Unity

Máster en Programación de Videojuegos

Ignacio Martínez Rodríguez

Curso 2013-2014

GameObject

- Los GameObjects son básicamente contenedores para componentes, que son los que implementan funcionalidad.
- Todos los objetos del juego heredan de GameObject.
- Tienen siempre obligatoriamente un componente Transform

GameObject: funciones

T AddComponent<T>();

Añade el componente del tipo T al objeto

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public SphereCollider sc;
    void Example() {
        sc = gameObject.AddComponent<SphereCollider>();
    }
}
```

GameObject: funciones

T GetComponent<T>();

Devuelve el componente de tipo T si el objeto tiene uno, null en caso contrario.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public SphereCollider sc;
    void Example() {
        sc = gameObject.GetComponent<SphereCollider>();
    }
}
```

GameObject: funciones

T GetComponentInChildren<T>();

Devuelve el componente de tipo T si el objeto tiene uno o si no busca en sus hijos la primera ocurrencia.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public RigidBody rb;
    void Example() {
        rb = gameObject.GetComponentInChildre<RigidBody>();
    }
}
```

GameObject: funciones

T[] GetComponentsInChildren<T>();

Devuelve un array con todos los componente de tipo T si el objeto tiene uno o si no busca en todos sus hijos.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public RigidBody[] rigidbodies;
    void Example() {
           rigidbodies = gameObject.GetComponentsInChildre<RigidBody>();
    }
}
```

GameObject: funciones

static GameObject Find(string name);

Busca un objeto en la escena por nombre.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public GameObject hand;
    void Example() {
        hand = GameObject.Find("Hand");
        hand = GameObject.Find("/Hand");
        hand = GameObject.Find("MonsterArm/Hand");
        hand = GameObject.Find("MonsterArmHand");
    }
}
```

GameObject: funciones

static Object Instantiate(Object original, Vector3 position, Quaternion rotation);

Crea una nueva instancia del objeto original (lo clona). Se suele usar con prefabs.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Rigidbody projectile;
    void Update() {
        if (Input.GetButtonDown("Fire1")) {
            Rigidbody clone;
            clone = Instantiate(projectile, transform.position, transform.rotation)
                    as Rigidbody;
            clone.velocity = transform.TransformDirection(Vector3.forward * 10);
        }
    }
}
```

GameObject: funciones

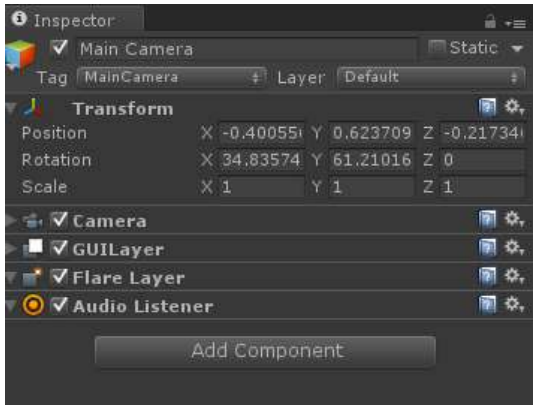
static void DontDestroyOnLoad(Object target);

Crea una nueva instancia del objeto original (lo clona). Se suele usar con prefabs.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Rigidbody projectile;
    void Update() {
        if (Input.GetButtonDown("Fire1")) {
            Rigidbody clone;
            clone = Instantiate(projectile, transform.position, transform.rotation)
                    as Rigidbody;
            clone.velocity = transform.TransformDirection(Vector3.forward * 10);
        }
    }
}
```

Componentes

- Son las piezas de funcionalidad que agregamos a los objetos.
- Heredan de MonoBehaviour.
- Se pueden comunicar entre si mediante varios mecanismos (mensajes, llamadas directas).
- Se pueden agregar o borrar de un objeto en tiempo de ejecución.



- Uso en el editor:
  - ¿Cómo agregarlos desde el editor?
  - Activar/desactivar desde el editor.
  - Exponer variables al inspector mediante public.
  - Asignar referencias a variables de componentes mediante el editor.

Componentes: funciones

MonoBehaviour.Awake()

Se llama a Awake cuando el script se instancia. Se usa para inicializar variables antes de que comience el juego. Awake siempre es llamado antes que cualquier función Start.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    private GameObject target;
    void Awake() {
        target = GameObject.FindWithTag("Player");
    }
}
```

Componentes: funciones

MonoBehaviour.Start()

Start es llamado cuando el script es habilitado justo antes de llamar al Update.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    private GameObject target;
    void Start() {
        target = GameObject.FindWithTag("Player");
    }
}
```

Componentes: funciones

MonoBehaviour.Update()

La función Update es llamada una vez por frame si el objeto está activo.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Update() {
        transform.Translate(0, 0, Time.deltaTime * 1);
    }
}
```

Componentes: funciones

MonoBehaviour.FixedUpdate()

La función FixedUpdate es llamada una vez por cada ciclo fijo de actualización si el objeto está activo. Se usa sobre todo para manipular objetos con física (con Rigidbody).

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void FixedUpdate() {
        rigidbody.AddForce(Vector3.up);
    }
}
```

Componentes: funciones

MonoBehaviour.LateUpdate()

La función LateUpdate es llamada una vez por frame si el objeto está activo. Se llama siempre después de Update y se usa por ejemplo para el código de moviemiento de cámaras.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void LateUpdate() {
        transform.Translate(0, 0, Time.deltaTime * 1);
    }
}
```

Componentes: funciones

MonoBehaviour.LateUpdate()

La función LateUpdate es llamada una vez por frame si el objeto está activo. Se llama siempre después de Update y se usa por ejemplo para el código de moviemiento de cámaras.

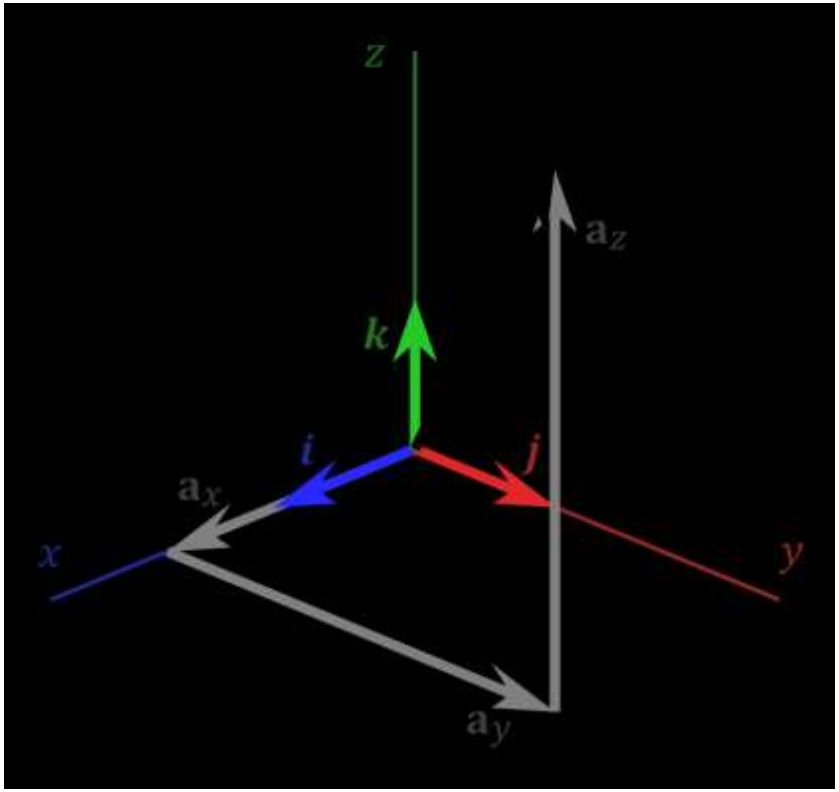
```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void LateUpdate() {
        transform.Translate(0, 0, Time.deltaTime * 1);
    }
}
```

Componentes: notas

- Algunas cosas a tener en cuenta:
  - Los scripts no pueden ir dentro de un namespace.
  - No meter nunca código de inicialización en el constructor, hacerlo en las funciones Awake o Start.
  - El valor de las variables public se guarda junto con la escena.

Vector3

- Estructura que representa un punto o vector en el espacio 3D.
- Tiene todas las operaciones que podamos necesitar para la manipulación de vectores:
  - Suma
  - Resta
  - Multiplicación
  - División



Vector3: variables

float x, y, z

Las componentes en flotante del vector para cada eje.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void LateUpdate() {
        Vector3 pos = new Vector3( 10, 10, 10 );
    }
}
```

Vector3: variables

float magnitude;

Longitud del vector.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Vector3 otherPosition;
    public float closeDistance = 5.0F;
    void Update() {
        if (other) {
            Vector3 offset = otherPosition - transform.position;
            float len = offset.magnitude;
            if (len < closeDistance )
                print("Estamos cerca!");

        }
    }
}
```

Vector3: variables

Vector3 normalized;

Devuelve el vector con magnitud 1.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Vector3 velocity;

    void Example() {
        Vector3 direction = velocity.normalized;
    }
}
```

Vector3: funciones

static float Angle(Vector3 from, Vector3 to);

Devuelve el ángulo en grados entre from y to.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Transform target;
    void Update() {
        Vector3 targetDir = target.position - transform.position;
        Vector3 forward = transform.forward;
        float angle = Vector3.Angle(targetDir, forward);
        if (angle < 5.0F)
            print("cerca");

    }
}
```

Vector3: funciones

static float Distance(Vector3 a, Vector3 b);

Devuelve la distancia entre a y b.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Transform other;
    void Example() {
        if (other) {
            float dist = Vector3.Distance(other.position, transform.position);
            print("Distancia: " + dist);
        }
    }
}
```

Vector3: funciones

static float Dot(Vector3 lhs, Vector3 rhs);

Devuelve el producto escalar de los dos vectores dados.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Transform enemy;
    void Update() {
        if (other) {
            Vector3 forward = transform.TransformDirection(Vector3.forward);
            Vector3 toOther = enemy - transform.position;
            if (Vector3.Dot(forward, enemy) < 0)
                print("El enemigo está detrás de mi.");

        }
    }
}
```

Vector3: funciones

static Vector3 Cross(Vector3 lhs, Vector3 rhs);

Devuelve el producto vectorial de los dos vectores dados.

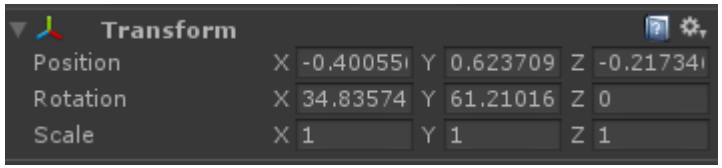
Vector3: funciones

static Vector3 Lerp(Vector3 from, Vector3 to, float t);

Interpola linealmente entre 2 vectores en un factor de 0 a 1.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Transform startMarker;
    public Transform endMarker;
    public float speed = 1.0F;
    private float startTime;
    private float journeyLength;
    public Transform target;
    public float smooth = 5.0F;
    void Start() {
        startTime = Time.time;
        journeyLength = Vector3.Distance(startMarker.position, endMarker.position);
    }
    void Update() {
        float distCovered = (Time.time - startTime) * speed;
        float fracJourney = distCovered / journeyLength;
        transform.position = Vector3.Lerp(startMarker.position, endMarker.position,
fracJourney);
    }
}
```

Transform



- Clase que representa la posición, rotación y escala de un objeto.
- Todos lo objetos de la escena tienen uno obligatoriamente.
- Los Transform pueden tener un padre, lo que permite formar jerarquías en las que un objeto hereda la posición, rotación y escala de su padre.
- Se puede iterar por los hijos de un objeto de la siguiente manera:

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Example() {
        foreach (Transform child in transform) {
            // …
        }
    }
}
```

Transform: variables

Vector3 position;

Posición del Transform en el mundo.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Example() {
        transform.position = new Vector3(0, 0, 0);
        transform.position += new Vector3(0, 1, 0 );
    }
}
```

Transform: variables

Vector3 forward;

Vector3 right;

Vector3 up;

Direcciones de los ejes del objeto en coordenadas de mundo.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Example() {
        rigidbody.velocity = transform.forward * 10;
    }
}
```

Transform: variables

Transform parent;

Objeto padre de este Transform.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Transform cameraTransform = Camera.main.transform;
    void Example() {
        cameraTransform.parent = transform;
        cameraTransform.localPosition = -Vector3.forward * 5;
        cameraTransform.LookAt(transform);
    }
}
```

Transform: funciones

void LookAt(Transform target);

Rota el Transform de tal manera que su vector forward apunte directamente a target.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public Transform target;
    void Update() {
        transform.LookAt(target);
    }
}
```

Transform: funciones

void **Rotate**(Vector3 **eulerAngles**);

Rota el Transform la cantidad de grados euler alrededor de cada eje.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Update() {
        transform.Rotate(Vector3.up * Time.deltaTime);
    }
}
```

Time

- Clase que representa el tiempo en Unity

Time: variables

static float deltaTime;

El tiempo que ha tardado en completarse el último frame. Se usa para hace que la velocidad el juego sea independiente del frame rate.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Update() {
        float translation = Time.deltaTime * 10;
        transform.position += new Vector3(0, 0, translation);
    }
}
```

Time: variables

static float timeScale;

La escala a la que el tiempo de Unity está pasando. Se puede usar para ralentizar o acelerar el paso del tiempo. Es el mismo tiempo que podemos configurar en

Edit->Project Settings->Time->Time Scale

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Update() {
        if (Input.GetButtonDown("Fire1")) {
            if (Time.timeScale == 1.0F)
                Time.timeScale = 0.7F;
            else
                Time.timeScale = 1.0F;
            Time.fixedDeltaTime = 0.02F * Time.timeScale;
        }
    }
}
```

Time: variables

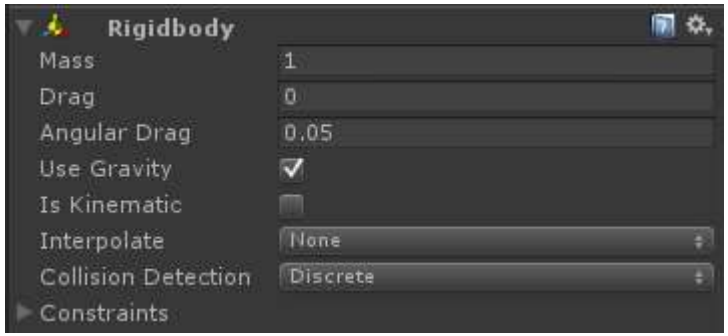
static float timeScale;

La escala a la que el tiempo de Unity está pasando. Se puede usar para ralentizar o acelerar el paso del tiempo. Es el mismo tiempo que podemos configurar en

Edit->Project Settings->Time->Time Scale

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Update() {
        if (Input.GetButtonDown("Fire1")) {
            if (Time.timeScale == 1.0F)
                Time.timeScale = 0.7F;
            else
                Time.timeScale = 1.0F;
            Time.fixedDeltaTime = 0.02F * Time.timeScale;
        }
    }
}
```

Rigidbody



- Añade al objeto la capacidad de ser controlado por el motor físico de Unity.
- Tiene en cuenta la gravedad y fuerzas aplicadas, además de poder responder de forma realista a colisiones.
- Los cambios a un rigidbody hay que hacerlos en FixedUpdate().
- Usa el motor PhysX
- Si usamos un rigidbody no debemos aplicar cambios a su posición modificando el transform directamente (a menos que sea kinematic)

Rigidbody: variables

float mass;

La masa del objeto. Para que la simulación funcione bien este valor ha de estar siempre entre 0.1 y 10. Por lo tanto hay que tener en cuenta cuales van a ser nuestros objetos más pesados y los más ligeros para que estén dentro de estos valores.

Rigidbody: variables

float drag;

Representa la resistencia al aire. Cuanto mayor sea este valor más se frenará el obejeto.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void OpenParachute() {
        rigidbody.drag = 20;
    }
    void Update() {
        if (Input.GetButton("Space"))
            OpenParachute();

    }
}
```

Rigidbody: variables

float angularDrag;

Representa la resistencia al aire pero solo para las rotaciones. . Cuanto mayor sea este valor más se frenará la rotación del objeto.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Example() {
        rigidbody.angularDrag = 10;
    }
}
```

Rigidbody: variables

bool useGravity;

Controla si el objeto es afectado por la gravedad. La fuerza de la gravedad se puede modificar en la clase Physics.gravity

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Example() {
        Physics.gravity = new Vector3(0, -1.0F, 0);
    }
}
```

Rigidbody: funciones

void AddForce(Vector3 force);

Aplica una fuerza con la dirección y magnitud dadas al Rigidbody.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void FixedUpdate() {
        rigidbody.AddForce(Vector3.up * 10);
    }
}
```

Rigidbody: funciones

void AddExplosionForce(float explosionForce, Vector3 explosionPosition, float explosionRadius);

Simula la fuerza de una explosión en el objeto.

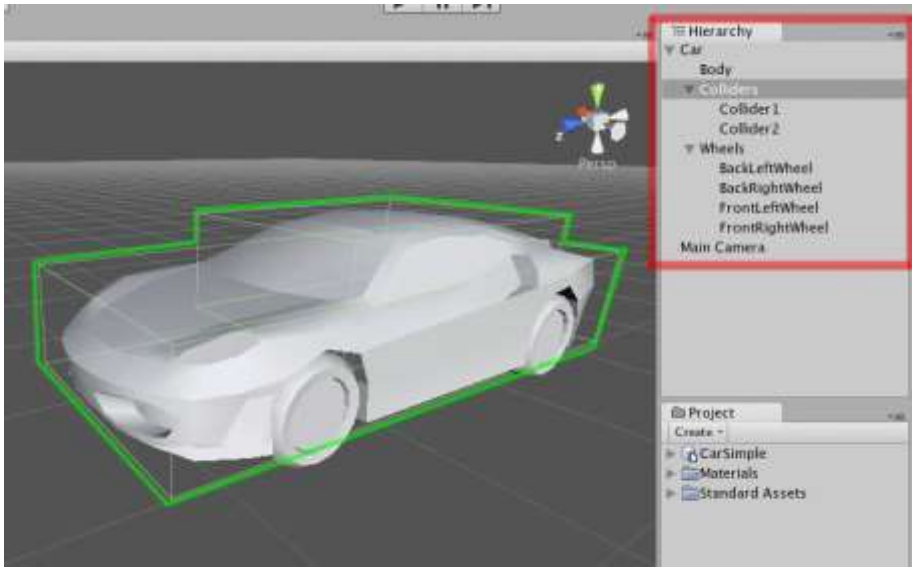
```
using UnityEngine;
using System.Collections;

public class Explosion : MonoBehaviour {
    public Transform bomb;
    void FixedUpdate()
    {
        if( Input.GetKeyDown( KeyCode.Space ) ){
            rigidbody.AddExplosionForce( 2000, bomb.position, 20 );
        }
    }
}
```

Collider

- Añade al objeto la capacidad de colisionar con otros Colliders
- Si el objeto se va a mover necesita tener un Rigidbody asociado
- Se les puede asociar un material físico para darle diferentes características.

Collider

- Como hemos dicho en Unity se crean objetos con propiedades físicas agregando el componente **Rigidbody**.
- Si además queremos que colisione con otros objetos tenemos que añadirle un componente **Collider**. Habitualmente la forma de este Collider es sólo una aproximación de la forma real del objeto. Esto se hace así por temas de rendimiento.



Collider: variables

bool isTrigger;

Si el objeto es un Trigger no tendrá interacción con otros Rigidbody. Lo usaremos para zonas de detección.

Collider: mensajes

Collider.OnCollisionEnter(Collision)

Collider.OnCollisionExit(Collision)

Collider.OnCollisionStay(Collision)

Collider.OnTriggerEnter(Collider)

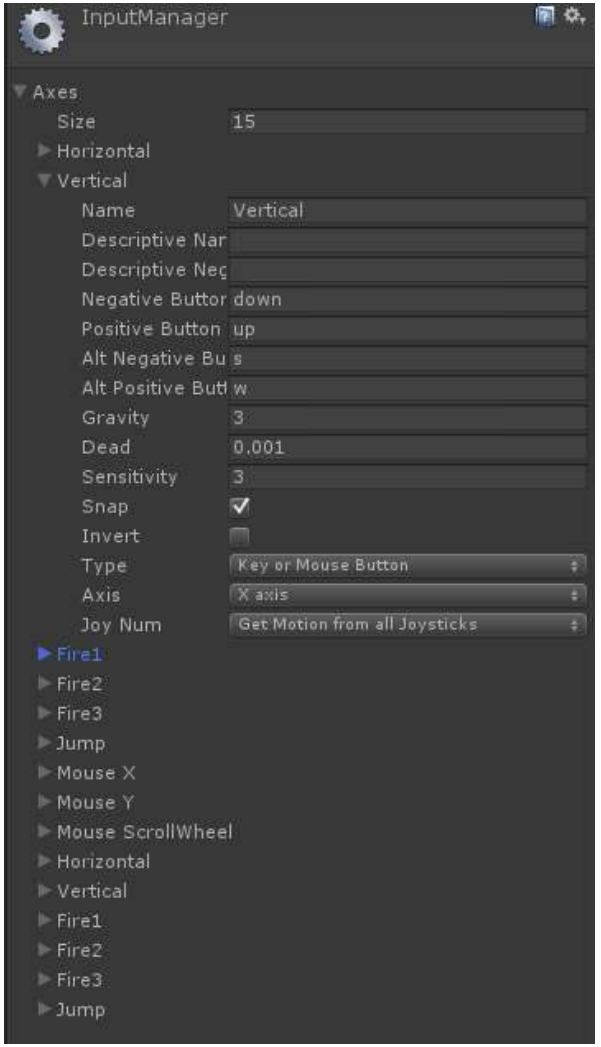
Collider.OnTriggerExit(Collider)

Collider.OnTriggerStay(Collider)

Cuando existe algún evento de colisión para este Collider se manda uno de estos mensajes a todos los componentes del GameObject.

Como parámetro recibimos una estructura con información de la colisión o el Collider en si dependiendo de si el es un Trigger o no.

Input



- Clase para acceder a los dispositivos de entrada, ya sean teclados, joysticks, ratones o datos dispositivos “móviles”.
- Se pueden definir teclas y ejes virtuales

Input: variables

static Vector3 mousePosition;

Posición del ratón en coordenadas de pixel.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public GameObject particle;
    void Update() {
        if (Input.GetButtonDown("Fire1")) {
            Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
            if (Physics.Raycast(ray))
                Instantiate(particle, transform.position, transform.rotation) as
GameObject;

        }
    }
}
```

Input: variables

static Touch[] touches;

Devuelve una lista de objetos que representa el estado de los toques en el último frame.

La estructura **Touch** contiene los siguientes campos:

- **deltaPosition**: delta de moviemiento
- **deltaTime**: tiempo transcuirrido desde la útima lectura
- **fingerId**: índide que representa ese toque
- **phase**: fase actual del toque (Began, Moved,Stationary,Ended,Canceled)
- **position**: posición en píxeles
- **tapCount**: número de taps (para detectar doble clic)

```
public class Example : MonoBehaviour {
    void Update() {
        int fingerCount = 0;
        foreach (Touch touch in Input.touches) {
            if (touch.phase != TouchPhase.Ended && touch.phase != TouchPhase.Canceled)
                fingerCount++;

        }
        if (fingerCount > 0)
            print("El usuario tiene " + fingerCount + " dedos en la pantalla");
    }
}
```

Input: funciones

static float GetAxis(string axisName);

Devuelve el valor del eje virtual con el nombre axisName. El valor está entre -1 y 1.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public float speed = 10.0F;
    public float rotationSpeed = 100.0F;
    void Update() {
        float translation = Input.GetAxis("Vertical") * speed;
        float rotation = Input.GetAxis("Horizontal") * rotationSpeed;
        translation *= Time.deltaTime;
        rotation *= Time.deltaTime;
        transform.Translate(0, 0, translation);
        transform.Rotate(0, rotation, 0);
    }
}
```

Input: funciones

static bool **GetButton**(string **buttonName**);

Devuelve el estado de pulsación del botón.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public GameObject projectile;
    public float fireRate = 0.5F;
    private float nextFire = 0.0F;
    void Update() {
        if (Input.GetButton("Fire1") && Time.time > nextFire) {
            nextFire = Time.time + fireRate;
            GameObject clone = Instantiate(projectile, transform.position,
transform.rotation) as GameObject as GameObject;
        }
    }
}
```

Input: funciones

static bool **GetButton**(string **buttonName**);

static bool **GetButtonDown**(string **buttonName**);

static bool **GetButtonDown**(string **buttonName**);

Devuelve el estado de pulsación del botón.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public GameObject projectile;
    void Update() {
        if (Input.GetButtonDown("Fire1"))
            Instantiate(projectile, transform.position, transform.rotation) as GameObject;

    }
}
```

Input: funciones

static bool GetKey(KeyCode key);

static bool GetKeyUp(KeyCode key);

static bool GetKeyDown(KeyCode key);

Devuelve el estado de pulsación de la tecla.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Update() {
        if (Input.GetKey("up"))
            print("up arrow key is held down");

        if (Input.GetKey("down"))
            print("down arrow key is held down");

    }
}
```

Input: funciones

static bool **GetMouseButton**(int **button**);

static bool **GetMouseButtonUp**(int **button**);

static bool **GetMouseButtonDown**(int **button**);

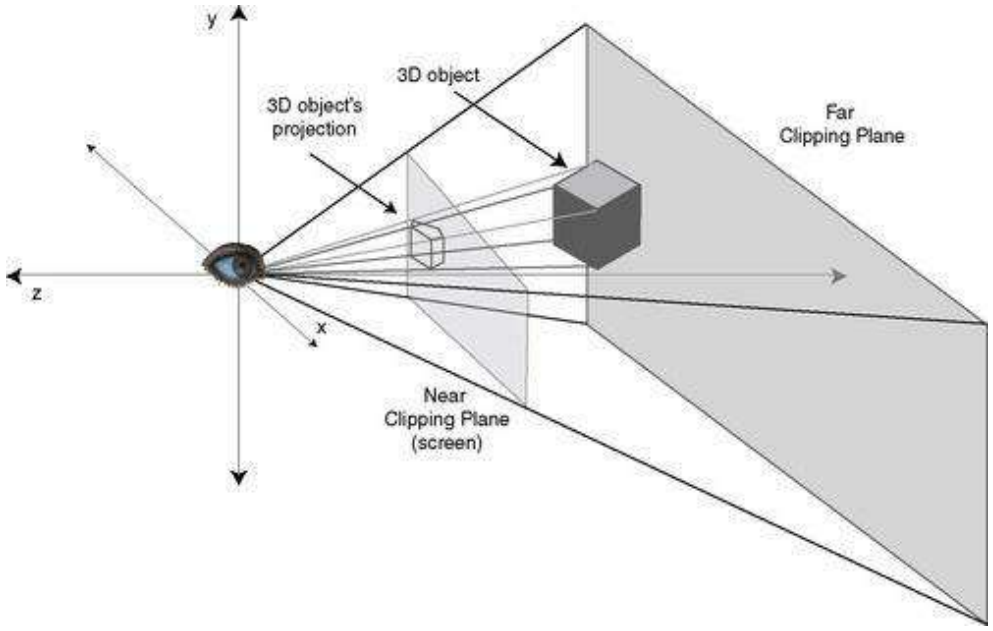
Devuelve el estado de pulsación de un botón del ratón.

```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    void Update() {
        if (Input.GetMouseButton(0))
            Debug.Log(“Izquierdo pulsado.");

        if (Input.GetMouseButton(1))
            Debug.Log(“Derecho pulsado.");

        if (Input.GetMouseButton(2))
            Debug.Log(“Central pulsado.");

    }
}
```

Camera

Representa las cámaras en Unity

- La cámara en un juego define el área visible del mundo por el jugador, tanto en altura, anchura y profundidad.
- Si un objeto no está dentro de esta área no se dibuja por el motor.
- Pueden ser de 2 tipos:
  - Ortográfica: se usa para juegos en 2 dimensiones
  - Perspectiva: son las cámaras que se usan en 3 dimensiones



Camera

- El "**far plane**" es el plano más lejano a partir del cual no se renderizan objetos
- El "**near plane**" es el plano que representa la pantalla y sobre el que se proyectan los objetos.
- El **FOV** o **field of view** es el ángulo de visión de la cámara.

Variables:

- **Clear flags**: indica como se borrará la pantalla.
- **Background**: color de fondo.
- **Culling Mask**: filtro que indica que layers se van a pintar en esta cámara.
- **Projection**: tipo de proyección.
- **Field of View**: ángulo en grados del ancho de la cámara.
- **Clipping Planes**: distancias a la cámara de los planos cercanos y lejanos.

GUI

- Los controles de la GUI de Unity van siempre en la función OnGUI().
- No se instancian como otros objectos, se pintan frame a frame.
- El estado del control se devuelve en la propia llamada
- Pueden usar skins

```
void OnGUI ()
{
  GUI.Box(new Rect(10,10,100,90), "Loader Menu");

  if(GUI.Button(new Rect(20,40,80,20), "Level 1"))
  {
    Application.LoadLevel(1);
  }

  if(GUI.Button(new Rect(20,70,80,20), "Level 2"))
  {
    Application.LoadLevel(2);
  }
}
```

GUI

- Anatomía de un control: todos los controles comparten varios datos que que se le pasará al crearlos. Son los siguientes:
  - Tipo: Label, Button, Box…
  - Posición: la posición donde se pintará (en pixels)
  - Contenido: dependiendo del control puede ser texto, textura, etc…

```
GUI.Button(new Rect(0, 0, 100, 20), "Click Me");
GUI.Button(new Rect(0, 30, 100, 20), new GUIContent("Click Me"));
GUI.Button(new Rect(0, 0, 100, 20), new GUIContent("Click me", icon));
```