Unity 2019
Controladores

1 **Desplazamiento y giro de objetos**

2 **Control simple de personajes**

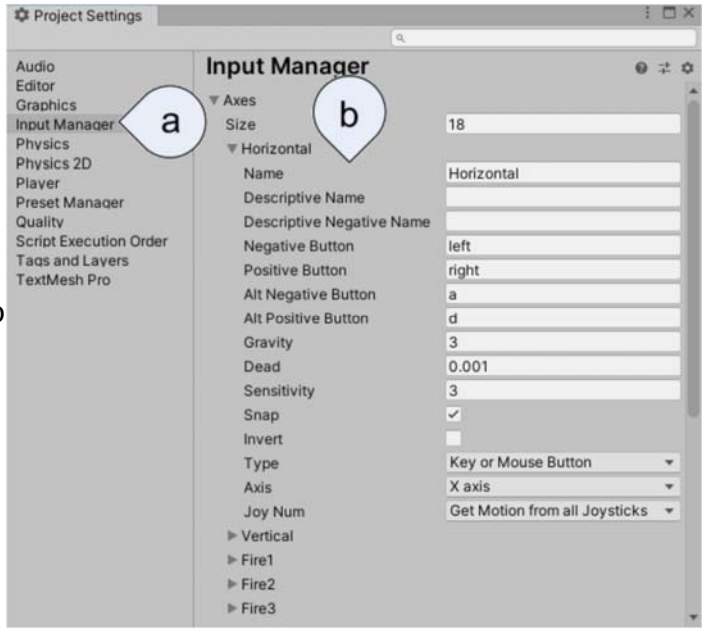
3 **Lanzar objetos**

4 **Destruir objetos**

1 Desplazamiento y giro de objetos

Input Manager

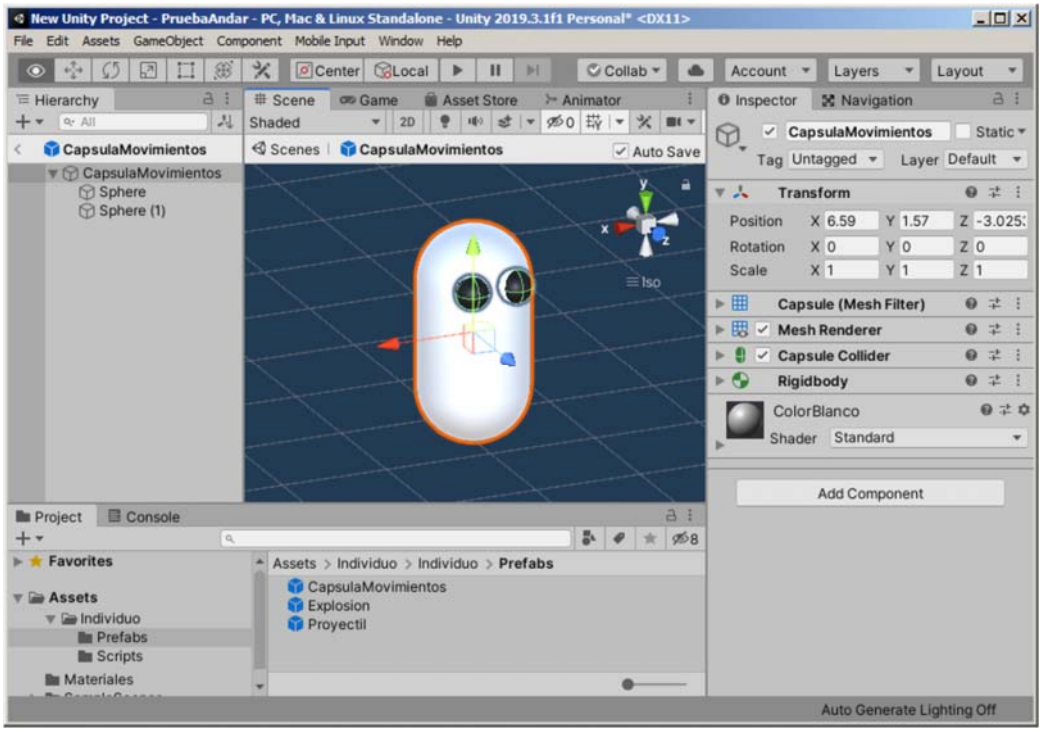
- Dentro de *Project Settings* (accesible desde el menú *Edit*) , tenemos la opción de gestionar los *Inputs* de nuestra aplicación.
- Así, desde nuestra aplicación comprobaremos si se ha accionado algún control que indique un desplazamiento en vez de tener que comprobar si se ha pulsado una tecla, un botón del joystick o del guante de VR…
  a) Dentro de Input Manager es posible controlar distintas entradas:
  b) Horizontal: entradas que indican desplazamiento horizontal, tanto en sentido positivo (de 0 a 1) como en sentido negativo (de 0 a -1).
     – Flecha dcha / izda
     – Letras d / a
     – **Snap:** si está activado, no disminuye hasta cero si hay cambio de dirección (cambio brusco)



1 Desplazamiento y giro de objetos

Definimos un prefab para pruebas

- Dibujamos una cápsula con un par de *ojos*
- Le añadimos un *Collider* y un *RigidBody* y la definimos como *prefab*



1 Desplazamiento y giro de objetos

Modificar el *transform.position* del objeto

- La forma más sencilla de desplazar objetos consiste en modificar directamente el valor de *transfom.position* del objeto

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class xMovTransform : MonoBehaviour
6   {
7       public float lecturaDesp;  // Lectura del desplazamiento
8
9       public float velocidad = 1.5f;  //  Velocidad de desplazamiento en m/s
10      public Vector3 desplazamiento;  //  Vector donde guardar el desplazamiento
11
12      // Start is called before the first frame update
13      void Start() { }
14
15      // Update is called once per frame
16      void Update()
17      {
18          lecturaDesp = Input.GetAxis("Vertical");  //  Leemos si pulsada la flecha arriba/abajo ó w/s ...
19
20          desplazamiento = Vector3.forward * lecturaDesp * velocidad * Time.deltaTime;  //  Calculamos el desplazamiento
21          transform.position = transform.position + desplazamiento;                     //  Cambiamos el valor de transform
22
23
24          lecturaDesp = Input.GetAxis("Horizontal");  //  Leemos si pulsada la flecha izda/dcha ó a/d ...
25
26          desplazamiento = Vector3.right * lecturaDesp * velocidad * Time.deltaTime;    //  Calculamos el desplazamiento
27          transform.position = transform.position + desplazamiento;                     //  Cambiamos el valor de transform
28
29      }
30  }
```

1 Desplazamiento y giro de objetos

Desplazamiento mediante *transform.Translate* del objeto

- Utilizando este método permite avanzar al objeto aunque esté girado

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

public class xMovTransformMetodos : MonoBehaviour
{
    public float lecturaDesp;  // Lectura del desplazamiento

    public float velocidad = 1.5f;  //  Velocidad de desplazamiento en m/s
    public Vector3 desplazamiento;  //  Vector donde guardar el desplazamiento

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

    // Update is called once per frame
    void Update()
    {
        lecturaDesp = Input.GetAxis("Vertical");  //  Leemos si pulsada la flecha arriba/abajo ó w/s ...

        desplazamiento = Vector3.forward * lecturaDesp * velocidad * Time.deltaTime;  //  Calculamos el desplazamiento
        transform.Translate(desplazamiento);  //  Llamamos al método, indicando el desplazamiento


        lecturaDesp = Input.GetAxis("Horizontal");  //  Leemos si pulsada la flecha izda/dcha ó a/d ...

        desplazamiento = Vector3.right * lecturaDesp * velocidad * Time.deltaTime;    //  Calculamos el desplazamiento
        transform.Translate(desplazamiento);  //  Llamamos al método, indicando el desplazamiento

    }
}
```


1 Desplazamiento y giro de objetos

Giro de objetos. Método *transform.Rotate*

- Independientemente del modo de desplazamiento utilizado, la forma de girar los objetos puede ser siempre la misma:

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

public class xMovTransformMetodos : MonoBehaviour
{
    public float lecturaDesp;  // Lectura del desplazamiento

    public float velocidad = 1.5f;  //  Velocidad de desplazamiento en m/s
    public Vector3 desplazamiento;  //  Vector donde guardar el desplazamiento

    public float giro = 90;   //  Velocidad de giro, en grados / s
    public Vector3 rotacion;  //  Vector donde guardar el giro

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

    // Update is called once per frame
    void Update()
    {
        lecturaDesp = Input.GetAxis("Vertical");  //  Leemos si pulsada la flecha arriba/abajo ó w/s ...

        desplazamiento = Vector3.forward * lecturaDesp * velocidad * Time.deltaTime;  //  Calculamos el desplazamiento
        transform.Translate(desplazamiento);  //  Llamamos al método, indicando el desplazamiento

        lecturaDesp = Input.GetAxis("Horizontal");  //  Leemos si pulsada la flecha izda/dcha ó a/d ...

        // desplazamiento = Vector3.right * lecturaDesp * velocidad * Time.deltaTime;    //  Calculamos el desplazamiento
        // transform.Translate(desplazamiento);  //  Llamamos al método, indicando el desplazamiento
        rotacion = Vector3.up * lecturaDesp * giro * Time.deltaTime;  //  Calculamos el giro
        transform.Rotate(rotacion);     //  Llamamos al método, indicando el giro
    }
}
```

1 Desplazamiento y giro de objetos

Desplazamiento del RigidBody

- El objeto se comporta mejor si actuamos sobre su RigidBody, cambiando el valor de *RigidBody.velocity*:

```
public class xMovRigidBody : MonoBehaviour
{
    public float lecturaDesp;  // Lectura del desplazamiento

    public float velocidad = 1.5f;  //  Velocidad de desplazamiento en m/s
    public Vector3 desplazamiento;  //  Vector donde guardar el desplazamiento

    public float giro = 90;   //  Velocidad de giro, en grados / s
    public Vector3 rotacion;  //  Vector donde guardar el giro

    public Rigidbody rigidBody;  //  Variable donde almacenar el rigidbody del objeto

    // Start is called before the first frame update
    void Start()
    {   rigidBody = gameObject.GetComponent<Rigidbody>();  //  Inicializamos el rigidBody
    }

    // Update is called once per frame
    void FixedUpdate()
    {   lecturaDesp = Input.GetAxis("Vertical");  //  Leemos si pulsada la flecha arriba/abajo ó w/s ...
        float velY = rigidBody.velocity.y;        //  Guardamos la velocidad de "caida"

        desplazamiento = transform.forward * lecturaDesp * velocidad;  //  Calculamos el desplazamiento
        rigidBody.velocity = desplazamiento;        //  Cambiamos el valor de velocity (en el plano x,z)
        rigidBody.velocity = new Vector3 (rigidBody.velocity.x, velY, rigidBody.velocity.z);  //  añadimos la velocidad en Y

        lecturaDesp = Input.GetAxis("Horizontal");  //  Leemos si pulsada la flecha izda/dcha ó a/d ...

        rotacion = Vector3.up * lecturaDesp * giro * Time.deltaTime;  //  Calculamos el giro
        transform.Rotate(rotacion);     //  Llamamos al método, indicando el giro
    }
}
```


1 Desplazamiento y giro de objetos

Desplazamiento del *CharacterController*

- El objeto también se puede controlar actuando sobre su *CharacterController*, el problema es que no incluye la gravedad:

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

public class xMovCharacterCon : MonoBehaviour
{
    public float lecturaDesp;  // Lectura del desplazamiento

    public float velocidad = 1.5f;  //  Velocidad de desplazamiento en m/s
    public Vector3 desplazamiento;  //  Vector donde guardar el desplazamiento

    public float giro = 90;   //  Velocidad de giro, en grados / s
    public Vector3 rotacion;  //  Vector donde guardar el giro

    public CharacterController characterController;   //  Variable donde almacenar el characterController del objeto

    // Start is called before the first frame update
    void Start()
    {   characterController = gameObject.GetComponent<CharacterController>();   //  Inicializamos el characterController
    }

    // Update is called once per frame
    void Update()
    {   lecturaDesp = Input.GetAxis("Vertical");  //  Leemos si pulsada la flecha arriba/abajo ó w/s ...

        desplazamiento = transform.forward * lecturaDesp * velocidad * Time.deltaTime;  //  Calculamos el desplazamiento
        characterController.Move(desplazamiento);   //  Llamamos al método, indicando el desplazamiento

        lecturaDesp = Input.GetAxis("Horizontal");  //  Leemos si pulsada la flecha izda/dcha ó a/d ...

        rotacion = Vector3.up * lecturaDesp * giro * Time.deltaTime;  //  Calculamos el giro
        transform.Rotate(rotacion);     //  Llamamos al método, indicando el giro
    }
}
```

2 Control simple de personajes

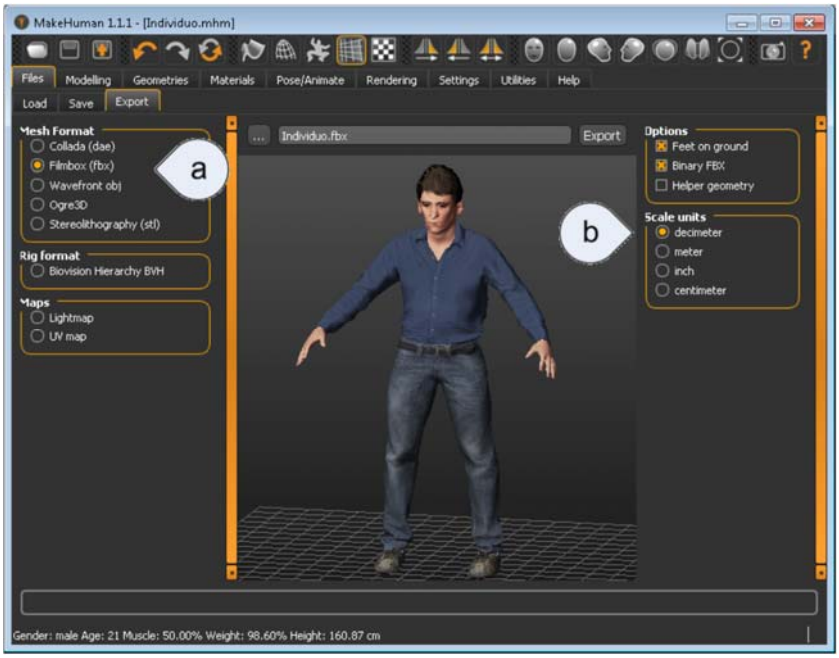
Personaje inicial

- Para hacer la práctica, partimos de un personaje creado en MakeHuman®

a) Lo exportamos en formato *filmbox* (.fbx)

b) Indicamos las unidades en decímetros
   - Se puede cambiar desde Unity

c) Comprobar que, junto al fichero .fbx, se encuentra la carpeta texturas
   - Si no es así, volver a exportar en otro formato



2 Control simple de personajes

Añadir personaje a la escena

- Una vez seleccionado el personaje en la escena:

a) Arrastrar el fichero .fbx y la carpeta de texturas al proyecto
   - Aparece un mensaje de error. Pulsar ***Fix Now***

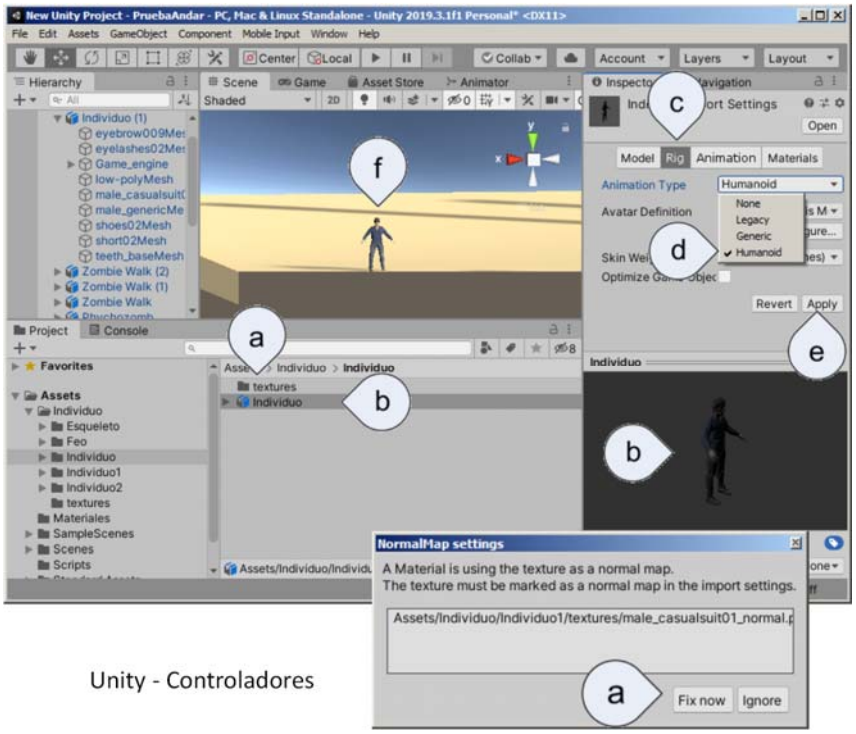
b) Pinchar sobre el personaje
   - Se puede ver la previsualización del personaje

c) Pinchar el botón Rig

d) Seleccionar la opción ***Humanoid*** del desplegable *Animation Type*

e) Aplicar cambios

f) Arrastrar el personaje a la escena



2 Control simple de personajes

Añadir controlador al personaje

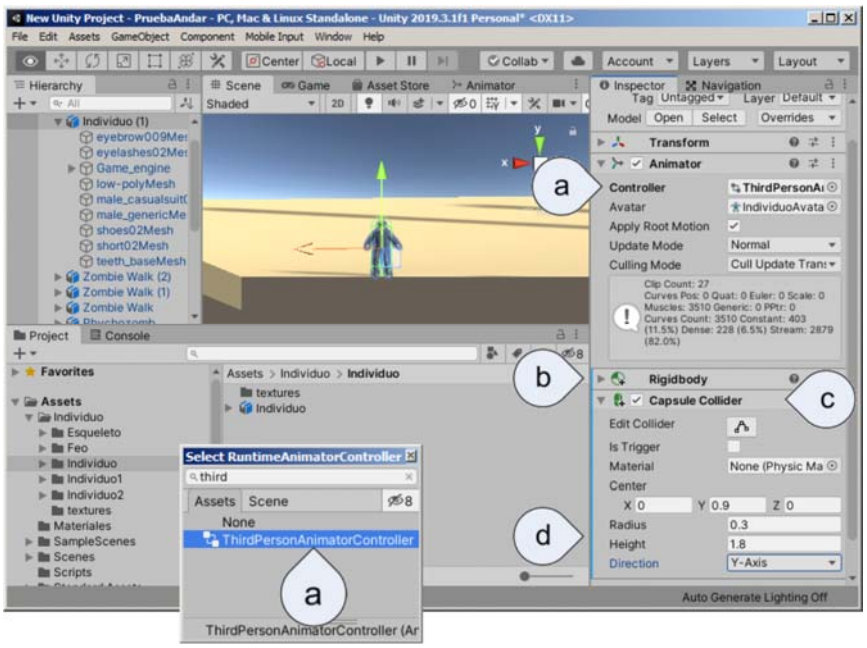
- Para añadir el controlador al personaje:

a) Seleccionar el controlador ***ThirdPersonAnimatorController*** en ***Controller***

b) Añadir el componente ***RigidBody***
   - No es necesario cambiar nada

c) Añadir el componente CapsuleCollider

d) Configurar el collider según las dimensiones del personaje (el del ejemplo mide 1,60m)
   - Center: Y = 0,8
   - Height: 1,8
   - Radius: 0,3
     (se ajusta a ojo)



2 Control simple de personajes

Añadir controlador al personaje

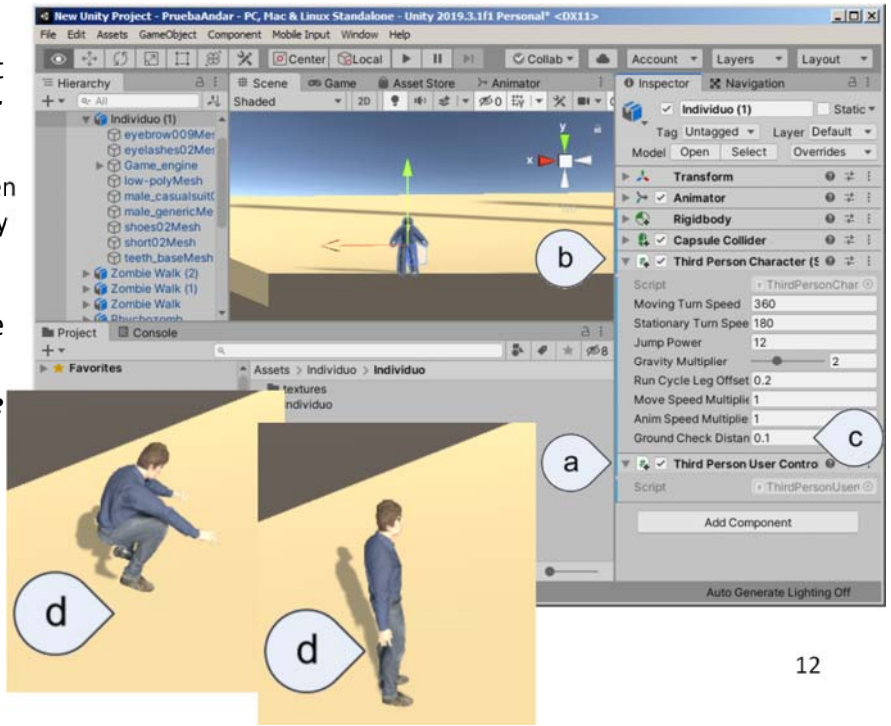
- Para terminar hay que añadir los Scripts:

a) Añadir el componente ***Third Person User Control***

b) Automáticamente se añade también el script ***Third Person Character***
   - Si no se han añadido anteriormente, también se añade el RigidBody y el collider

c) Generalmente, hay que modificar el valor de ***Ground Check Distance***

d) Ejecutar la aplicación y ver el mejor valor, ya que el personaje suele aparecer en el aire



3 Lanzar objetos

Definir el proyectil

- Vamos a utilizar como proyectil una esfera. Puede ser cualquier objeto

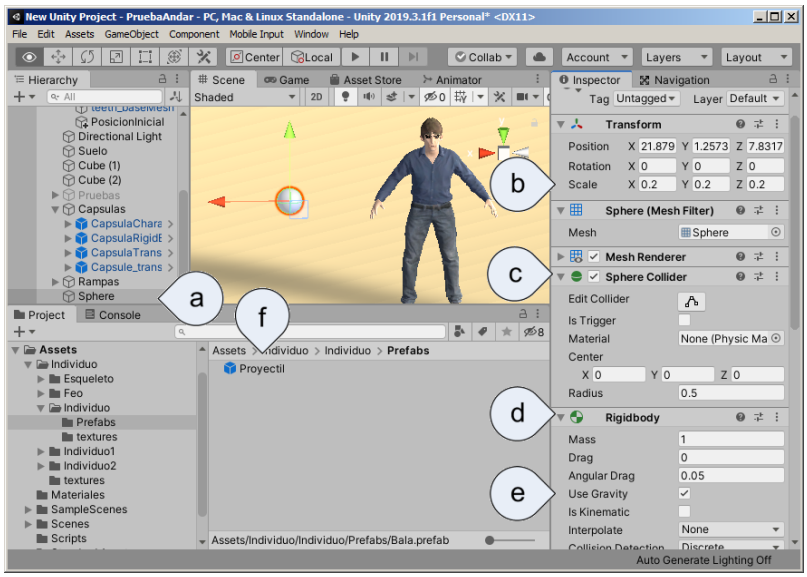
a) Crear una esfera

b) Definir las dimensiones
   - ***Scale*** = 0,2 (un palmo)

c) Comprobamos que tiene el ***Collider*** correcto

d) Añadimos un ***RigidBody***

e) Decidimos si aplicamos gravedad o no
   - Con gravedad es más realista. Las esferas acaban rodando por la escena



f) Definir la esfera como ***Prefab***. Le hemos dado el nombre de *Proyectil*.
   - Una vez definido como *Prefab*, se borra la esfera de *Hierarchy*

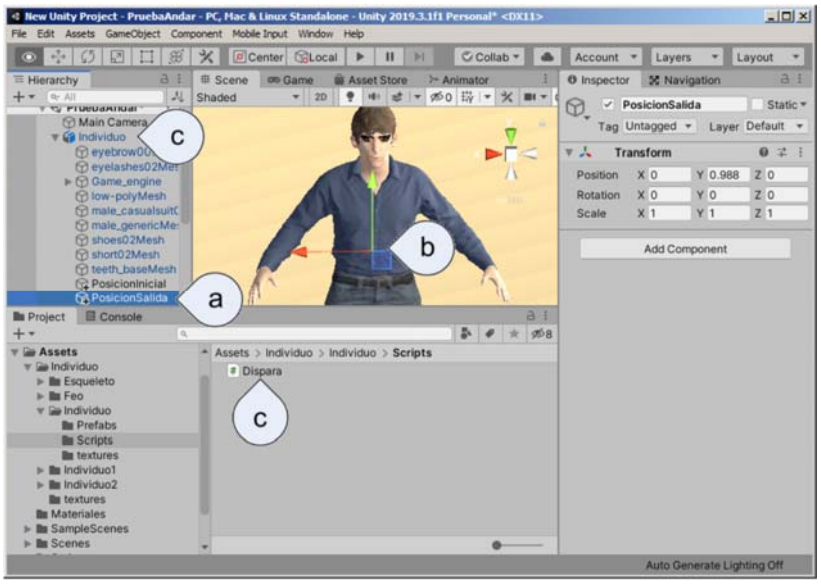
3 Lanzar objetos

Definir el punto de salida del objeto

- Es recomendable crear un GameObject que sirva de referencia para iniciar el disparo. Puede ser cualquiera del personaje, pero vamos a crear uno:

a) Creamos un objeto vacío (***Create Empty***) dentro del personaje
   - Le damos el nombre *PosicionSalida*

b) Situamos el punto de salida sobre el personaje
   - En el ejemplo lo hemos puesto cerca del ombligo para que no salga el disparo demasiado alto



c) Para el punto siguiente, creamos un script llamado disparo.cs y se lo añadimos al *GameObject* principal del personaje

3 Lanzar objetos

Script de disparo

- Para lanzar objetos, se crea uno nuevo y se desplaza como hemos visto en los temas anteriores:

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

public class Dispara : MonoBehaviour
{
    private float velocidad = 50f;  //  Velocidad a la que sale la esfera

    GameObject disparo;  //  GameObject que vamos creando por cada disparo

    public GameObject Bala;     //  Prefab con la bala que hemos creado
    public GameObject posSal;  //  GameObject situado en el personaje para indicar la posición de disparo

    bool apretado = false;  //  booleano para evitar los disparos repetidos

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

    // Como tratamos RigidBody, usamos FixedUpdate
    void FixedUpdate()
    {
        if (Input.GetKeyDown(KeyCode.LeftAlt) && !apretado)
        {
            disparo = Instantiate (Bala, posSal.transform.position, posSal.transform.rotation); //  Instanciamos bala en la posición inicial
            disparo.GetComponent<Rigidbody>().velocity = transform.forward * velocidad; //  Aplicamos la velocidad a la bala
            Destroy(disparo.gameObject, 3); //  Eliminamos la bala a los 3 segundos de haberla disparado

            apretado = true;    //  Marcamos tecla apredada
        }

        if (!Input.anyKeyDown) apretado = false;  //  Marcamos tecla suelta
    }
}
```

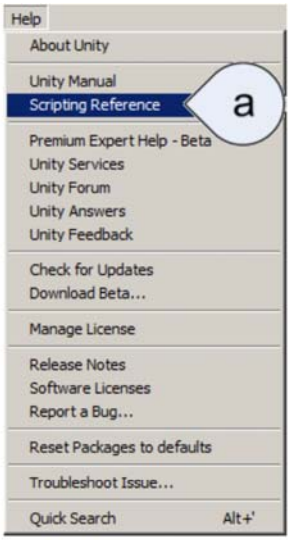
4 Destruir objetos

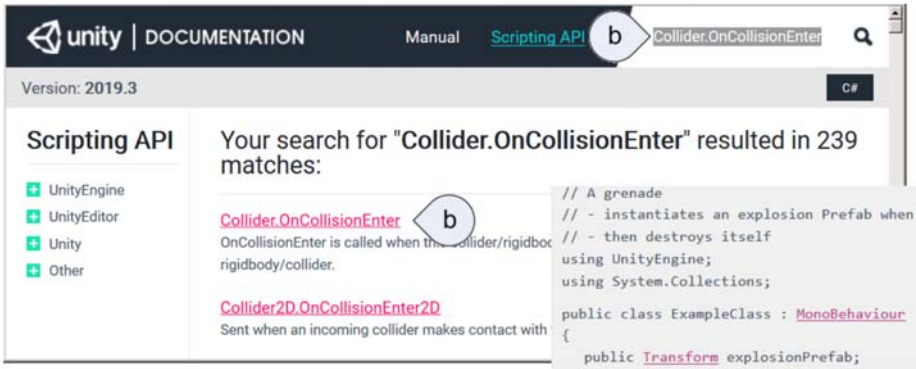
Descargar el script

- Vamos a utilizar un script ya definido en la biblioteca de scripts proporcionados por Unity

a) Dentro del menú Help, opción Scripting Reference.

b) Se accede a la página web de la documentación Unity. Buscamos el texto: Collider.OnCollisionEnter





c) Lo abrimos y copiamos la parte que nos interesa:

```
// A grenade
// - instantiates an explosion Prefab when hitting a surface
// - then destroys itself
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour
{
   public Transform explosionPrefab;

   void OnCollisionEnter(Collision collision)
   {
      ContactPoint contact = collision.contacts[0];
      Quaternion rotation = Quaternion.FromToRotation(Vector3.up, contact.normal);
      Vector3 position = contact.point;
      Instantiate(explosionPrefab, position, rotation);
      Destroy(gameObject);
   }
}
```

4 Destruir objetos

Script para destruir objetos

- Creamos un script llamado *Explota* y se lo añadimos al *Prefab* de las cápsulas con ojos (que van a ser nuestros objetivos)

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

public class Explota : MonoBehaviour
{
    public Transform explosionPrefab;  //  Prefab con las partículas de la explosión

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

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


    void OnCollisionEnter(Collision collision)
    {
        string etiqueta = collision.collider.tag;  //  Leemos la etiqueta del objeto con el que hemos colisionado

        if (etiqueta == "proyectil")  // Solo actúa si hemos colisionado con un proyectil
        {
            ContactPoint contact = collision.contacts[0];  //  Guardamos el primer punto de colisión guardado
            Quaternion rotation = Quaternion.FromToRotation(Vector3.up, contact.normal); // Calculamos ángulo donde colisión
            Vector3 position = contact.point;   //  Guardamos el punto de la colisión
            Instantiate(explosionPrefab, position, rotation);  //  instanciamos un gameObject de partículas en el punto de colisión
            Destroy(gameObject);  //  Destuimos el objeto en el que estamos
        }
    }
}
```


4 Destruir objetos

Crear sistema de partículas

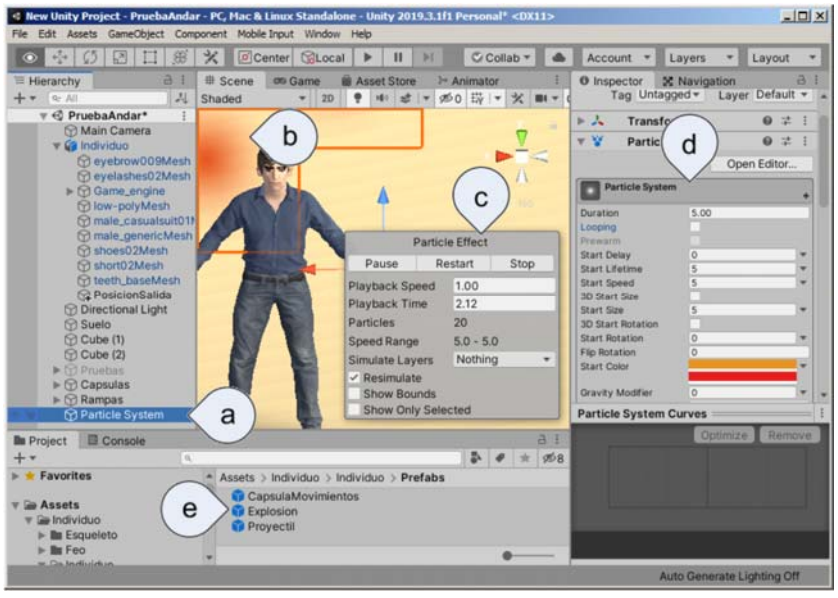
- Utilizamos un sistema de partículas para simular el efecto de la explosión

a) Creamos el sistema de partículas (Effects → Particle System).

b) El sistema se previsualiza en la escena

c) El recuadro permite controlar parámetros de la previsualización

d) Parámetros que cambiamos:
   - *Duration*: 5
   - *Looping*: desactiva
   - *Start Size*: 5
   - *Start Color*: Random



e) Una vez configurado el sistema se define como ***prefab*** y se borra de *Hierarchy.* El prefab se arrastrará a nuestro script

4 Destruir objetos

Asignar la etiqueta al proyectil

- Para hacer que funcione el script, solo falta asignar el ***Tag*** *proyectil* al ***prefab*** de la esfera