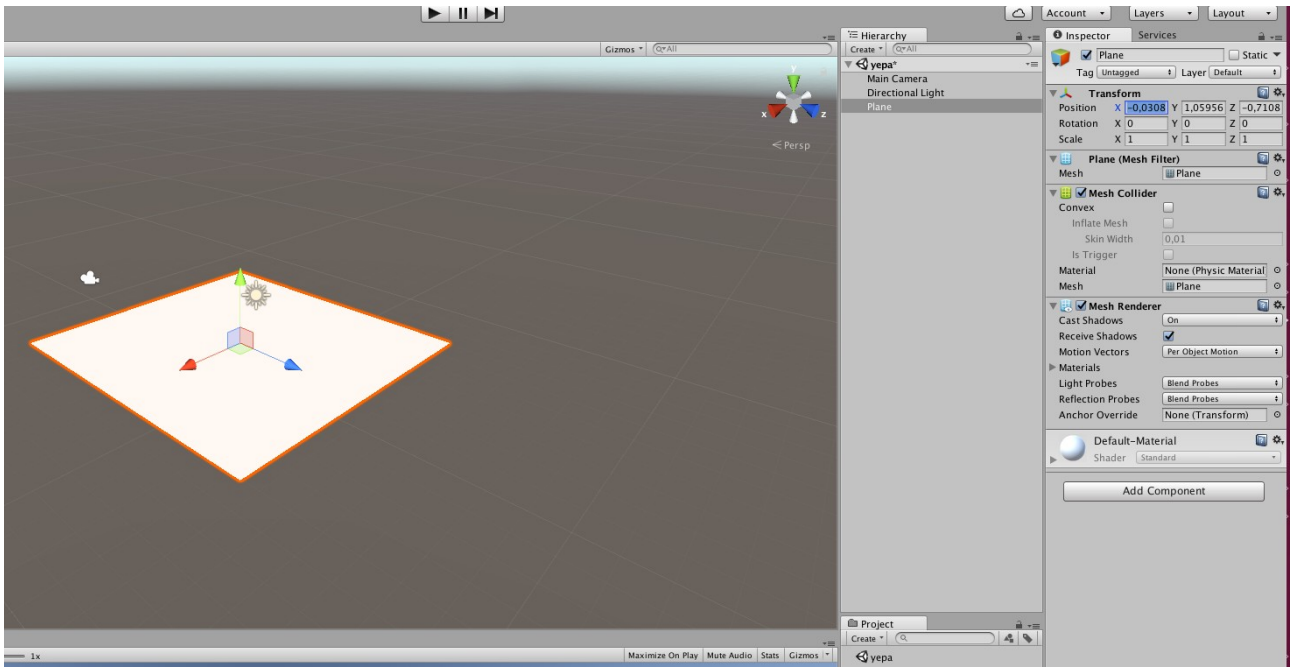
(si no aparece hierarchy se puede agregar con el menú de la derecha arriba)



Nos encontramos con un espacio vacío. Para agregar objetos a la escena
Pulsamos sobre Create:



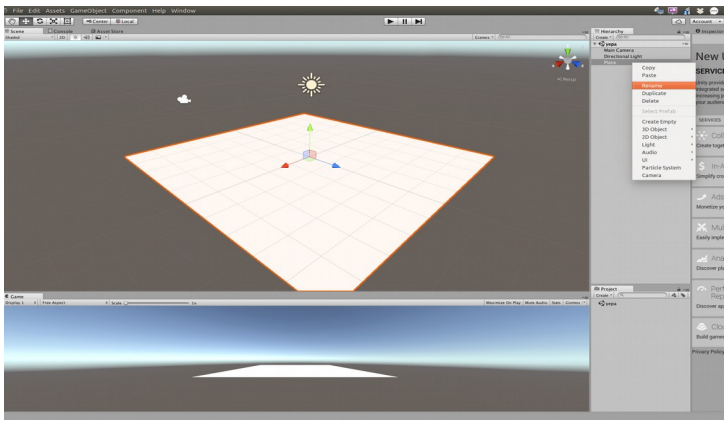
Al crear un objeto vemos donde nos ha quedado ubicado y podemos modificarlo mediante el inspector:



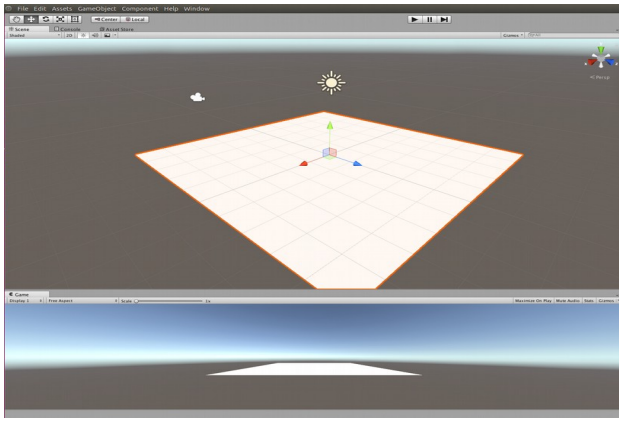
Se puede ver que aparecen los valores para x,y,z
Ubiquemos por ejemplo en 0,0,0

También vemos que aparece información sobre rotar el objeto y sobre escalarlo
Si quisiéramos que tuviera más tamaño bastaría con cambiar los valores allí expuestos
Debemos acostumbrarnos a asignar un nombre identificativo a cada elemento que hayamos creado.

Botón derecho sobre el objeto y pulsamos en “rename”

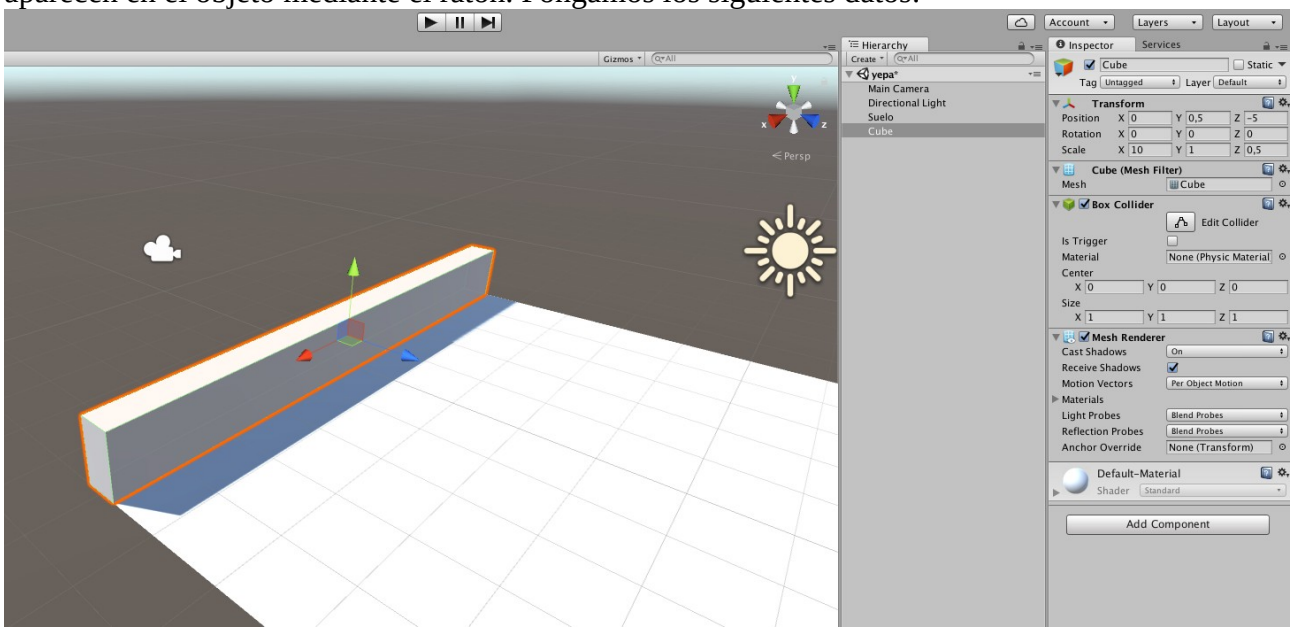


Vamos a crear las paredes del escenario. Para ello utilizaremos el cubo:



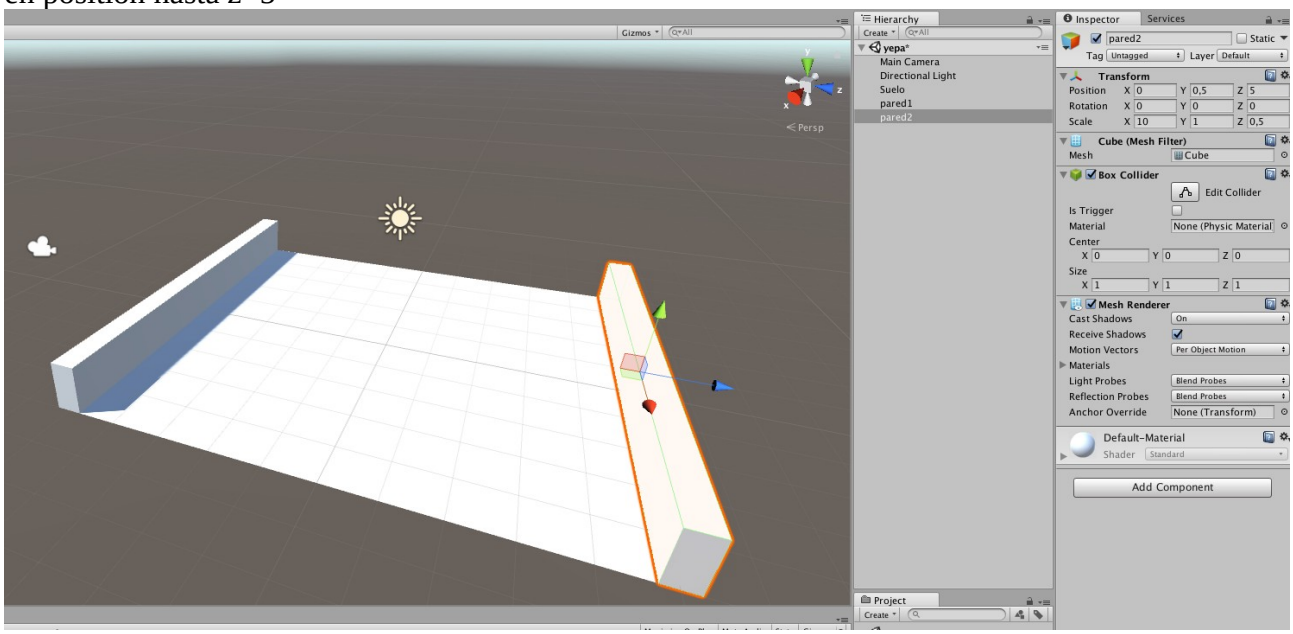
el tamaño de los objetos como el cubo nos lo hace en total de 1 así que si lo subimos 0.5 en la coordenada y conseguiremos que esté exactamente sobre el 0 en coordenada y (sobre nuestro suelo que previamente hemos creado)

Podemos mover y escalar el objeto mediante las propiedades del inspector o con las flechas que aparecen en el objeto mediante el ratón. Pongamos los siguientes datos:

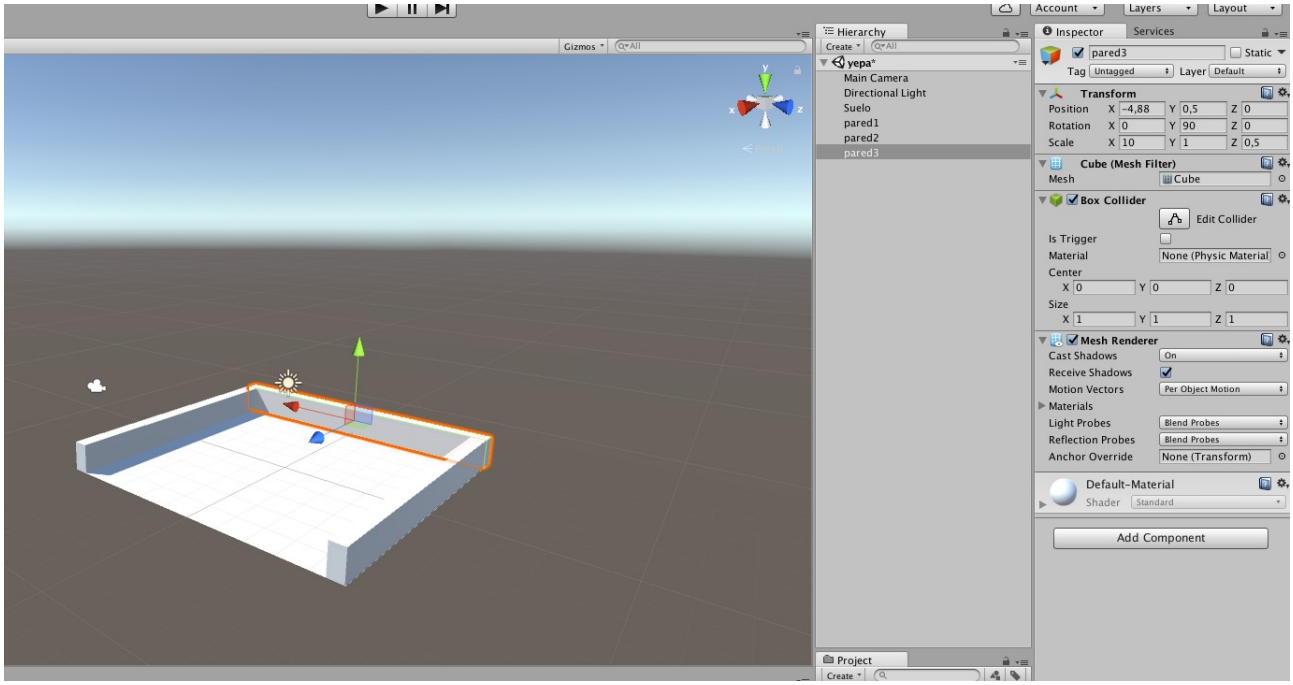


Para hacer las otras paredes podemos aprovechar este objeto que tenemos creado

Hacer la pared de enfrente bastará con usar ctrl+c ctrl+v sobre la primera pared y luego desplazarlo en position hasta z=5

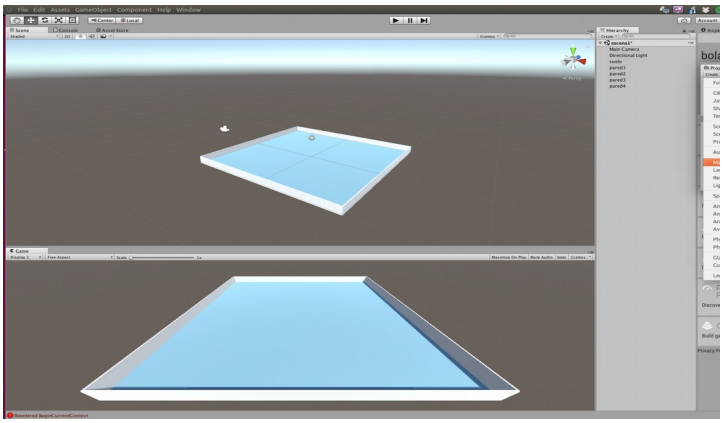


Aprovechando las propiedades de Rotation y position podemos ubicar apropiadamente la tercera pared:

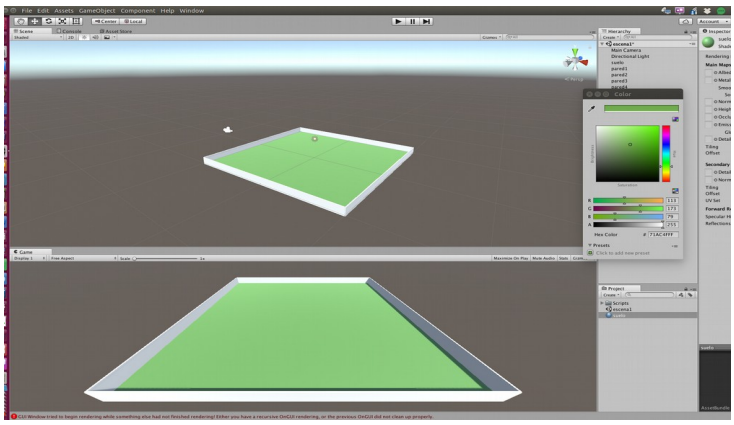


Luego hacemos lo propio para la cuarta y cerramos el espacio

Vamos a darle algo de color a estos objetos
Para ello encima del proyecto creado(la escena) Project → Create → material

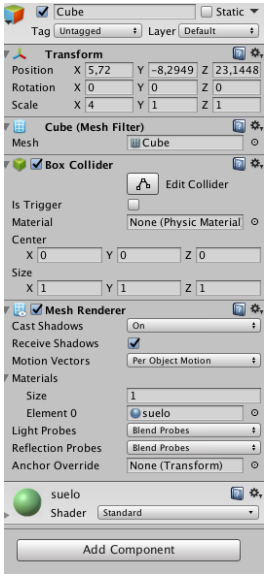


Elegimos el color que queramos dar al material (es interesante que también se le de un nombre significativo al material)



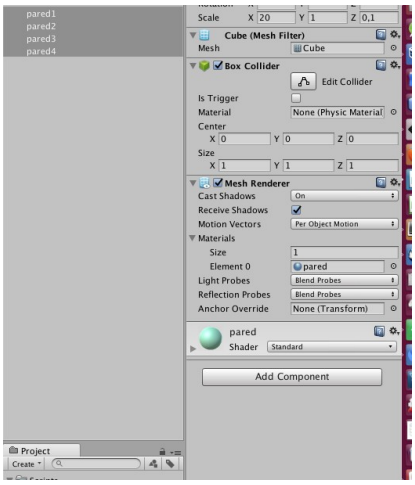
Podemos arrastrar el material al objeto que se quiere asignar y lo tomará.

Otra alternativa: Para asignar un material a un objeto lo que hacemos es seleccionar el objeto Buscamos dentro de Material qu dirá default-material y hacemos click ahí. En la ventana que abrirá seleccionamos el material que hemos creado y de esa forma quedan vinculados



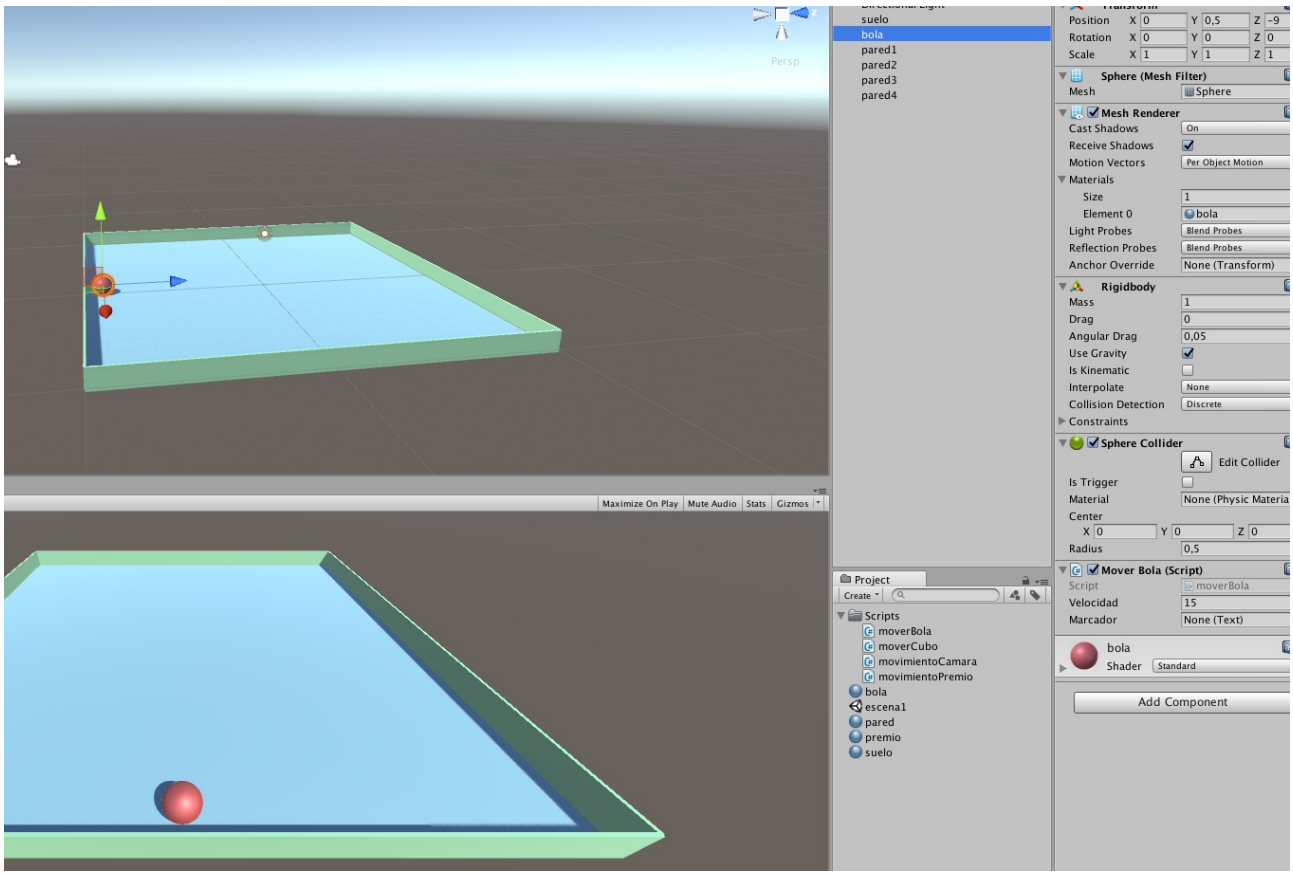
Observar que ahora en Element 0 aparece seleccionado el material: “suelo” que antes hemos creado

Para asignar un material a varios objetos a la vez (para asignar el mismo material a las cuatro paredes) seleccionamos las custro y hacemos el mismo procedimiento que antes



Ahora crearemos una bola que será nuestro jugador. Para ello elegimos Create-→3D object→ object->Sphere

y también le asignamos un material. Le podemos dejar la escala 1 en la que viene y lo posicionamos más cerca de nuestra cámara:

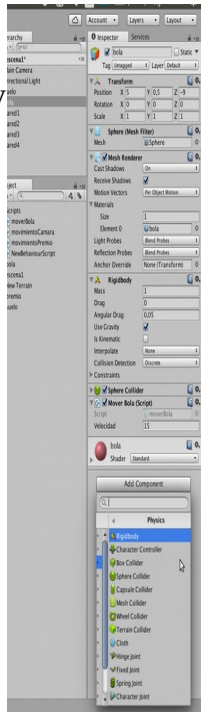


Vamos a empezar a hacer cosas de programación. Haremos un script para que la bola se mueva con las arrow-keys

Hay que añadir rigidbody a la bola si queremos que actúe como un objeto reacciona a la física. Según la documentación de unity:

Rigidbody

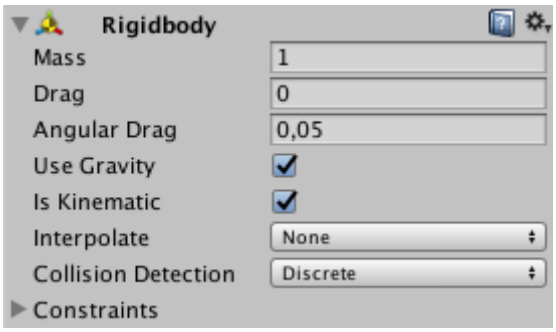
Los **Rigidbodies** le permite a sus **GameObjects** actuar bajo el control de la física. El Rigidbody puede recibir fuerza y torque para hacer que sus objetos se muevan en una manera realista. Cualquier GameObject debe contener un Rigidbody para ser influenciado por gravedad, actué debajo fuerzas agregadas vía scripting, o interactuar con otros objetos a través del motor de física NVIDIA Physx.



Para agregarlo una vez hemos seleccionado la bola pulsamos sobre Add Component → Physics → RigidBody

RigidBody nos permite múltiples posibilidades (más adelante veremos como se mueven los objetos pero hay muchas más posibilidades si has elegido RigidBody)

Veamos:

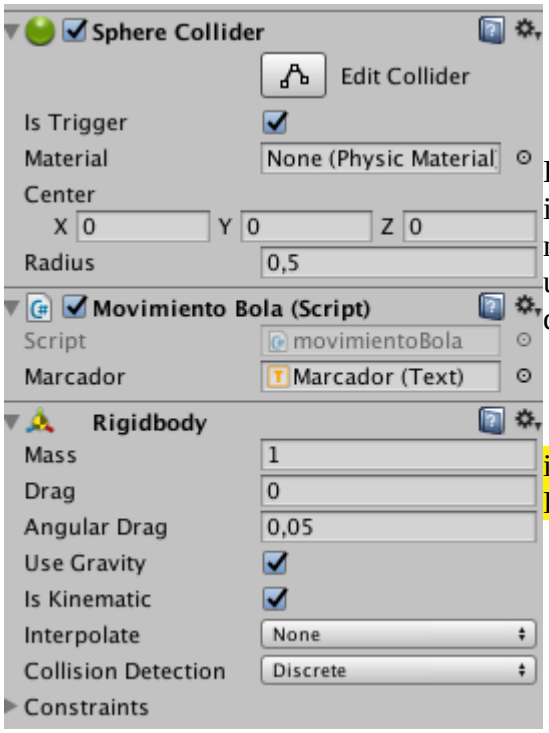


Como se ve se puede hacer que afecte la gravedad, establecer una cantidad de masa al objeto.

En la pantalla anterior se marcado “Kinematic” Si está activado, el objeto no será manejado por el motor de física, y puede solo ser manipulado por su Transform.

Observar que si un objeto es Rigidbody y tiene marcado Kinematic PUEDE REGISTRAR COLISIONES PERO NO LAS FUERZAS Así que el objeto se moverá mediante transform (por ejemplo si vas aplicando un desplazamiento el efecto será constante sin el efecto de la pérdida de fuerza por gravedad o aumento) pero aún se detectarán las colisiones(eso sí tiene que existir el compoente collider)

En concreto el caso anterior se podría obtener mediante:

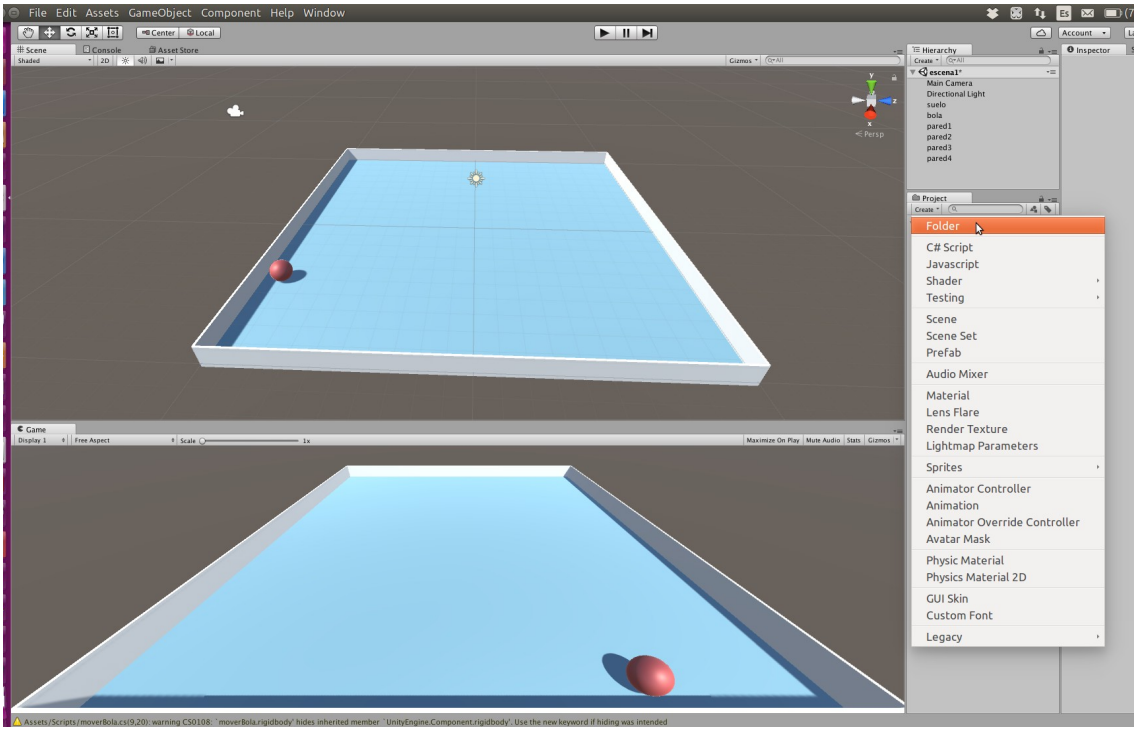


Vemos que hay un collider y es un rigidbody que es Kinematic. Puede ser interesante en algunos juegos (imaginemos que queremos que el personaje se esté moviendo siempre a velocidad constantes que únicamente cambiamos la dirección pero queremos detectar las colisiones)

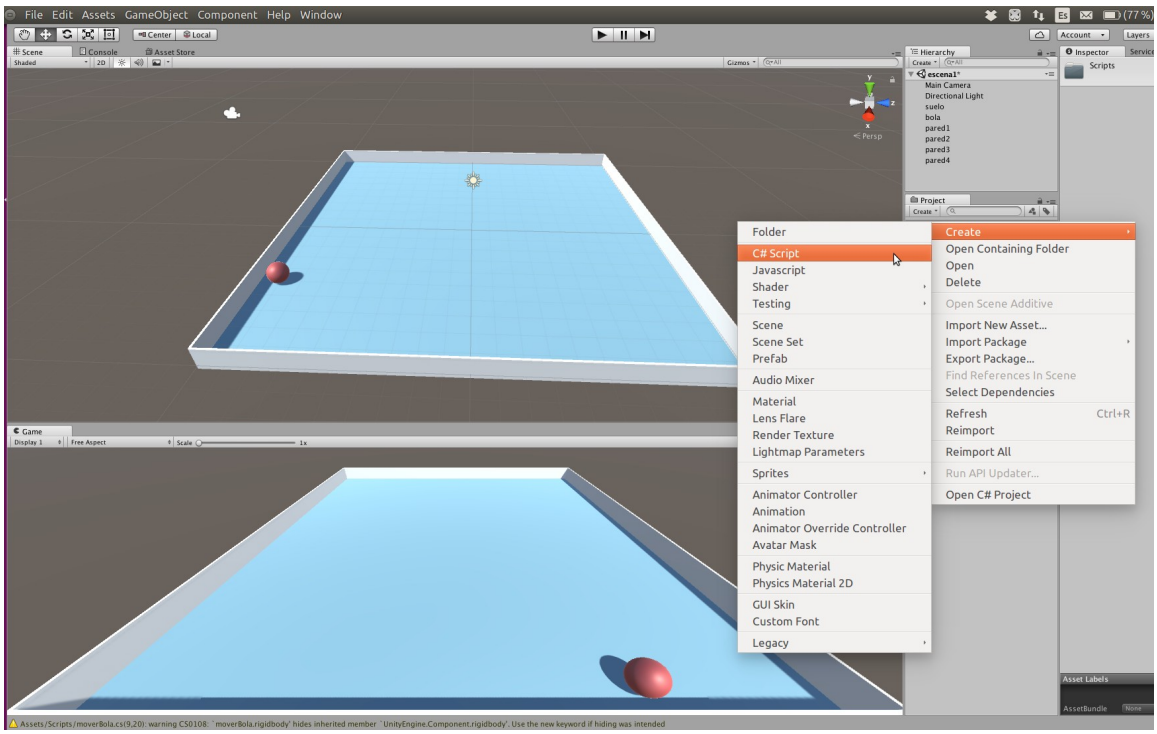
Para el juego que estamos describiendo ahora nos interesa que SE APLIQUEN LAS FUERZAS DE LA FÍSICA así que no usaremos Kinematic

Ahora para darle movimiento tenemos que crear un script:

Creemos primero una carpeta para alojar los scripts: Pestaña project y Create-→Folder



Ahora creamos el script: Botón derecho sobre la carpeta-→create-→C# Script



Para acceder a editarlo debe estar instalado mono. Con doble click nos accede y nos muestra lo que crea por defecto
Nos aparece el método Start() ahí pondremos lo que queremos que se haga desde el inicio.

Update y fixupdate

Uno de los problemas más frecuentes cuando se programa movimiento en un videojuego es hacer que la velocidad a la que se reproduce sea independiente de la máquina.

Imaginemos que queremos hacer que un objeto se mueva hacia la derecha a velocidad constante. En un primer intento haríamos algo así...

```
public float speed=1f;
void Update() {
    transform.position+=Vector3.right*speed;
}
```

...e iríamos ajustando el valor de speed en el inspector mientras ejecutamos el juego hasta que consiguiésemos la velocidad que queremos.

El problema

El método Update() se ejecuta por defecto a 60fps (60 veces cada segundo) pero cuando hay mucha carga de procesado o la máquina no es lo suficientemente potente, los fps caen, y con ellos las veces por segundo que se ejecuta el método Update(). Si por ejemplo el juego estuviese corriendo a 30fps, los objetos se moverían a la mitad de velocidad que en nuestra situación de referencia (60fps).

La solución
Hay dos formas de arreglar esto.

La primera es reajustar la velocidad para que en lugar de ser en unidades/frame sea en unidades/segundo. Para ello necesitamos saber cuantos segundos pasan en cada frame. Por suerte, Unity nos lo pone fácil, ya que el solito saca el reloj, lo mide y lo guarda en la variable Time.deltaTime . El ejemplo corregido quedaría así:

```
public float speed=1f;
void Update() {
    transform.position+=Vector3.right*speed*Time.deltaTime;
}
```

La segunda forma es utilizar el método FixedUpdate(). Este método se ejecuta cada cierto número preestablecido y fijo de frames, lo que hace que no presente los problemas de update. Se debe utilizar en aquellos scripts que impliquen un componente de físicas, y sobre todo, siempre se ha de utilizar cuando haya que añadir una fuerza a un Rigidbody. El ejemplo usando esta solución:

```
public float speed=1f;
void FixedUpdate() {
    transform.position+=Vector3.right*speed;
}
```

Cada forma tiene su caso de aplicación. La primera tiene menor coste computacional por lo general, ya que suele ejecutarse sólo 60 veces por segundo. Es la forma adecuada si estamos programando nuestro propio movimiento.

La segunda es más adecuada cuando el movimiento que programamos es mixto entre las físicas de Unity y nuestro movimiento "custom". Además el método FixedUpdate() es el único en el que debemos aplicar fuerzas cuando usamos la física de Unity si queremos un comportamiento consistente.

Veamos el siguiente código para FixedUpdate:

```
    void FixedUpdate () {

        float moverHorizontal = Input.GetAxis ("Horizontal");
        float moverVertical = Input.GetAxis ("Vertical");
        Vector3 vector = new Vector3 (moverHorizontal, 0.0f, moverVertical);
        rigidbody.AddForce (vector * 4.0f);

    }
```

mediante Input.GetAxis podemos tomar la información que nos introduce el usuario por teclado con las arrow-keys

Trabajaremos siempre con vectores 3d desde que es una aplicación 3d. Observar que la coordenada y (la altura) se le ha establecido el dato 0.0f especificando así que es un número float. Introduciremos nuestros datos habitualmente de esta forma

podemos ver que le hemos añadido a una variable llamada rigidbody una fuerza. Esto es porque es un atributo del tipo rigidbody que hemos tomado en el método Start():

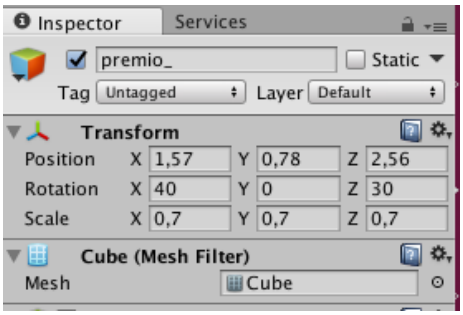
```
private Rigidbody rigidbody;

void Start () {
    rigidbody = GetComponent<Rigidbody> ();
}
```

mediante GetComponent estamos tomando el componente de nuestro objeto bola que sea del tipo Rigidbody

Si pulsamos en el botón play y luego usamos las arrow-keys podremos mover nuestro objeto bola

Ahora vamos a crear los premios:

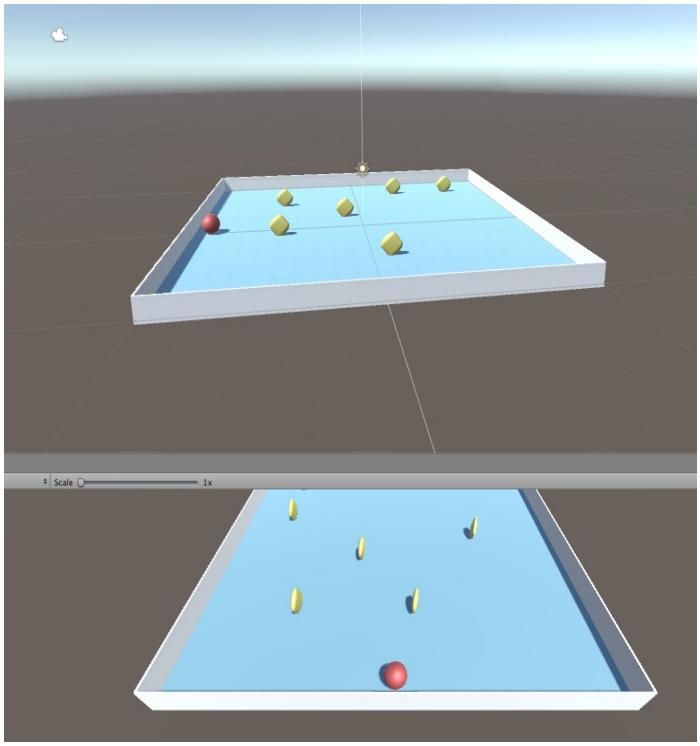
Hacer un cubo, lo llamamos premio y le damos propiedades similares a:



Se harán varias copias.

Le añadimos un script para que rote de forma automática durante la ejecución del juego.

```
void FixedUpdate () {
      transform.Rotate(2f, 2f, 0f);

}
```



Cuando la bola pase por donde está cada premio debiera desaparecer el premio al haber sido capturado por la bola

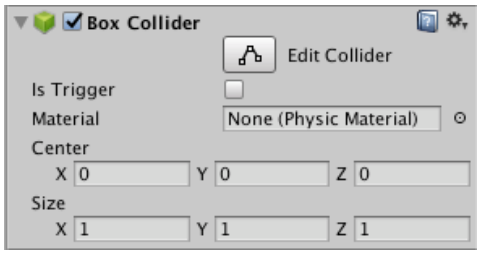
¿ cómo hacemos eso ?

Haremos uso del componente Box Collider agregándolo a los premios:

Documentación de Unity:

Box Collider

Un Box Collider es una primitiva básica de colisión en forma de cubo.



Propiedades

Propiedad:	*Función:*
Is Trigger	Si está habilitada, este Collider será usado para eventos de triggering, y será ignorado por el motor de física.
Material	Referencia al Physics Material que determina en qué forma este Collider inteactúa con otros.
Center	La posición del Collider en el espacio local del objeto.
Size	El tamaño del Collider en las direcciones X, Y, Z.

Detalles

Los Box Colliders son evidentemente útiles para cualquier objeto con forma cúbica, como una caja o un cofre. No obstante, una caja delgada puede ser usada como piso, muro o rampa.La forma en cubo también es un elemento útil en un collider compuesto.

Una vez le agregemos el componente le debemos marcar el checkbox “Is Trigger” y así nos desencadenará la acción.

Vamos a usar un método llamado OnTriggerEnter(Collider other) en la bola que nos detecta cuando chocamos con otro objeto

```
  void OnTriggerEnter(Collider other) {

        }
    }
```

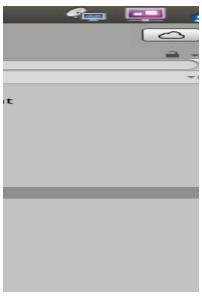
Ahora bien, esto es un comportamiento que queremos que se de únicamente al chocar con los premios. Observar el siguiente código:

```
  void OnTriggerEnter(Collider other) {
        if(other.gameObject.CompareTag("Premio")){
            other.gameObject.SetActive (false);
        }
    }
```

Si el objeto con el que chocamos tiene el tag “Premio” entonces desactivamos el objeto con el que nos hemos encontrado

Así pues debemos etiquetar los premios con la palabra “Premio”

Para hacer esto:



Así pues ya capturamos los premios

Poner puntuaciones:

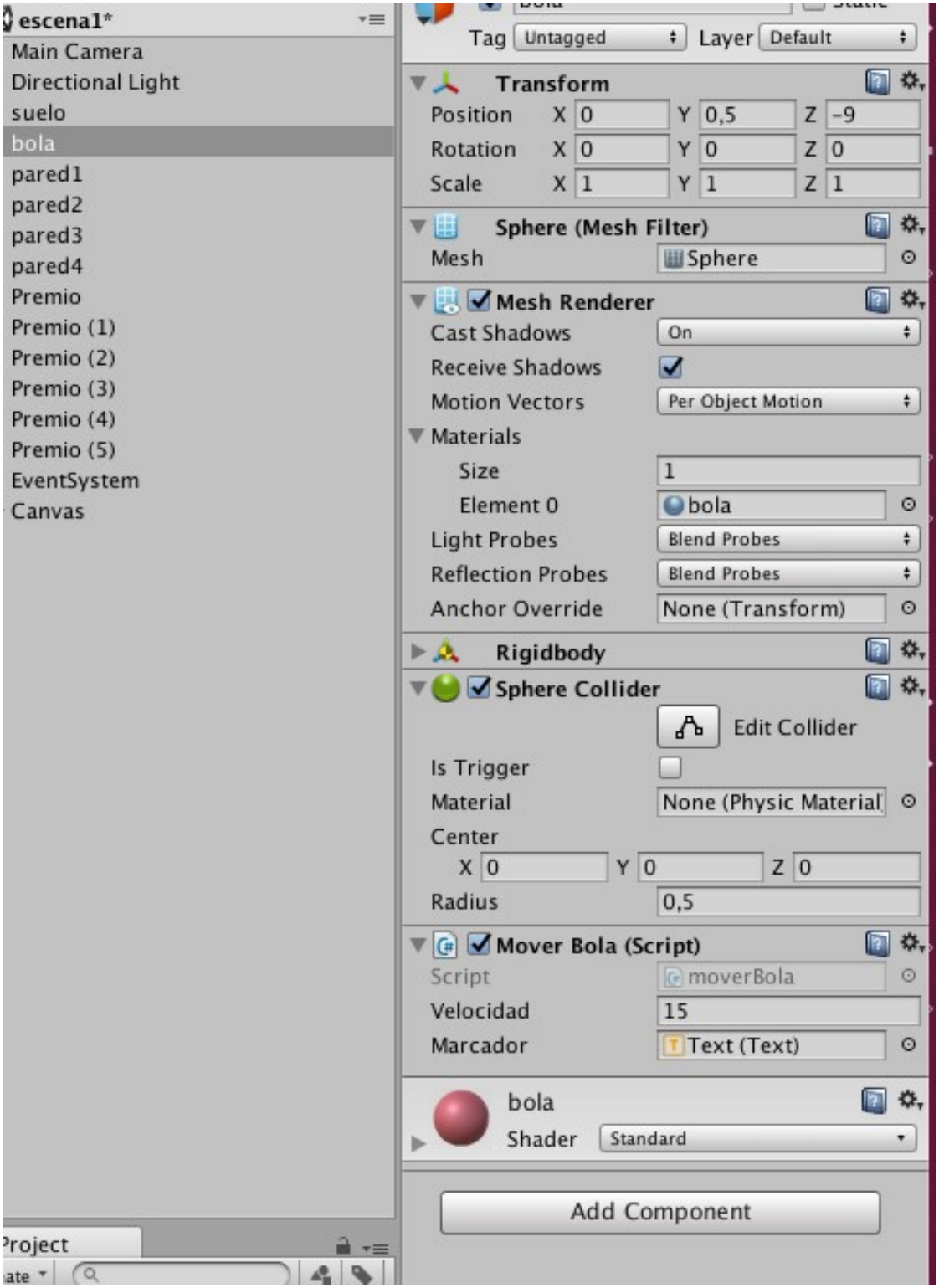
Contar los éxitos es fácil. Tiene lugar cuando colisionamos con los premios

```
private int contadorPremiosAdquiridos;
public Text marcador;

  void Start () {
        rigidbody = GetComponent<Rigidbody> ();
        contadorPremiosAdquiridos = 0;
        marcador.text = "Premios: " + contadorPremiosAdquiridos;
    }


  void OnTriggerEnter(Collider other) {
        if(other.gameObject.CompareTag("Premio")){
            other.gameObject.SetActive (false);
            contadorPremiosAdquiridos++;
```

```
            marcador.text = "Premios: " + contadorPremiosAdquiridos;

        }

    }
```

Observar que está puesto como pública una variable tipo Text llamada marcador.

Haciendo eso conseguimos que nos aparezca en las propiedades del objeto esa variable:



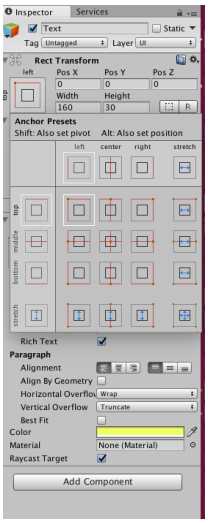
Observar que aparece tanto la variable Marcador (nos dice que es de tipo Text) y la variable Velocidad (que también se ha dejado como pública para poder modificar fácilmente la velocidad desde fuera del objeto

Ahora lo que haremos es crear un objeto del juego que reproduzca texto en pantalla y lo vincularemos a esa variable Marcador. De esa forma el dato de la variable marcador se mostrar en pantalla

para eso haremos: Create-→UI-→Text

Esto nos creará un objeto de tipo hijo de Canvas de tipo Text

Este tipo de objetos se posiciona en la pantalla mediante Rect Transform:



Elegimos la posición que queramos (por ejemplo superior izquierda) y mientras pulsamos Shift y Alt pulsamos con el ratón en la posición deseada

Finalmente lo único que hay que hacer es arrastrar el elemento text sobre la variable pública Marcador que hemos creado en en el objeto bola de esta forma ya quedarán vinculados y mostrará la puntuación

Uso de la cámara en el juego

La cámara es un elemento más podemos posicionarla y girarla mediante las propiedades Transform igual que con el resto de los objetos

Ahora bien, si queremos que se mueva de forma dinámica durante el juego tendrémos que hacerlo mediante un script

Un movimiento interesante sería el de seguir la bola. Veamos como hacerlo:

```
public GameObject bola;
```

private Vector3 offset;

```
  void Start () {

        offsetRecto = this.transform.position - bola.transform.position;

    }


  void LateUpdate () {
      transform.position = bola.transform.position + offset
    }
```

con el objeto public GameObject únicamente tendremos que arrastrar nuestro objeto bola hasta la variable pública que se habrá creado en las propiedades de la cámara y ya estarán vinculados

Lo que hace este script es simplemente calcula la diferencia inicial entre cámara y bola. Durante el juego si la bola se mueve a otra posición la posición de la camara se recalculará como la nueva posición de la bola manteniendo la distancia inicial. Así pues se seguirá la bola

Introduciendo un botón de reset

Introducimos un nuevo objeto para canvas: Create → UI → button

Al hacerlo nos creará un subelemento text que podemos usar para introducir texto al botón (es opcional este elemento y se puede eliminar)

Lo ubicamos por el mismo procedimiento que el marcador

Ahora para introducir el listener vamos a apoyarnos en un Script

Creamos un script y lo vinculamos al botón como hemos hecho con otros objetos. Pongamos por ejemplo que hemos llamado al Script: Reiniciar.cs

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

public class Reiniciar : MonoBehaviour {


    private Button btnReiniciar;
    // Use this for initialization
    void Start () {
        btnReiniciar = GetComponent<Button>();
        btnReiniciar.onClick.AddListener(reiniciar);

    }

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

    }


    void reiniciar()
    {
        //Debug.Log("has pulsado reset!!");
        SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    }
}
```

Observamos que en el método start() Localizamos el botón y le agregamos un listener

Luego en el método reiniciar() vemos que cargamos de nuevo la escena que estamos ejecutando teniendo por efecto que reiniciamos la escena

Unity adicionalmente nos permite poner luces y otras muchas posibilidades. Sólo hay que intentarlo!