

**INTRODUCCIÓN A LA PROGRAMACIÓN
DE VIDEOJUEGOS**

INTRODUCCIÓN A LA PROGRAMACIÓN DE VIDEOJUEGOS

SIMULACIÓN FÍSICA EN UNITY 3D

Ramón Mollá

rmolla at dsic.upv.es - ext. 73549

Grupo de Informática Gráfica

Departamento de Sistemas Informáticos y Computación

Objetivos de aprendizaje

- Conocer los principios básicos de la simulación física
- Presentar los elementos básicos que permiten la simulación física en el motor de videojuegos Unity
- Conocer los principios básicos de la detección y resolución de colisiones
- Detección y Resolución de colisiones en Unity3D

Índice

Simulación física

- Cuerpo Rígido

 - Juntas

 - Materiales

- Detección y Resolución de colisiones

 - Colisionadores

- Ejemplo

Introducción (I)

Justificación

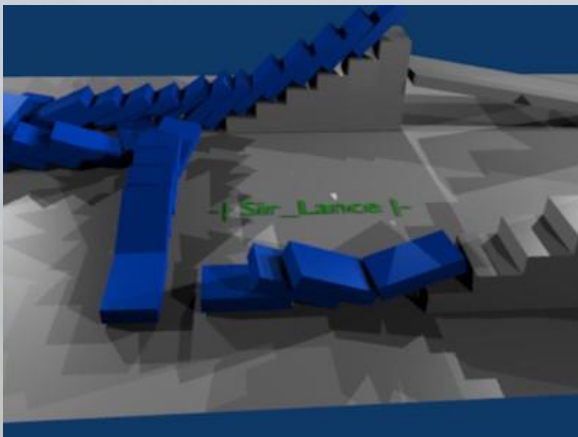
Videojuegos son para los humanos

Humanos acostumbrados a mundo real basado en la física

Introducción de reglas físicas en mundo virtual los hace parecer más naturales y creíbles

Simulación física enriquece la experiencia de juego

- Empleo de simulación física
- Comportamiento emergente



Introducción (II)

Animación tradicional / simulación física

Animación	Simulación física
Creada por artistas	Generada por algoritmos
Captura de movimiento real	Mínima intervención humana
Mucha mano de obra. Caro	Reducción de costes de desarrollo de contenidos



Introducción (III)

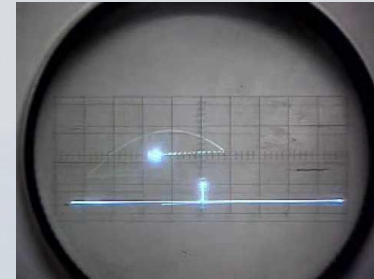
Simulación Física en videojuegos

La física en los juegos se reduce a la mecánica clásica

Simulación de las leyes del movimiento de objetos grandes

Empleo de gravedad y otras fuerzas

Objetivo: dar la sensación de que los objetos son sólidos, con masa, presentan inercia o rebotan, colisionan,...



La simulación física siempre ha existido desde el primer videojuego: *tenis for two* o *asteroids*

Está en

- Partículas: chispas, fuegos artificiales, humo y explosiones
- Balística de las balas, misiles, flechas,...
- Simuladores de vuelo, coches, carreras,...



Introducción (IV)

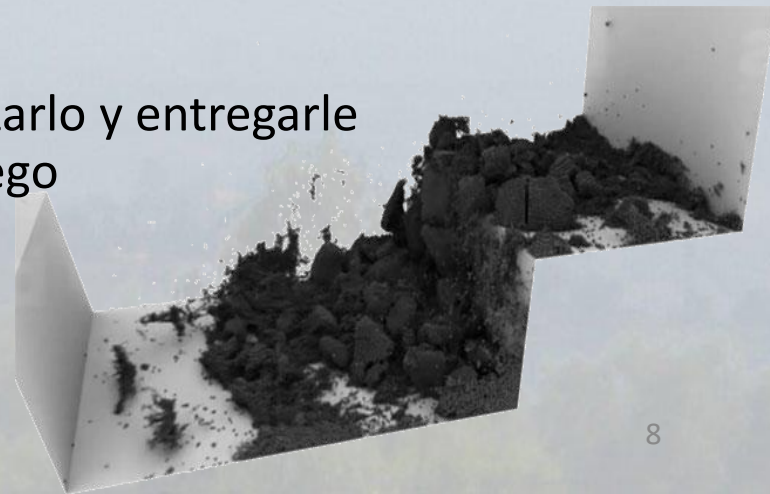
Motor de simulación física

Un API de simulación de la física en general que no está programado con los detalles concretos de cada juego

Es básicamente un API de cálculo matricial: escalares, puntos, vectores, matrices,...

Contiene las ecuaciones necesarias para simular la física:
Posición, velocidad, aceleración, mecánica y cinemática básica, leyes de Newton, masa, momentos, fuerzas

Para realizar la simulación hay que parametrizarlo y entregarle los detalles de simulación de cada nivel de juego



Introducción (V)

Motor de simulación física

Soporte de

Partículas

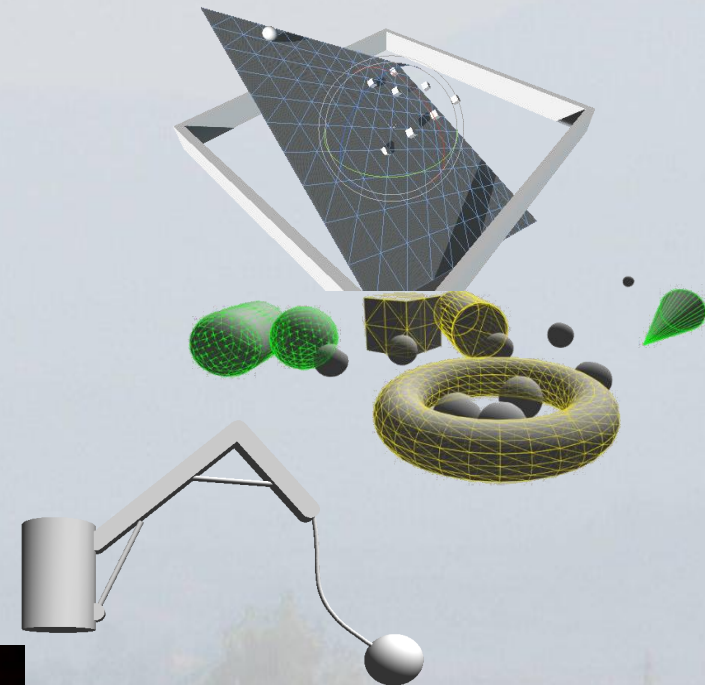
Cuerpos rígidos. Cinemática lineal y rotacional

Cuerpos elásticos. Muelles y elasticidad

Articulaciones. Cinemática directa e inversa

Dinámica. Masas y fuerzas. Inercia. Fricción

Detección y Resolución de colisiones



Introducción (VI)

Cuerpo rígido

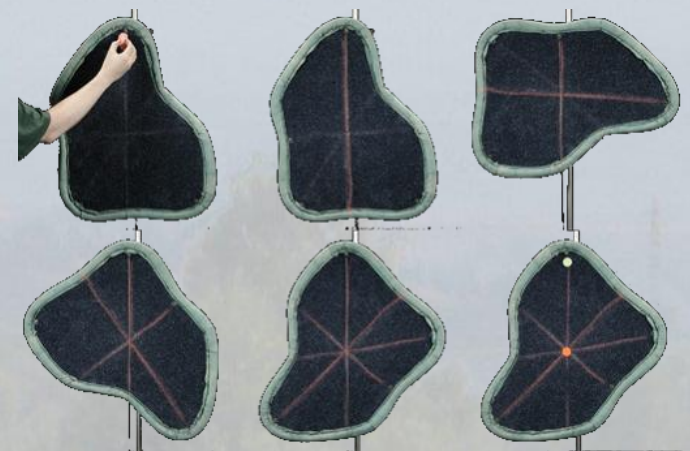
No necesariamente de forma esférica o puntual

Posición representa la localización del centro de masas

Superficie puede no ser perfectamente pulida. Pueden existir fuerzas de rozamiento

Experimentan movimiento rotacional además del traslacional

Componente *RigidBody* en Unity



Simulación física

Unity3D(I)

Unity3D

Simulación física 3D basada en motor *PhysX* de NVIDIA

Simulación física 2D basada en motor open source *Box2D*

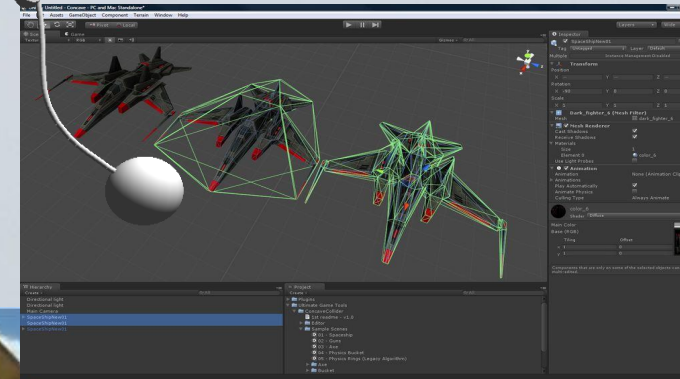
Elementos básicos

Cuerpo Rígido

Juntas

Controladores de personajes

Colisionadores



Simulación física

Unity3D(II)

Cuerpo Rígido (I)

Componente principal que permite el comportamiento físico de un objeto

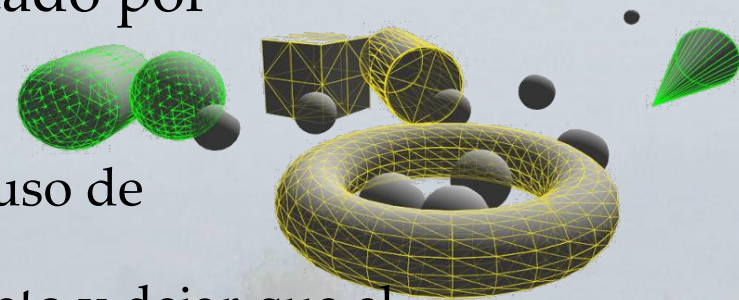
Permite que las fuerzas afecten al objeto, incluida la gravedad

Si se le añade al objeto un componente Colisionador, el objeto es afectado por colisiones con otros objetos

Desplazamiento del objeto

Deja de realizarse mediante uso de componente *Transform*

Debe aplicarse fuerzas al objeto y dejar que el motor de física calcule los resultados



Simulación física

Unity3D(III)

Cuerpo Rígido (II)

En algunos casos es necesario que un objeto tenga un cuerpo rígido **sin** tener su movimiento controlado por el motor de física

El personaje es directamente controlado mediante un *script*, pero todavía permiten que pueda ser detectado por *triggers*. Movimiento no simulado por física.

Movimiento cinemático

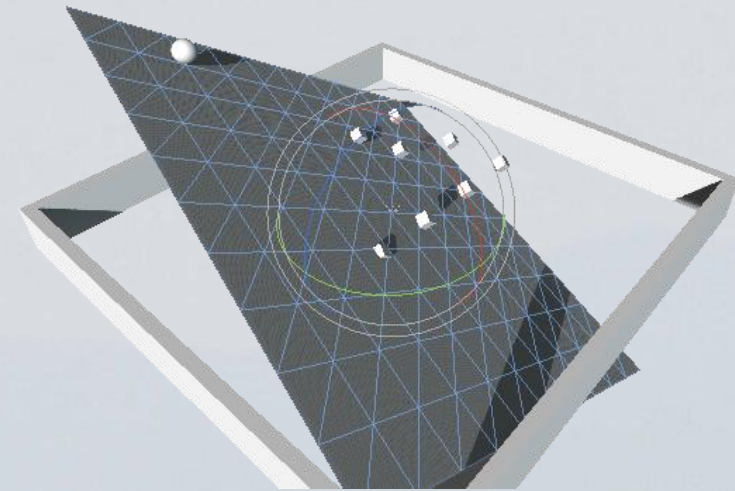
Propiedad *IsKinematic* a *true*, lo eliminará del control del motor de física y permitirá el control de transformación mediante un guión

Activar y desactivar la cinemática de un personaje sobrecarga el rendimiento del motor. Mejor no cambiar comportamiento si no es estrictamente imprescindible



Simulación física

Unity3D(IV)



Cuerpo Rígido (III)

Dormido/Despierto (I)

Existe umbral mínimo de velocidad lineal o de rotación. Por eficiencia computacional

Si un objeto se mueve por debajo de ese límite, el motor de física considera que ha llegado a su fin y lo duerme (detiene implícitamente). Extraído del grafo de simulación física

Los objetos dormidos NO

Se moverán de nuevo hasta que reciban una colisión o fuerzas que los pongan de nuevo en movimiento

Consumen tiempo de procesador



Simulación física

Unity3D(V)

Cuerpo Rígido (IV)

Dormido/Despierto (II)

Cambio de estado sucede de forma transparente al programador

Sin embargo, un objeto podría no despertar si un colisionador estático (uno sin un cuerpo rígido) colisiona o deja de hacerlo mediante la modificación de la posición de transformación (cuerpo cinemático). Típicamente una plataforma

Así, un objeto podría acabar flotando en el aire si el suelo se ha movido hacia abajo

En estos casos el objeto se puede reactivar de forma explícita utilizando el método *WakeUp()*



Simulación física

Unity3D(VI)

Cuerpo Rígido (V)

Propiedades

Masa. Unidades arbitrarias. Se asumen Kg

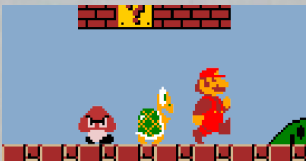
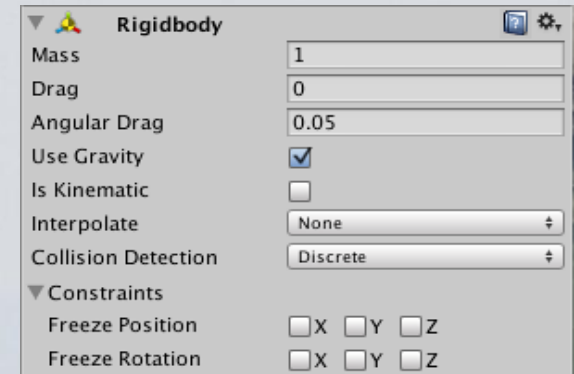
Diferencias entre objetos más pesados y menos ≤ 100

Drag. Fricción del aire. 0 = no resistencia del aire. Infinito = objeto se para de inmediato

Angular Drag. Fricción del aire cuando el objeto está girando.
0 = no resistencia del aire. Infinito = objeto giratorio no parará inmediatamente

Usar Gravedad. Si el objeto es afectado por la gravedad o no

IsKinematic. El objeto no se verá impulsado por el motor de física.
Sólo puede ser manipulado por su componente transform



Simulación física

Unity3D(VII)

Cuerpo Rígido (VI)

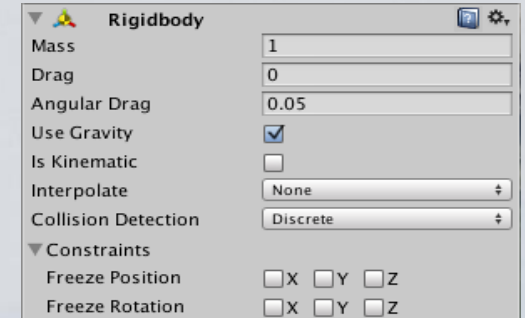
Propiedades

Interpolate. Define cómo se suaviza el cálculo de colisiones

Ninguno. No se aplica

Interpolar. Suaviza el movimiento basándose en la transformación anterior

Extrapolar. Suaviza el movimiento basándose en la transformación siguiente



Simulación física

Unity3D(VIII)

La forma de mover un objeto mediante el simulador de físicas

1. No es explícitamente mediante la cinemática: componente transform
2. Sino implícitamente aplicando fuerzas al objeto. El objeto se acelerará de acuerdo con la ley de Newton $F = m \cdot a$ de forma que, cuanto mayor sea la masa, mayor deberá ser la fuerza aplicada para poder cambiar su velocidad

En unity, se puede aplicar una fuerza al objeto mediante el método *AddForce*

```
public void AddForce(Vector3 fuerza, ForceMode mode = ForceMode.Force);
```

Donde

1. **fuerza** es un vector 3D que indica la dirección (X, Y y Z) y la intensidad de la fuerza aplicada
2. **mode** es la forma en la que se calcula la aceleración resultante sobre el objeto. Dos tipos: **Force** e **Impulse**



Simulación física

Unity3D(IX)

Los dos tipos de modos de aplicar la fuerza

1. **ForceMode.Force.** Si no se indica nada, es el valor por defecto que se asume al aplicar la fuerza. Aplica la parte proporcional de la fuerza en cada ciclo de *FixedUpdate*. Este modo es útil para generar efectos físicos realistas de forma que se necesita más fuerza para mover objetos más pesados
2. **ForceMode.Impulse.** Este modo es útil para aplicar fuerzas que ocurren instantáneamente, como las fuerzas de explosiones o colisiones. Toda la fuerza se aplica en un único fotograma y no a lo largo de un segundo

Ejemplos

```
private float empuje = 10.0f;
```

```
rb.AddForce(transform.up * empuje);
```

//Idéntico resultado tendría escribir el siguiente código

```
rb.AddForce(transform.up * empuje, ForceMode.Force);
```

```
rb.AddForce(transform.up * empuje, ForceMode.Impulse);
```

Simulación física

Unity3D(X)

Asúmase que

1. Se dispone de un objeto con una masa de 1Kg al que se le está aplicando una fuerza de 5N, en modo *ForceMode.Force*
2. Se está utilizando un paso de tiempo de 0.1s (10 fps)

Aplicando la ley de Newton $F = m \cdot a = m \cdot d/t^2$ y sustituyendo los términos, se tiene que

$F = m \cdot \frac{d}{t^2}$; $5 = 1 \cdot \frac{d}{0.1^2}$; es decir, que el objeto ha avanzado una distancia $d = 5 \cdot 0.01 = 0,05\text{m}$ en un único fotograma y su aceleración constante durante ese fotograma habrá sido de $a = F/m$

La fuerza sólo se aplica si se ejecuta *AddForce* en cada fotograma

ForceMode.Impulse aplica toda la fuerza que debería aplicarse durante un segundo en un único fotograma, reconvirtiendo de facto la fórmula de la fuerza en $F = m \cdot \frac{d}{t}$

En este caso, el resultado final es 10 veces más potente (10 fps)

$d = 5 \cdot 0.1 = 0,5\text{m}$

Resumiendo: *ForceMode.Force* aplica la fuerza por unidad de tiempo y *ForceMode.Impulse* la aplica toda de inmediato en un único fotograma

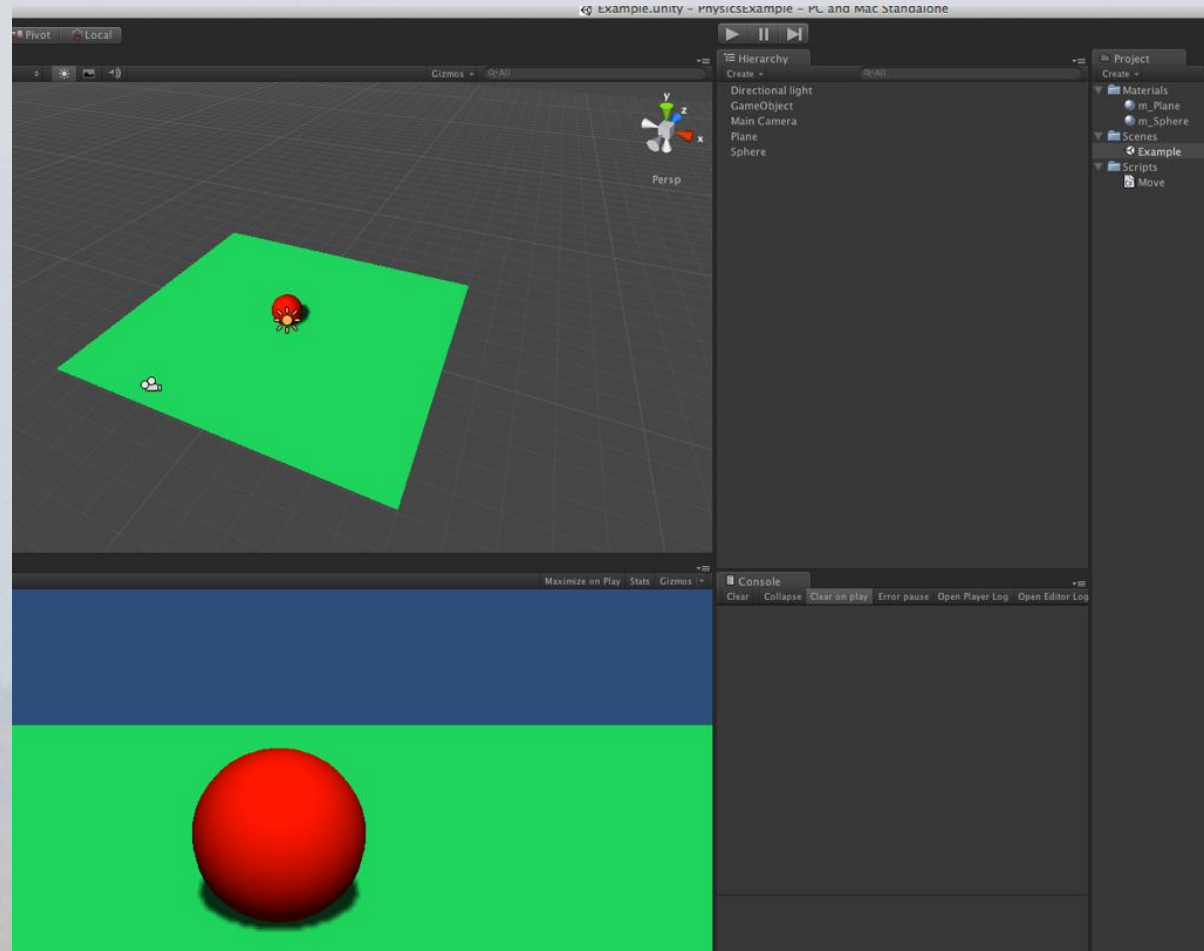
Un caso práctico (I)

Abrir el primer ejemplo
(FirstExample)

Este ejemplo contiene un
suelo de color verde y una
esfera sobre él

Objetivo

Conseguir
comportamiento más
realista utilizando el
motor de física



Un caso práctico (II)

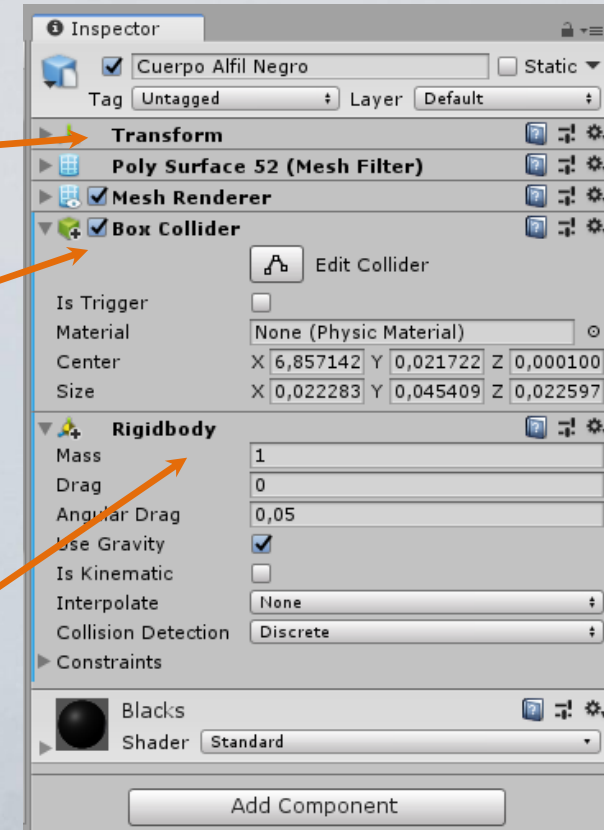
Por defecto, al insertar un objeto, sólo aparecen los componentes de transformación u el visor de malla

Añadir un componente *Colisionador*
el objeto es afectado por colisiones
con otros objetos
Objetos no se penetran entre si. Dan
sensación sólida

Añadir un componente *RigidBody*
Componente que permite la simulación
física de un objeto
Permite que las fuerzas afecten al objeto,
incluida la gravedad (*Use Gravity*)

Desplazamiento del objeto

Deja de realizarse mediante uso de
componente *Transform* (cinemática)
Debe aplicarse fuerzas al objeto y dejar que el
motor de física calcule los resultados
(dinámica)



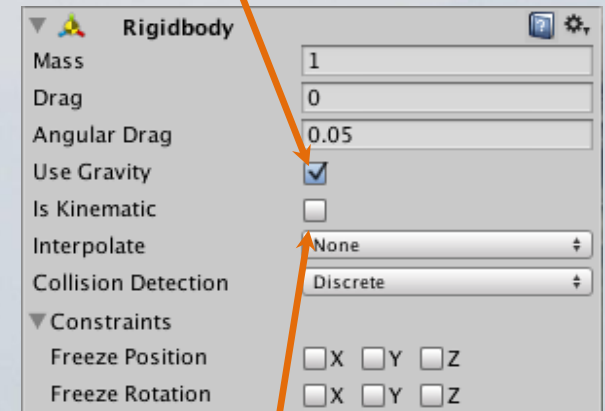
Un caso práctico (III)

En algunos casos es necesario que un objeto tenga un cuerpo rígido sin tener su movimiento controlado por el motor de física. Uso:

- El personaje es directamente controlado mediante un *script*. Movimiento no simulado por física. Movimiento cinemático
- Todavía puede ser detectado por *triggers*

(Des)Activar la cinemática de un personaje sobrecarga el rendimiento del motor. Mejor no cambiar comportamiento durante el juego si no es estrictamente imprescindible

Si no se desea que el objeto caiga aunque pueda seguir siendo simulado físicamente, desactivar la propiedad *Use Gravity*

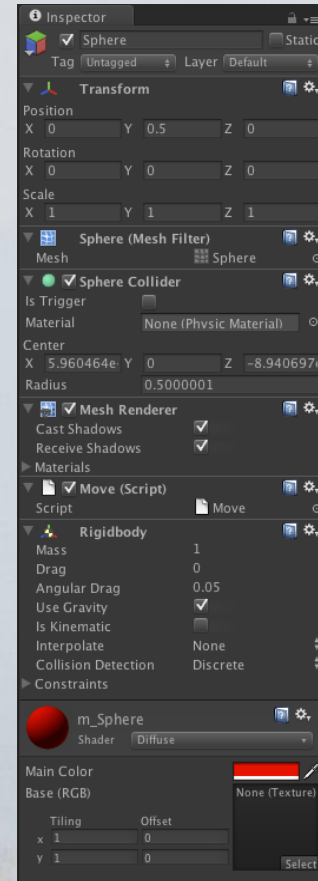


Propiedad *IsKinematic* a true, lo eliminará del control del motor de física y permitirá transformar a un objeto por medio de un *script*

Un caso práctico (IV)

La esfera ya dispone de un **Colisionador**
Se añade un *Rigidbody* (Componente->Físicas->Rigidbody)
Ahora hay que modificar el Movimiento existente en el script

```
void FixedUpdate()  
{  
    rigidBody.AddForce (Input.GetAxis("Horizontal"), 0.0f, Input.GetAxis("Vertical"));  
}
```



Un caso práctico (V)

Ahora la esfera rueda sobre la superficie del plano

La esfera se sitúa sobre el plano porque el plano tiene un **Collider**

Ejercicio

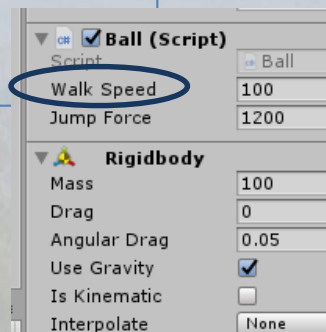
Experimentar con los cambios en *Drag*, *Angular Drag*

Usar una variable pública *walkSpeed* para regular desde interfaz la intensidad de la fuerza que se le aplica

Sugerencia

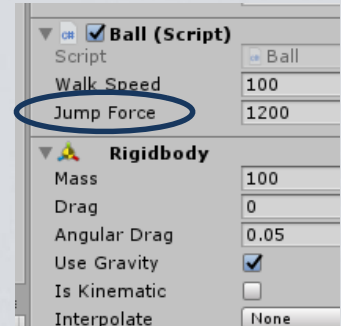
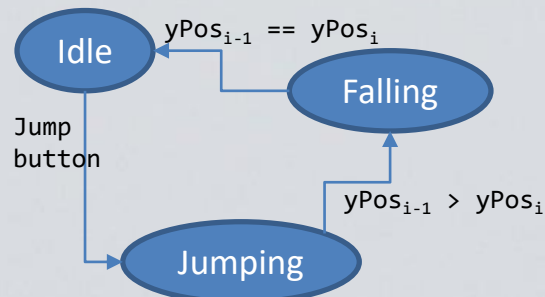
Se puede utilizar `Input.acceleration.x` (.y ó .z) en dispositivos móviles

```
private void FixedUpdate()    {  
    rigidBody.AddForce( Input.GetAxis("Vertical")    * walkSpeed, 0.0f,  
                        Input.GetAxis("Horizontal") * walkSpeed);  
}
```



Un caso práctico (VI)

También se puede hacer saltar la esfera



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

```
public class Ball : MonoBehaviour
{
```

```
    [SerializeField]
    float walkSpeed;
    [SerializeField]
    float jumpForce;
```

```
    enum States {IDLE, JUMPING, FALLING, MAX_STATES };
    States state = States.IDLE;
```

```
    Rigidbody rigidBody;
```

```
    float yPos;
```

```
    // Start is called before the first frame update
    void Start()
```

```
    {
        //define the animator attached to the player
        rigidBody = GetComponent<Rigidbody>();
        yPos      = transform.position.y;
    }
```

```
    // Update is called once per frame
```

```
    void Update()
```

```
    {
```

```
        //Ball state machine behaviour
```

```
        switch (state)
```

```
        {
```

```
            case States.JUMPING:
```

```
                if (yPos > transform.position.y)
                    state = States.FALLING;
                else yPos = transform.position.y;
                break;
```

```
            case States.FALLING:
```

```
                if (yPos == transform.position.y)
                    state = States.IDLE;
                else yPos = transform.position.y;
                break;
```

```
            case States.IDLE:
```

```
                if (Input.GetButton("Jump"))
                {
                    rigidBody.AddForce(Vector3.up * jumpForce,
                                         ForceMode.Impulse);
                    state = States.JUMPING;
                }
                break;
```

```
        }
```

Un caso práctico (VII)

También se puede hacer saltar la esfera

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

public class Ball : MonoBehaviour
{
    [SerializeField]
    float walkSpeed;
    [SerializeField]
    float jumpForce;

    enum States {IDLE, JUMPING, FALLING, MAX_STATES };
    States state = States.IDLE;

    Rigidbody rigidBody;

    float yPos;

    // Start is called before the first frame update
    void Start()
    {
        //define the animator attached to the player
        rigidBody = GetComponent<Rigidbody>();
        yPos = transform.position.y;
    }
}
```

Se pueda capturar su componente *rigidBody* junto con otros al comenzar el videojuego

```
// Update is called once per frame
void Update()
{
    //Ball state machine behaviour
    switch (state)
    {
        case States.JUMPING:
            if (yPos > transform.position.y)
                state = States.FALLING;
            else yPos = transform.position.y;
            break;
        case States.FALLING:
            if (yPos == transform.position.y)
                state = States.IDLE;
            else yPos = transform.position.y;
            break;
        case States.IDLE:
            if (Input.GetButton("Jump"))
            {
                rigidBody.AddForce(Vector3.up * jumpForce,
                                    ForceMode.Impulse);
                state = States.JUMPING;
            }
            break;
    }
}
```

Se emplea *ForceMode.Impulse* para una fuerza de efecto instantáneo

Mejora de la eficiencia capturando al principio los componentes necesarios para no tener que obtenerlos en cada pasada del bucle una y otra vez

Un caso práctico (VIII)

Ejercicio de física (I)

Sobre una escena nueva, colocar un suelo y añadirle un material de color verde

Añadir nuevo **GameObject** de tipo cubo

Escalarlo a un factor de escala (1,5,1)

Duplicarlo para hacer el otro pilar

Duplicarlo y rotarlo para hacer el techo

Montar un dolmen como el de la escena

Nombrar cada pieza por su nombre: **Techo**, **pilar dcho** y **pilar izq**

Insertar un empty denominado **Dolmen**

Agrupar los tres elementos dentro de Dolmen



Un caso práctico (IX)

Ejercicio de física (II)

Duplicar el grupo Dolmen y situarlo encima del anterior

Duplicar el doble dolmen y situarlo a la izquierda del original

Duplicarlo de nuevo y colocarlo a la derecha del original

Seleccionar todas sus piezas y añadirles un componente *RigidBody* no cinemático y que use la gravedad

El resto de elementos, incluido el suelo, no tiene ningún componente *RigidBody*

Seleccionar la pata izquierda que toca el suelo del dolmen de la derecha y añadirle a ella sola un componente *RigidBody*



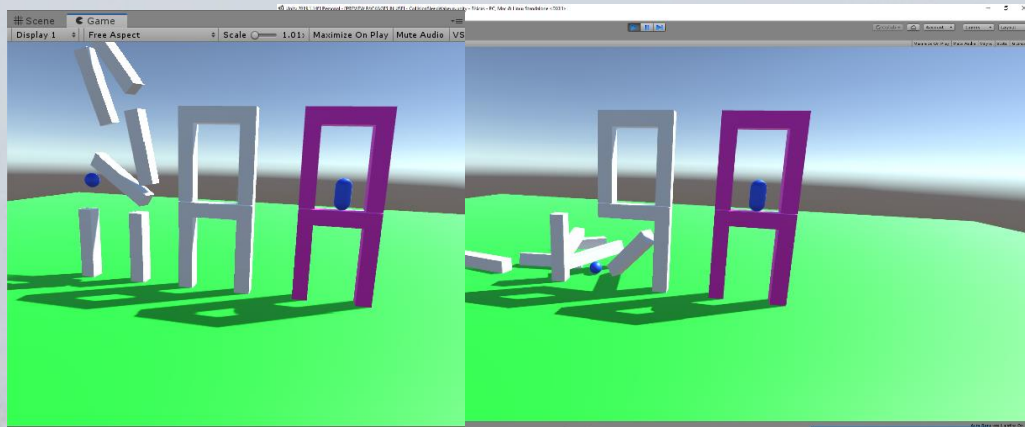
Un caso práctico (X)

Ejercicio de física (III)

Colocar una cápsula encima del larguero de la planta
baja del tercer dolmen en equilibrio sobre una tapa

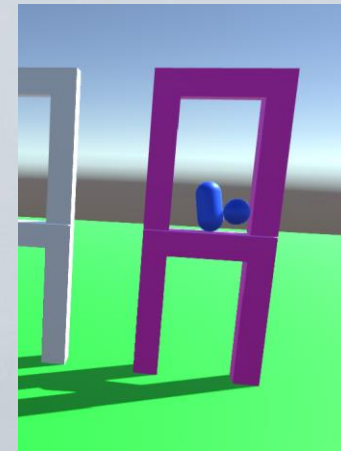
Ejecutar el videojuego e intentar chocar con los pilares
laterales de los tres dólmenes

¿Cuál es el comportamiento observado?



Todas las piezas del dolmen
de la izquierda obedecen a
las leyes de la física

Sólo la pata izquierda del
dolmen central obedece a
las leyes de la física
El resto de piezas son
inamovibles



La cápsula, obedece a las
leyes de la física
El resto del dolmen no.
Si se salta sobre la cápsula,
se le puede hacer caer

Un caso práctico (XI)


Modificar el script del objeto Bola para incluir ahora dos referencias a dos objetos

```
States state = States.IDLE;
```

```
Rigidbody    rigidBody;  
GameObject    larguero, capsula;
```



```
float yPos;
```

En este caso, la cápsula y un larguero



Modificar el método *Start* para actualizar al principio las referencias a los dos objetos

```
// Start is called before the first frame update  
void Start()  
{  
    //define the animator attached to the player  
    rigidBody = GetComponent<Rigidbody>();  
    larguero = GameObject.Find("Travesaño 1.edif2");  
    capsula = GameObject.Find("Capsula");  
    yPos = transform.position.y;  
}
```



Un caso práctico (XII)

Añadir al método *Update* el siguiente código

//The rest of the inputs

```
if (Input.GetKey(KeyCode.Z))  
{  
    capsula.GetComponent<Rigidbody>().Sleep();  
    larguero.transform.Translate(Vector3.forward * 2.0f);  
}
```

Cuando se pulse la tecla 'Z'

Se duerme al objeto cápsula

Se traslada
cinemáticamente al
larguero que está debajo

Se puede despertar a la cápsula
dormida y caerá por efecto de la
gravedad

```
if (Input.GetKey(KeyCode.X))  
{  
    capsula.GetComponent<Rigidbody>().WakeUp();  
    larguero.transform.Translate(Vector3.forward);  
}
```

Al estar la cápsula dormida, se
quedará flotando en el aire en lugar
de caer por efecto de la gravedad

Un caso práctico (XIII)

Probar a golpear la cápsula cuando se queda en el aire empleando la opción de salto de la esfera

Se han utilizado elementos nuevos

Input.GetButtonDown Averigua si cualquiera de los botones definidos en el Gestor de Entrada está pulsado. Será **cierto** únicamente en el fotograma exacto en el que el usuario haya pulsado el botón, **falso** después

Input.GetKey determina si la tecla indicada está pulsada

Simulación física

Unity3D(XI)

Juntas (Joints)

Enlaza un objeto rigidbody a otro o a un punto fijo en el espacio

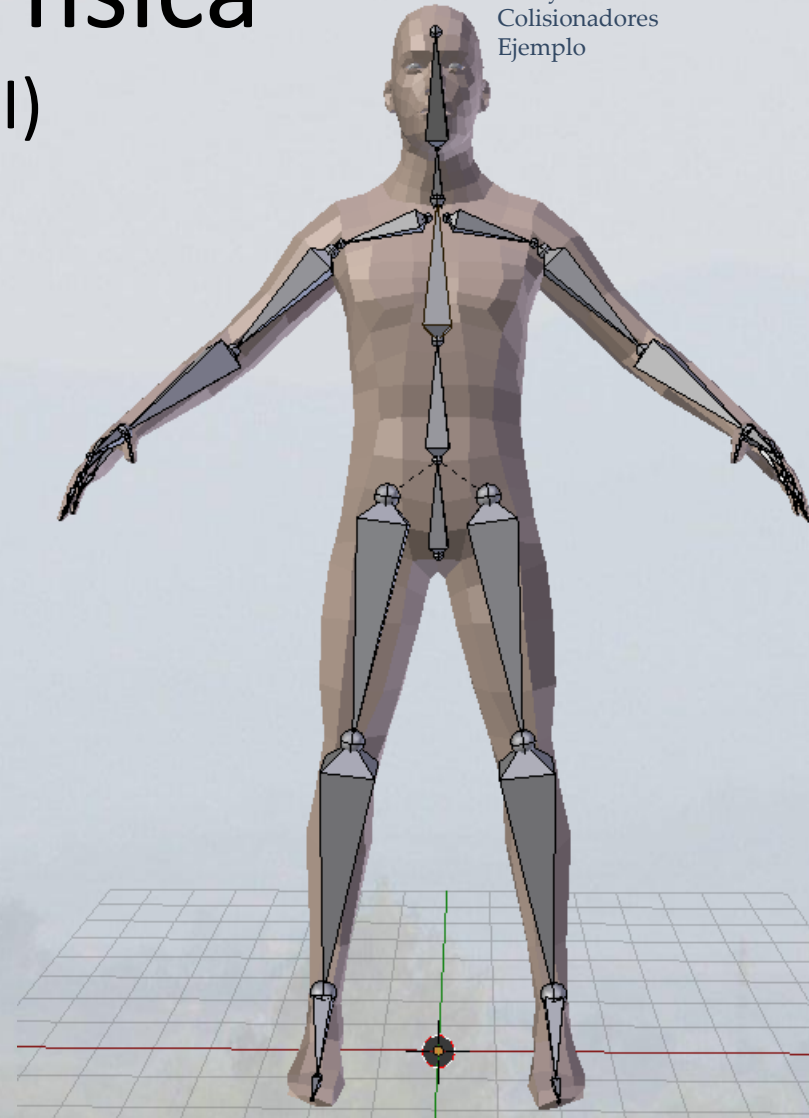
Soporta restricciones de movimiento:

- Bisagra (Hinge) permite la rotación alrededor de un punto y un eje específico

- Muelle mantiene dos objetos separados con una distancia entre ellos extensible

Puede establecerse un umbral de fuerza que rompa la articulación cuando se supere

Permitir fuerzas de accionamiento automáticas entre objetos conectados



Materiales físicos (I)

Cuando los colisionadores interactúan,
sus superficies deben simular las
propiedades físicas del material

Una capa de hielo será resbaladiza

Pelota de goma ofrece mucha fricción



Materiales físicos (II)

La fricción y el rebote se pueden configurar mediante la física de materiales

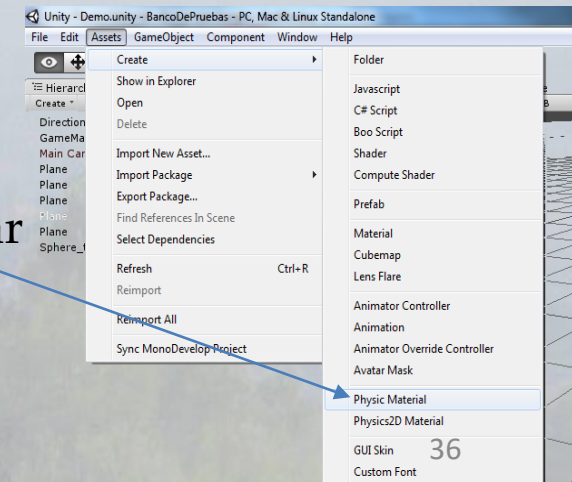
Obtención de los parámetros adecuados al videojuego suele obtenerse mediante prueba y error

Por razones históricas, al componente de física 3D se le denomina *Physic Material* y el equivalente 2D se denomina *Physics2D Material*

Crear un material nuevo tanto para la bola como para el tapete

Assets->Create->Physics Material

Darles el nombre del objeto sobre el que se va a asignar



Materiales físicos (III)

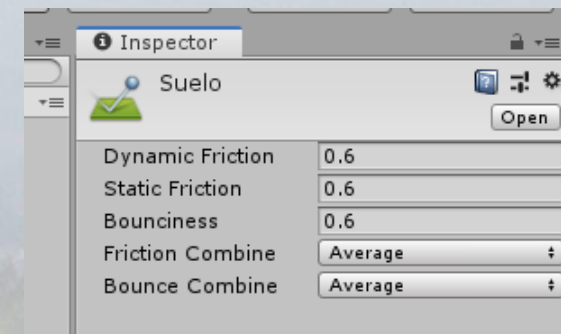
Duplicar las bolas. Cada una de ellas llevará asociado el mismo script

Crear un material nuevo para cada bola nueva creada y modificar sus componentes *Bounciness*

Hacer que el parámetro Bounciness del material bola sea 0.1, el de la bola1 sea 0.5 y el de bola2 sea 1

Observar al apretar el salto, como todas las bolas saltan por igual pero tanto la altura como la cantidad de los rebotes varían en función del tipo de material asignado a cada esfera

Si se quiere afectar por igual a todas las esferas, ¿qué material de tendrá que modificar?



Materiales físicos (IV)

Generar un nuevo cubo y situarlo encima del suelo

Asignarle un componente *RigidBody*

Duplicarlo dos veces y separar

Asignar a cada uno de ellos el mismo material físico que para las tres bolas

Modificar cada material para que el primero tenga los siguientes valores

Fricción	Bola	Bola1	Bola2
Estática	0,1	0,4	0,8
Dinámica	0,05	0,3	0,7

Asignar al suelo un *RigidBody* de tipo cinemático no afectado por la gravedad

Materiales físicos (V)

Generar un nuevo script denominado suelo y modificarlo según modelo adjunto

```
public class Suelo : MonoBehaviour
{
    [SerializeField]
    float angle;

    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.R)) {
            transform.Rotate(0.0f, 0.0f, angle);
        }
    }
}
```

Asociarlo al objeto suelo

Darle al atributo angle el valor de 0.5 grados en el inspector

Observar el comportamiento del sistema a medida que se va rotando el suelo en ejecución al apretar la tecla 'R'

Observar el comportamiento de deslizamiento de los cubos dependiendo de los coeficientes de rozamiento asignados

Detección y Resolución de colisiones (I)

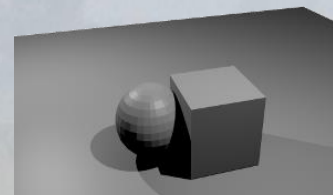
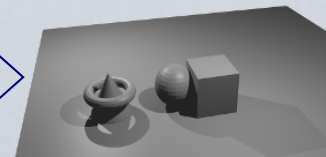
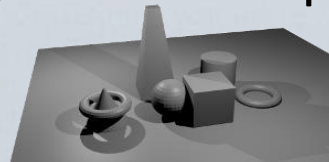
Objetivo

Detectar cuándo y dónde se superponen las superficies de dos objetos

Clasificación según

Instante de ejecución en pipeline detección de colisiones

1. Gruesa
2. Media
3. Fina



Detección y Resolución de colisiones (II)

Detección Gruesa

Fase de criba

Detección de potenciales colisiones

Falsos positivos pero nunca falsos negativos

Prioridad: velocidad detección pares potencialmente colisionantes



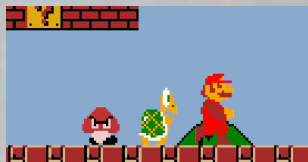
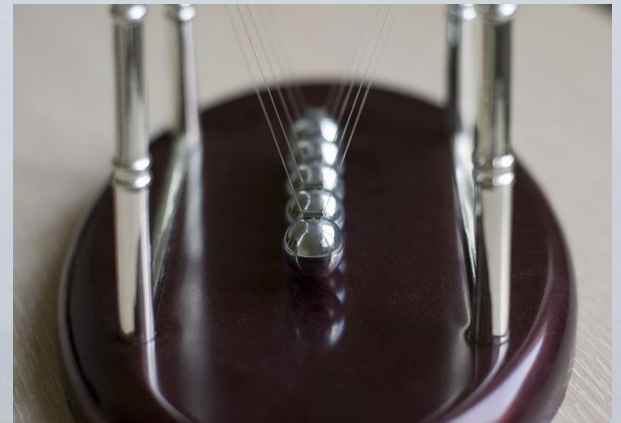
Detección y Resolución de colisiones (III)

Detección Media

Refinado de fase de criba anterior

Simplificar trabajo de fase fina

Prioridad: Refinar lista de falsos positivos



Detección y Resolución de colisiones (IV)

Detección Fina

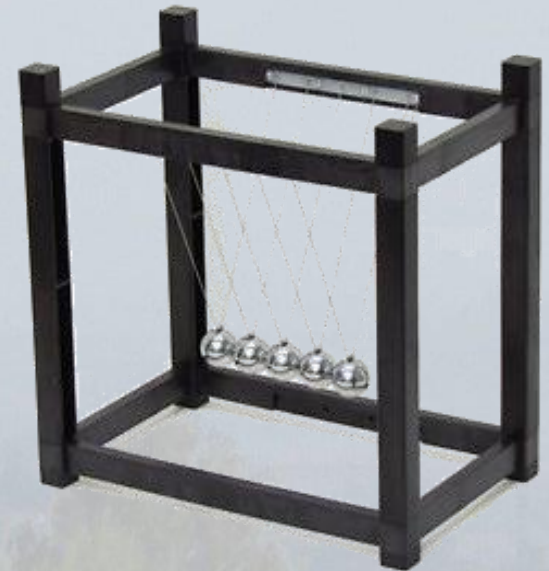
Refinado de fase anterior

Objetivo

Detectar t^0 exacto de colisión

Parte de la geometría interseccionada

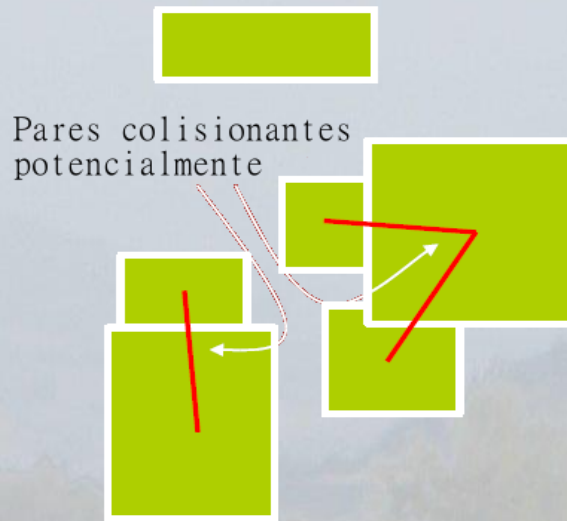
Rechazar falsos positivos



Detección y Resolución de colisiones (V)

Fase gruesa

Localización de
pares potenciales



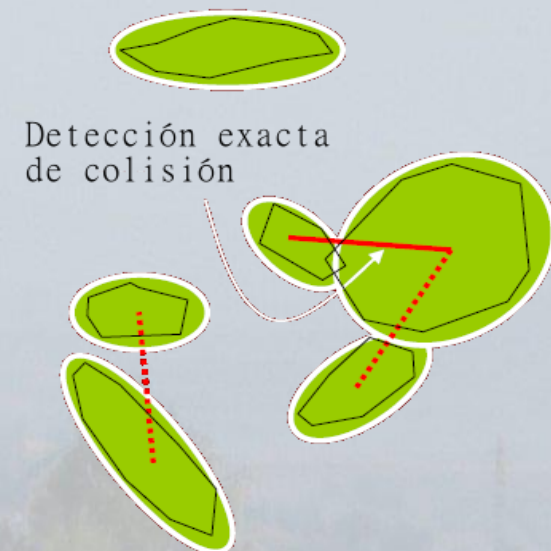
Fase media

Refinamiento de
pares potenciales



Fase fina

Localización de
colisiones exactas



Detección y Resolución de colisiones (VI)

Existe una geometría

Visual

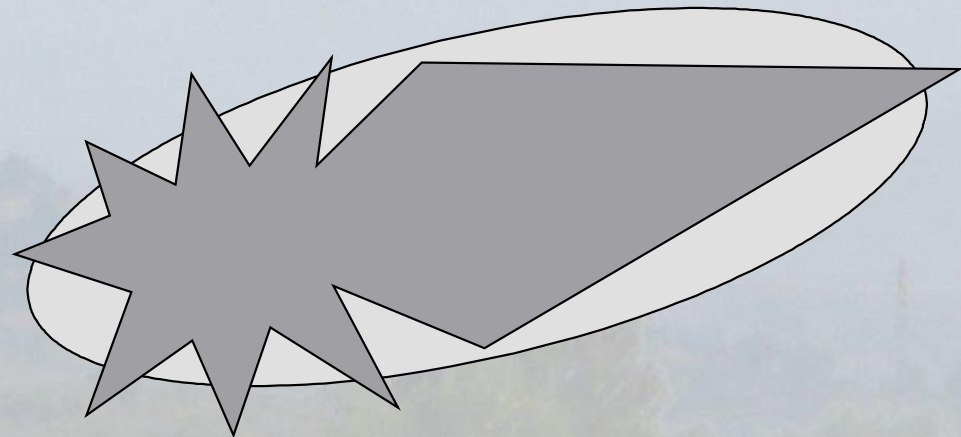
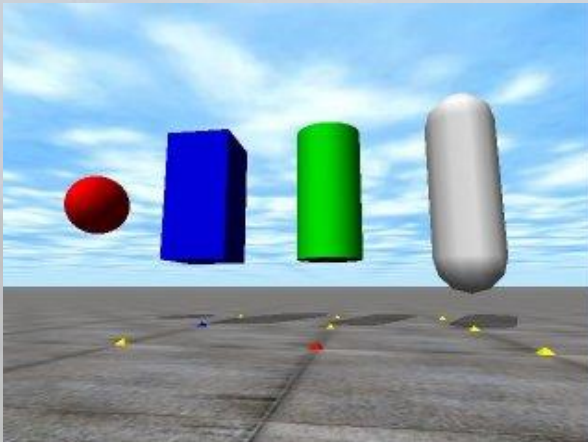
Malla visible en el render. Texturable

Colisión

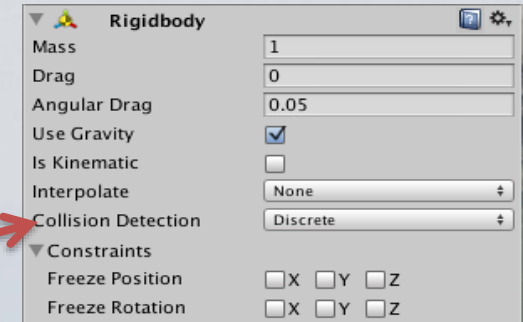
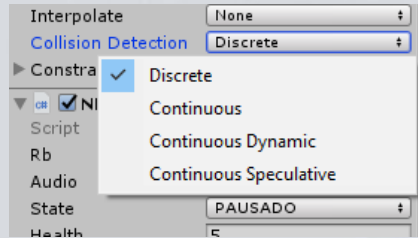
Aproximación burda de la visual mediante geometrías simples: esferas, elipsoides, cajas de inclusión,...

Envuelve completamente a visual

Es INVISIBLE



Detección y Resolución de colisiones Unity 3D (I)



Cuerpo Rígido (I)

Propiedades

Detección de Colisiones. Define el método de cálculo empleado para la detección de colisiones. Si no hay un componente *Collider*, no se aplica

Se utiliza para evitar que los objetos que se mueven rápidamente pasen a través de otros objetos sin detectar colisiones.

Discrete. Valor por defecto. El más rápido. Si no hay problemas de colisiones, no tocar esta opción. Se evalúa la colisión una vez por paso de interpolación

Continuo.

Continuo dinámico. Modo continuo para objetos muy veloces: disparos,... Más costoso computacionalmente. Usar sólo en caso de que el resto de métodos no funcionen

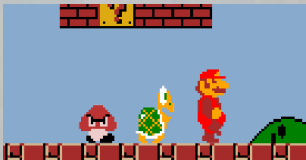
Continuo especulativo. Diseñado para gestionar con menor coste computacional el choque de geometrías estáticas y dinámicas y cinemáticos

Detección y Resolución de colisiones
Unity 3D (II)

Cuerpo Rígido (II)

Algoritmos de detección de colisiones

	Discreto	Continuo	Continuo Dinámico	Static Mesh Collider (no Rig. Body.)
Discreto	Discreto	Discreto	Discreto	Discreto
Continuo	Discreto	Continuo	Continuo	Continuo
Continuo Dinámico	Discreto	Continuo	Continuo dinámico	Continuo



Detección y Resolución de colisiones Unity 3D (III)

Colisionadores/Colliders (I)

La detección de colisiones implica la existencia de colisionadores o *Collider* (también para proyección de rayos o eventos del tipo *OnMouseDown*)

Puede ser de una geometría diferente a la visualizada

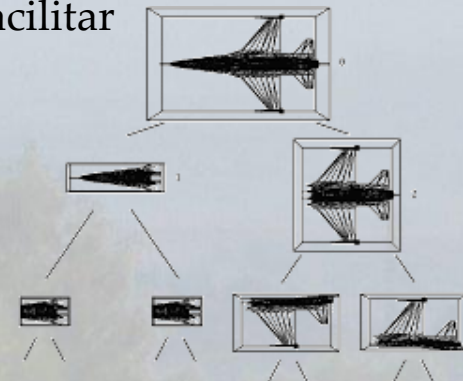
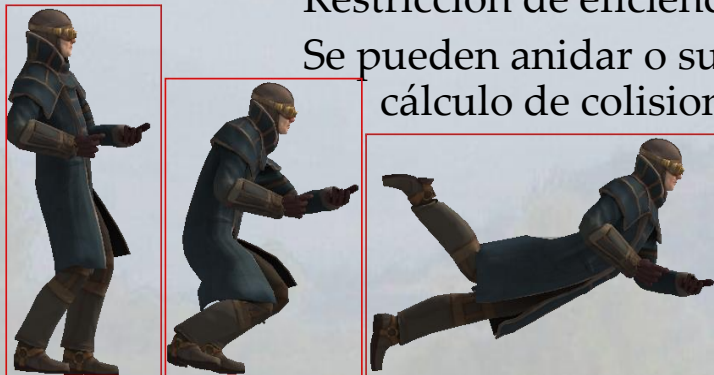
Cualquier malla puede

- Ser utilizada como Colisionador

- Colisionar con cualquier otro tipo de malla colisión

Restricción de eficiencia computacional: utilizar mallas convexas

Se pueden anidar o subdividir en otras más sencillas para facilitar cálculo de colisiones jerárquica



Detección y Resolución de colisiones Unity 3D (IV)

Colisionadores/Colliders (II)

Se puede crear un *GameObject* reactivo que

Sólo tenga un *Collider*

Ninguna malla ni ningún renderer

Cuando el personaje principal del juego toque a este reactivo (colisione con el *Collider*) se pueda disparar una acción concreta:

Nuevos enemigos pueden aparecer

Alguna acción o mecanismo puede empezar

Una puerta puede abrirse/cerrarse



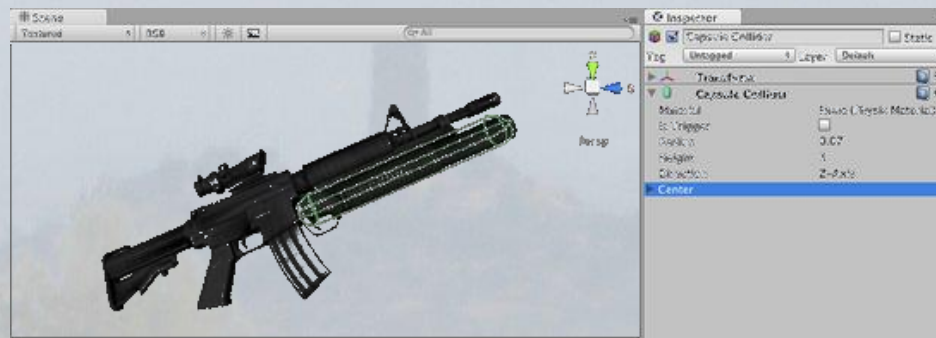
Detección y Resolución de colisiones Unity 3D (V)

Colisionadores/Colliders (III)

Componente que permite detectar colisiones entre mallas de objetos
Carcasa/envoltorio externo simplificado que envuelve completamente a la malla del objeto

Cuerpo de colisión primario

Si dos Rigidbodyes chocan entre sí, el motor de física no calculará una colisión a menos que ambos objetos tengan también un Collider adjunto



Detección y Resolución de colisiones Unity 3D (VI)

Colisionadores (IV)

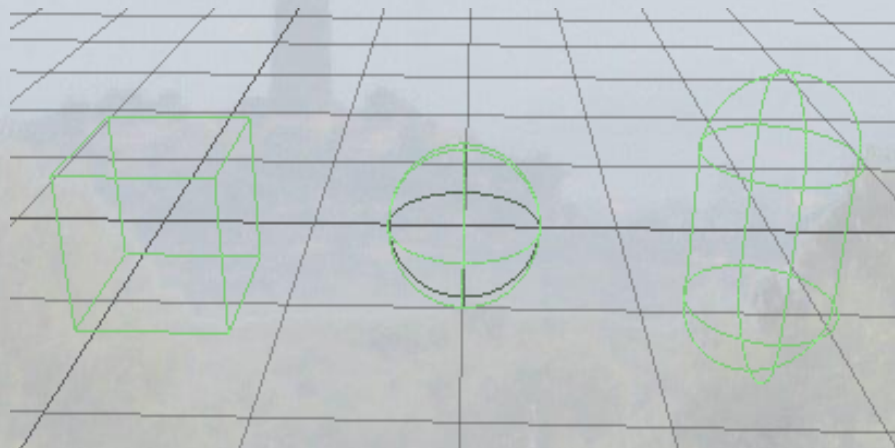
Rigidbody Collider-less simplemente dejará pasar uno a través de del otro durante la simulación física

Se pueden añadir sin *Rigidbody* en objetos estáticos: suelos, paredes, fondos con los que se puede colisionar,...

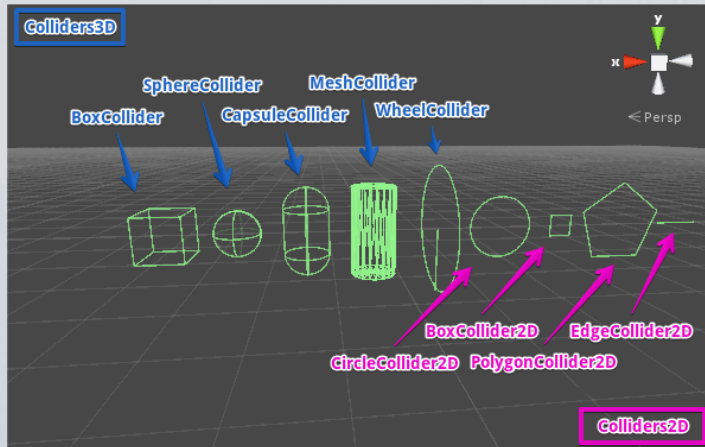
La forma de colisionadores no se deforma durante las colisiones

Aproximación más eficiente computacionalmente. Resultados prácticos análogos en tiempo de ejecución

Cuerpo de colisión primario diferente del visual (malla o Sprite)

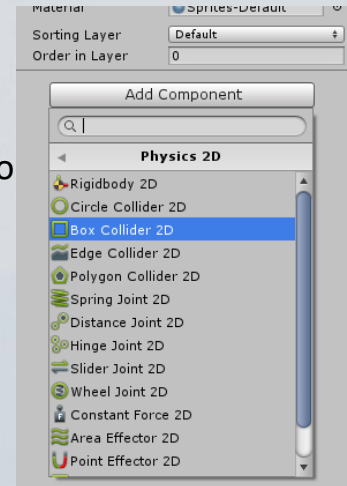


Detección y Resolución de colisiones Unity 3D (VII)



En 2D existen fundamentalmente cuatro tipos:

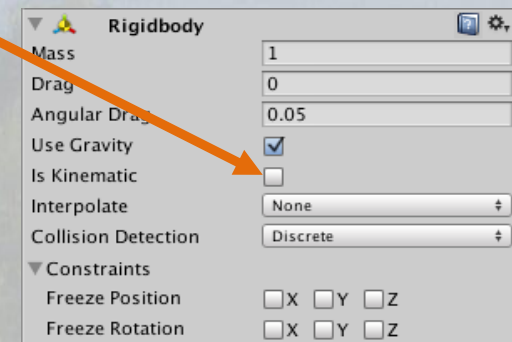
Circular
Caja
Lado
Poligonal



Los *Colliders* interactúan de manera diferente dependiendo en cómo están configurados sus *Rigidbody*

Las tres configuraciones:

- *Static Collider* (sin *Rigidbody*). Objetos estáticos que siempre se mantienen en su mismo lugar y nunca se mueven: suelos, paredes, fondos con los que se puede colisionar,...
- *Rigidbody Collider*. Caso más habitual
- *Kinematic Rigidbody Collider*. El caso anterior pero con la propiedad *isKinematic* activada



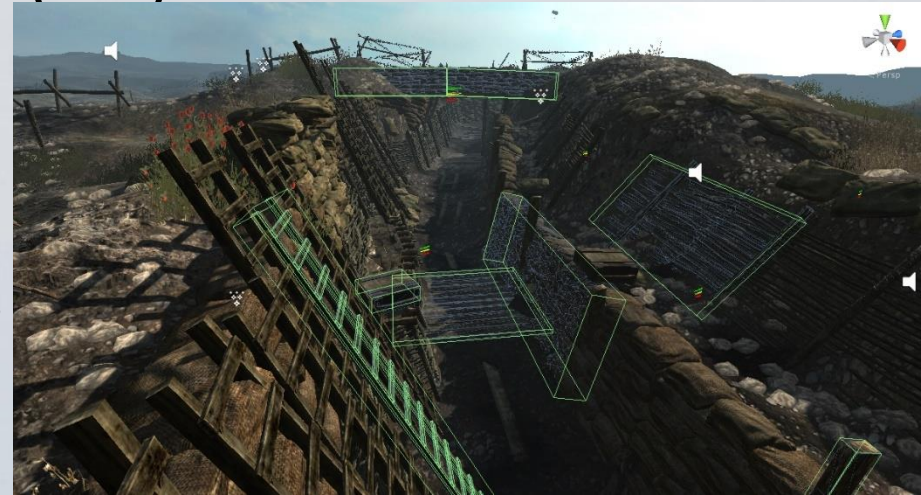
Detección y Resolución de colisiones Unity 3D (VIII)

Static Collider (sin *Rigidbody*)

Los objetos con *Rigidbody* que los colisionen no los moverán

El motor de física

1. Asume que los objetos estáticos nunca se mueven o cambian
2. Realiza optimizaciones internas



Por lo tanto, los objetos estáticos no deberían ser activados/desactivados, movidos o escalados durante el juego. Transformar un objeto estático

1. Obliga a recálculos internos por el motor de física que baja mucho el rendimiento
2. Puede dejar, a veces, el *collider* en un estado indefinido que produzca cálculos erróneos
3. Puede generar una colisión con *Rigidbody*s. Esta colisión no tiene por qué despertar al objeto colisionado si estaba dormido, ni tampoco aplicará fricción

Por lo tanto, si se desea mover un objeto estático,

1. Convertirlo en dinámico añadiéndole un *Rigidbody* y moverlo dinámicamente
2. Cinemáticamente desde un script, es conveniente adjuntar un componente *Kinematic Rigidbody*

Detección y Resolución de colisiones Unity 3D (IX)

Rigidbody Collider

Este es un *GameObject* con un

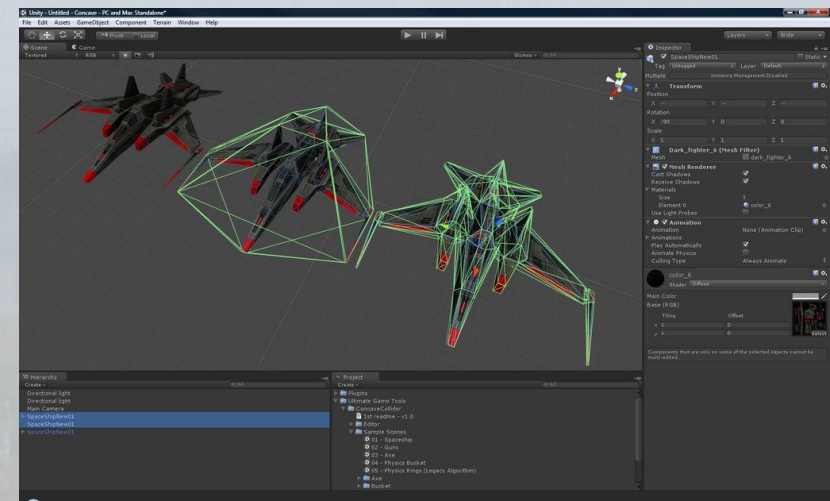
- Collider
- Rigidbody no cinemático (propiedad *isKinematic* **desactivada**)

Este es la configuración más comúnmente utilizada por la mayoría de los objetos en juegos que utilizan física

Son completamente simulados por el motor de física

Pueden

- Reaccionar a colisiones y fuerzas aplicadas desde un script
- Colisionar con otros objetos, incluyendo objetos estáticos



Detección y Resolución de colisiones Unity 3D (X)

Kinematic Rigidbody Collider (I)

Este es un *GameObject* con un

- Collider
- Rigidbody cinemático (propiedad *isKinematic* **activada**)

Se emplea para mover un objeto desde un script modificando su componente Transform

No responde a colisiones y fuerzas

Puede ser desactivados/activados ocasionalmente

Ejemplo:

- Puerta que normalmente actúa como un obstáculo inamovible pero que se puede abrir cuando sea necesario
- Animación de marioneta cinemática (*isKinematic* **activado**) que debe saltar de forma realista al recibir el impacto de una explosión cercana (*isKinematic* **desactivado**)

Al contrario de un static collider, un rigidbody cinemático en movimiento aplica fricción a otros objetos y despierta a otros rigidbodies cuando entren en contacto

Detección y Resolución de colisiones Unity 3D (XI)

Kinematic Rigidbody Collider (II)

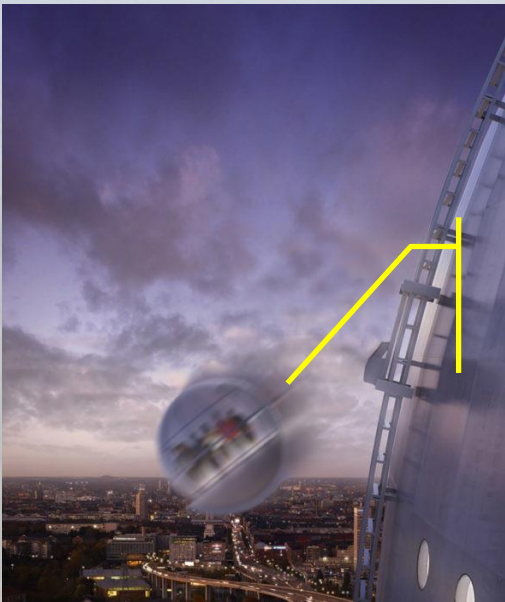
Un ascensor

En un momento determinado, al ascensor se le desactiva la propiedad `isKinematic`. Se convierte en un objeto físico normal
Pasa a ser controlado por la simulación física

No se mueve dinámicamente
Es un objeto cinemático
Se mueve por scripts

Ascensor cinemático
afecta a objetos
físicos dentro de él
Objetos internos no
atraviesan suelo del
ascensor

Si el objeto tiene activado el
parámetro "Use Gravity", caerá
junto sus objetos internos



Detección y Resolución de colisiones

Unity 3D (XII)

Las colisiones se resuelven en tres fases

1. Prólogo

- Se activa cuando la colisión ha ocurrido
- Determinar si la colisión debe ser ignorada
- Disparar otros eventos
 - Efectos de sonido
 - Envío de notificaciones de colisión

2. Colisión

- Situar los objetos sobre el punto de impacto
- Asignar nuevas velocidades
 - Empleo de ecuaciones físicas
 - Empleo de alguna lógica de decisión

3. Epílogo

- Propagar efectos post-colisión
- Posibles efectos
 - Destruir uno o ambos objetos colisionados
 - Reproducir un efecto de sonido
 - Infligir un daño a uno o ambos objetos:
 - deformación de malla, cambio en la textura,...
- Algunos de estos efectos pueden hacerse también en la fase de prólogo

Detección y Resolución de colisiones

Unity 3D (XIII)

Unity (I)

Cualquier objeto que herede de *MonoBehaviour* dispone de los métodos *OnCollision*

OnCollisionEnter() se invoca al producirse una colisión (fase de prólogo)

OnCollisionStay() mientras esta colisión está ocurriendo (fase de colisión)

OnCollisionExit () cuándo la colisión finaliza (fase de epílogo)

Las funciones en 2D acaban con el sufijo 2d: *OnCollisionEnter2d()*, *OnCollisionStay2d()* y *OnCollisionExit2d ()*

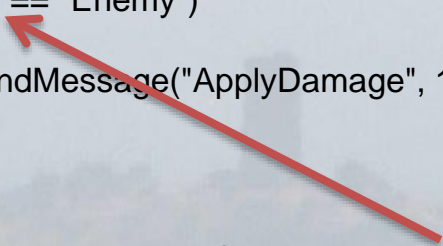
Detección y Resolución de colisiones Unity 3D (XIV)

Cuando se produce el contacto entre dos objetos, se invoca a cada uno y se le pasa como parámetro un colisionador que permite acceder al objeto con el que se ha colisionado

```
void OnCollision{Enter/Stay/Exit}(Collision)
```

Ejemplo

```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "Enemy")
    {
        collision.gameObject.SendMessage("ApplyDamage", 10);
    }
}
```



Por ello es conveniente utilizar **Etiquetas**

Las etiquetas pueden asignarse a *GameObjects* y agruparles en un tipo concreto o familia (enemigos, obstáculos, etc)

Detección y Resolución de colisiones Unity 3D (XV)

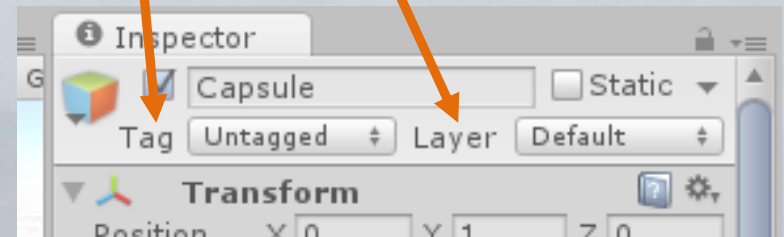
Etiquetar objetos en Unity (I)

Una etiqueta es una palabra de referencia que puede asignarse a los *GameObjects*. Se puede etiquetar con la etiqueta

- "Jugador" a todos los personajes controlados por el jugador
- "Enemigo" a todos los adversarios no controlados por el jugador
- "Coleccionable" a todos los elementos que el jugador puede recoger en una escena

El **inspector** muestra los menús desplegables de **etiqueta** y **capa** justo debajo del nombre de cualquier *GameObject*

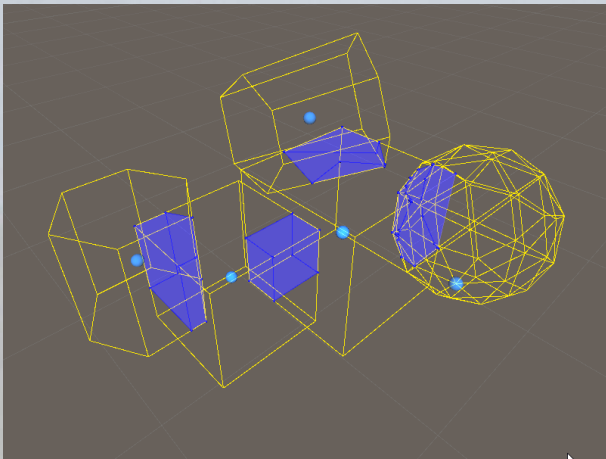
La función *GameObject.FindWithTag ()* se emplea para encontrar un *GameObject* que contenga la etiqueta deseada



Detección y Resolución de colisiones Unity 3D (XVI)

Si colisionan dos objetos

- Y uno de los dos objetos no dispone de Mesh Collider, los objetos se interseccionan. No se detectan colisiones
- La colisión sólo puede realizarse si uno de los dos objetos dispone de al menos un Rigidbody
- Detección de colisiones se realiza por sistema de simulación física; es decir, por dinámica. No se detecta por scripts cinemáticos



Detección y Resolución de colisiones Unity 3D (XVII)

Detección de área (Trigger/disparador)

Se quiere seguir reaccionando frente a colisiones

Un objeto disparador no registra una colisión con el Rigidbody de un objeto contra el que colisiona. No activa métodos virtuales *OnCollision*

Se marcará en el componente colisionador de los objetos disparadores el atributo *isTrigger*

Generará sobre el resto de objetos eventos *OnTrigger*:

OnTriggerEnter() (fase de prólogo)

OnTriggerStay() (fase de disparo)

OnTriggerExit () (fase de epílogo)



Detección y Resolución de colisiones Unity 3D (XVIII)

Detección de área (Trigger/disparador)

Este mensaje se envía tanto al Colisionador como al Cuerpo Rígido:

- 1.- Si no existe el Cuerpo rígido, entonces sólo al Colisionador
- 2.- Si no hay Colisionador, no se producen estos eventos

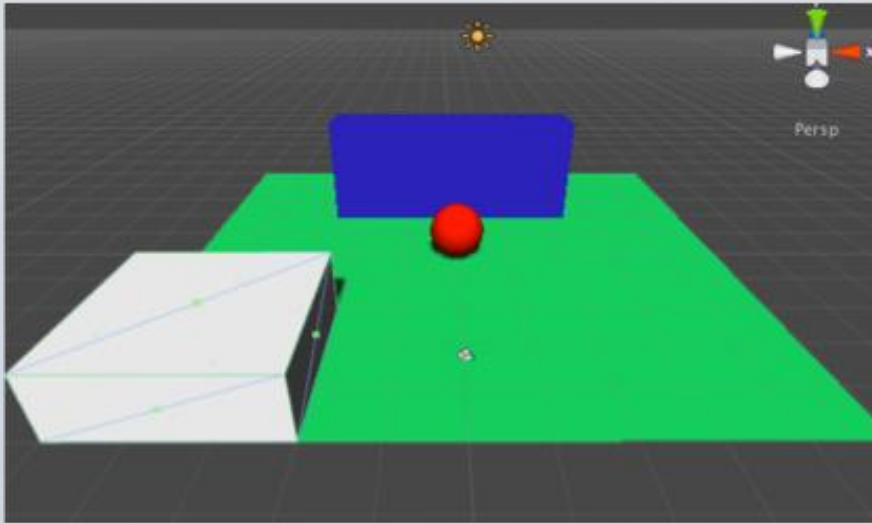
Los eventos de disparo se enviarán a MonoBehaviours desactivados, para permitir comportamientos habilitantes en respuesta a colisiones

OnTriggerEnter ocurre en el FixedUpdate después de una colisión

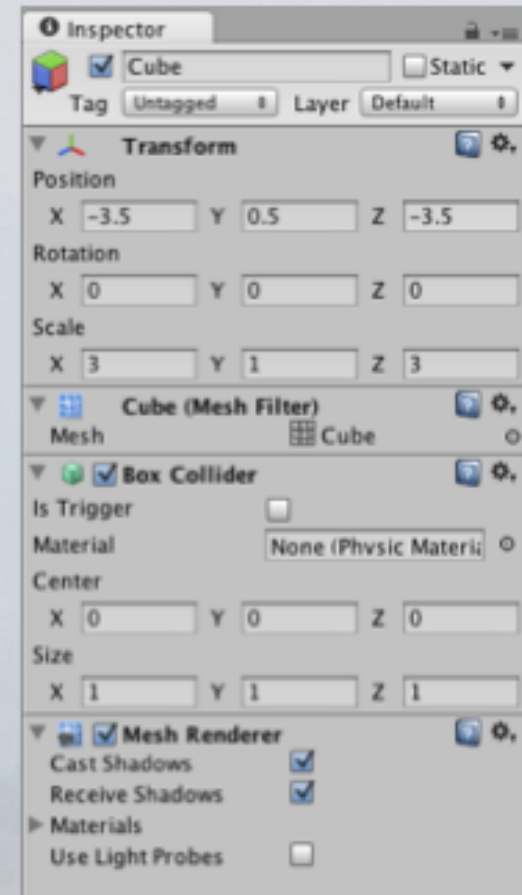


Detección y Resolución de colisiones

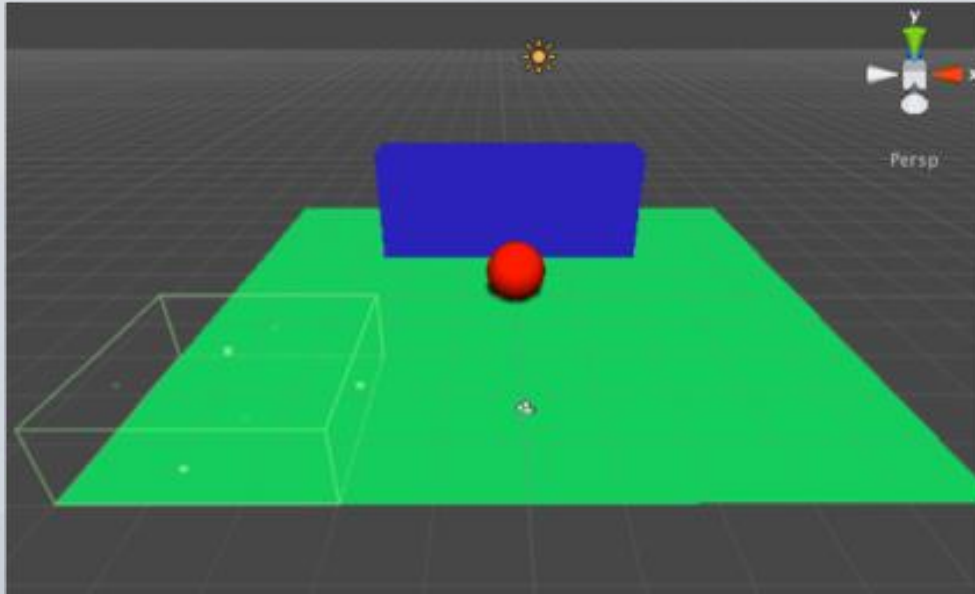
Ejemplo de colisiones (I)



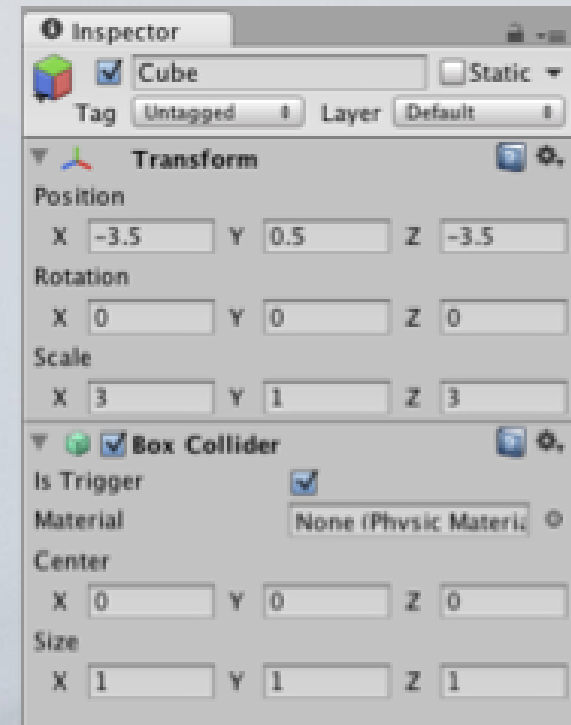
Crear un cubo, escalarlo y moverlo para cubrir una esquina del suelo plano



Detección y Resolución de colisiones Ejemplo de colisiones (II)



Eliminar o desactivar el *Filtro de Malla* y el *Renderer*
Sólo se necesita el *Collider* y activar *Is Trigger*



Detección y Resolución de colisiones

Ejemplo de colisiones (III)

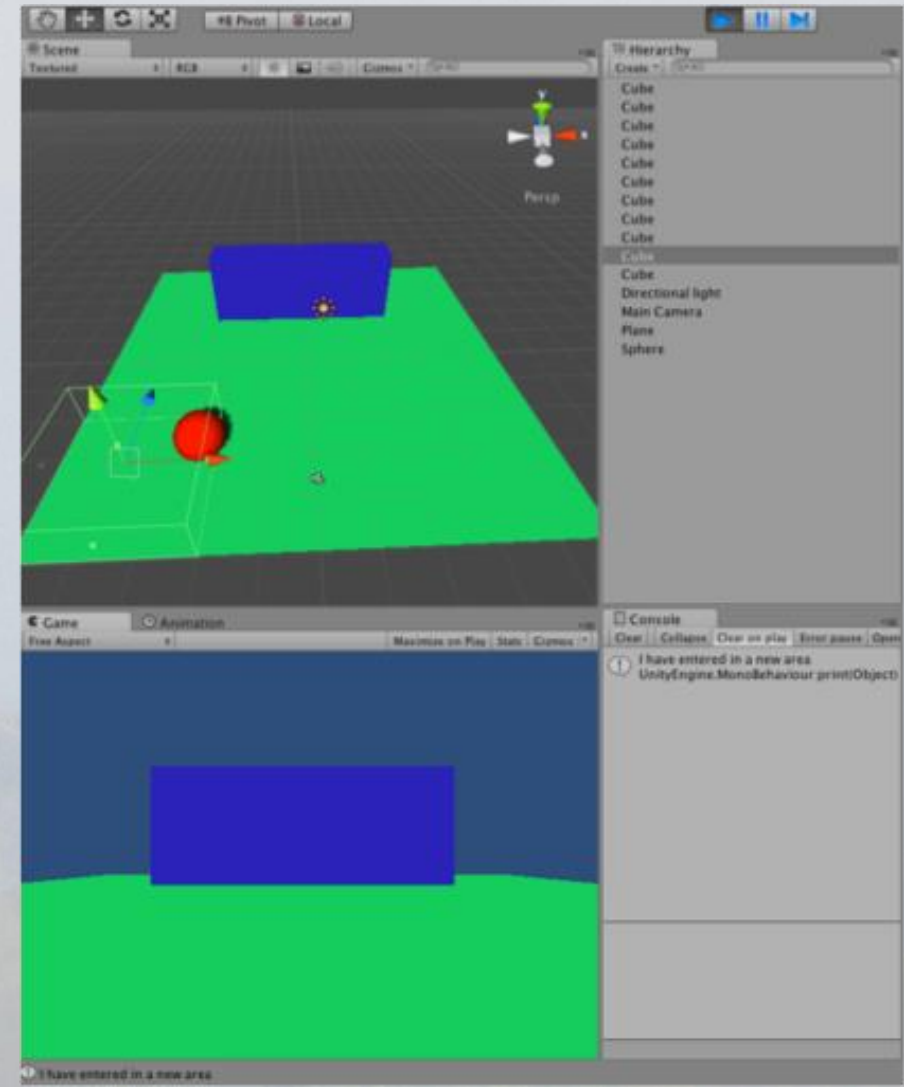
```
void OnTriggerEnter(Collider other) {  
    print ("Entrando en el area");}
```

Añadir al guión **Move** el método **OnTriggerEnter** donde se lleva a cabo la acción. En este caso sólo una instrucción *printf*

En el parámetro **other** del método **OnTriggerEnter**, se hace referencia al **GameObject** con el que la esfera está colisionando



Detección y Resolución de colisiones Ejemplo de colisiones (IV)



Detección y Resolución de colisiones

Ejemplo de colisiones (V)

OnBecameVisible(), OnBecameInvisible()

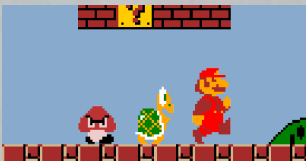
Estos dos métodos se invocan cuándo el objeto se hace visible o no para cualquiera de las cámaras activas

Son especialmente útiles cuándo el comportamiento de estos objetos consume bastante tiempo de CPU

Si existen muchos de estos objetos en la escena, habilitarlos cuando están o no visibles para la cámara podría ahorrar mucho proceso de cálculo

```
void OnBecameVisible() {  
    print ("Enabling");  
    enabled = true;  
}  
  
void OnBecameInvisible() {  
    print ("Disabling");  
    enabled = false;  
}
```

Si el objeto se inutiliza (enabled = false), entonces los métodos Update(), FixeUpdate(), etc. no se invocan



References

- Unity Game Development Essentials, Will Goldstone, Ed. Packt publishing, Cap. I. ISBN: 978-1-847198-18-1
- Cap 4.2 y 4.3. Introduction to Game Development. Steve Rabin. Charles River Media ISBN: 978-1-58450-377-4
- Cap 8 y 9. 3D Game engine design. David H. Eberly. Elsevier ISBN: 978-0-12-229063-3

Documentación generada por
Grupo de Informática Gráfica
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Reconocimiento-NoComercial-CompartirIgual 2.5

Usted es libre de:

copiar, distribuir y comunicar públicamente la obra
hacer obras derivadas bajo las condiciones siguientes:



Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador.



No comercial. No puede utilizar esta obra para fines comerciales.



Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.