

Módulo 8

Código: 489

Programación multimedia y dispositivos móviles

Técnico Superior en Desarrollo de
Aplicaciones Multiplataforma



UNIDAD 5

Programación multimedia y juegos – Desarrollo de juegos 2D

Contenidos teóricos



1. Objetivos generales de la unidad
2. Competencias y contribución en la unidad
3. Contenidos conceptuales y procedimentales
4. Evaluación
5. Distribución de los contenidos
6. Sesiones

1. Objetivos generales de la unidad

Objetivos curriculares

1. Identificar los elementos principales de un juego.

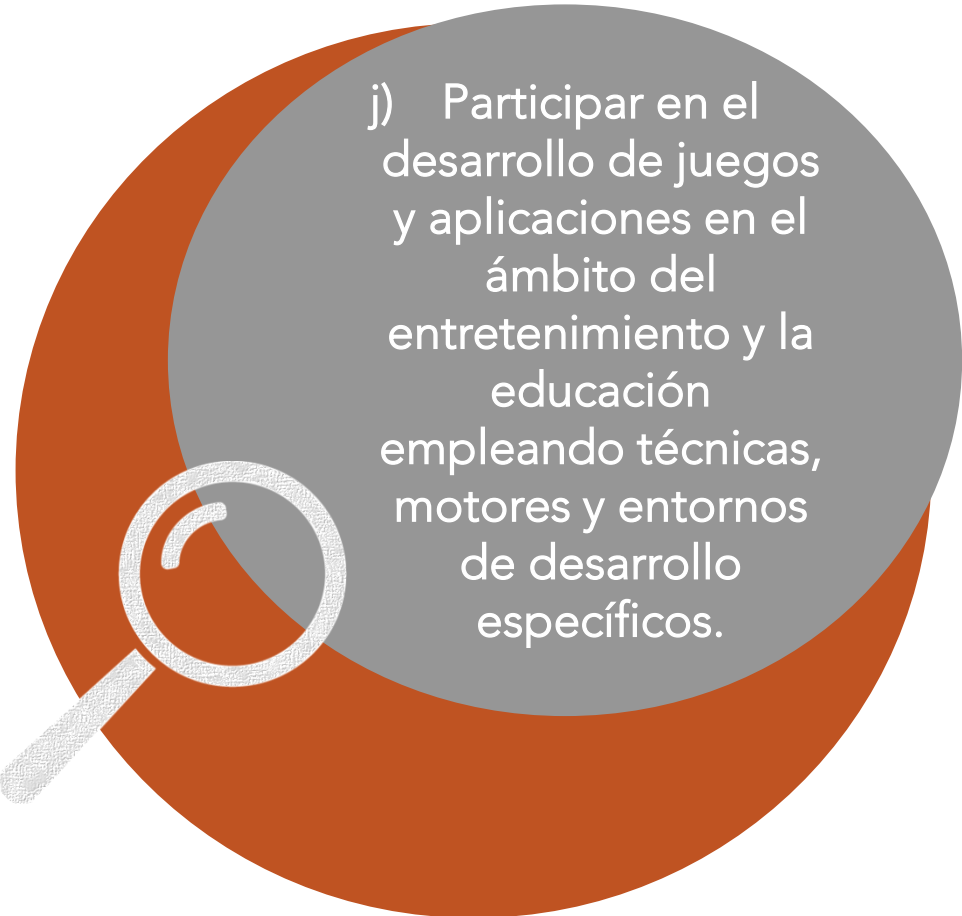
2. Comprobar los beneficios que aportan los motores de juegos.

3. Identificar los elementos principales de un juego.

4. Apreciar las características diferenciales del software para juegos

5. Construir desde cero las funcionalidades básicas de un videojuego.

2. Competencias y contribución en la unidad



j) Participar en el desarrollo de juegos y aplicaciones en el ámbito del entretenimiento y la educación empleando técnicas, motores y entornos de desarrollo específicos.

	Aprendiendo a desarrollar un videojuego en 2D, conociendo el entorno de desarrollo junto al motor seleccionado, y usándolo con este propósito.	
--	--	--

3. Contenidos

CONTENIDOS CONCEPTUALES

- Sprite
- Mod
- Scripting
- Render
- Tag
- Prefab

CONTENIDOS PROCEDIMENTALES

- Animación 2D.
- Arquitectura del juego. Componentes.
- Motores de juegos: Tipos y utilización.
- Áreas de especialización, librerías utilizadas y lenguajes de programación.
- Componentes de un motor de juegos.
- Librerías que proporcionan las funciones básicas de un Motor 2D/3D.
- Estudio de juegos existentes.
- Aplicación de modificaciones sobre juegos existentes.
- Entornos de desarrollo para juegos.
- Integración del motor de juegos en entornos de desarrollo.
- Aplicación de las funciones del motor gráfico.

4. Evaluación

a) Se han identificado los elementos que componen la arquitectura de un juego 2D

b) Se han analizado entornos de desarrollo de juegos

c) Se han analizado diferentes motores de juegos, sus características y funcionalidades

d) Se han identificado los bloques funcionales de un juego existente

e) Se ha establecido la lógica de un nuevo juego.

1. INTRODUCCIÓN

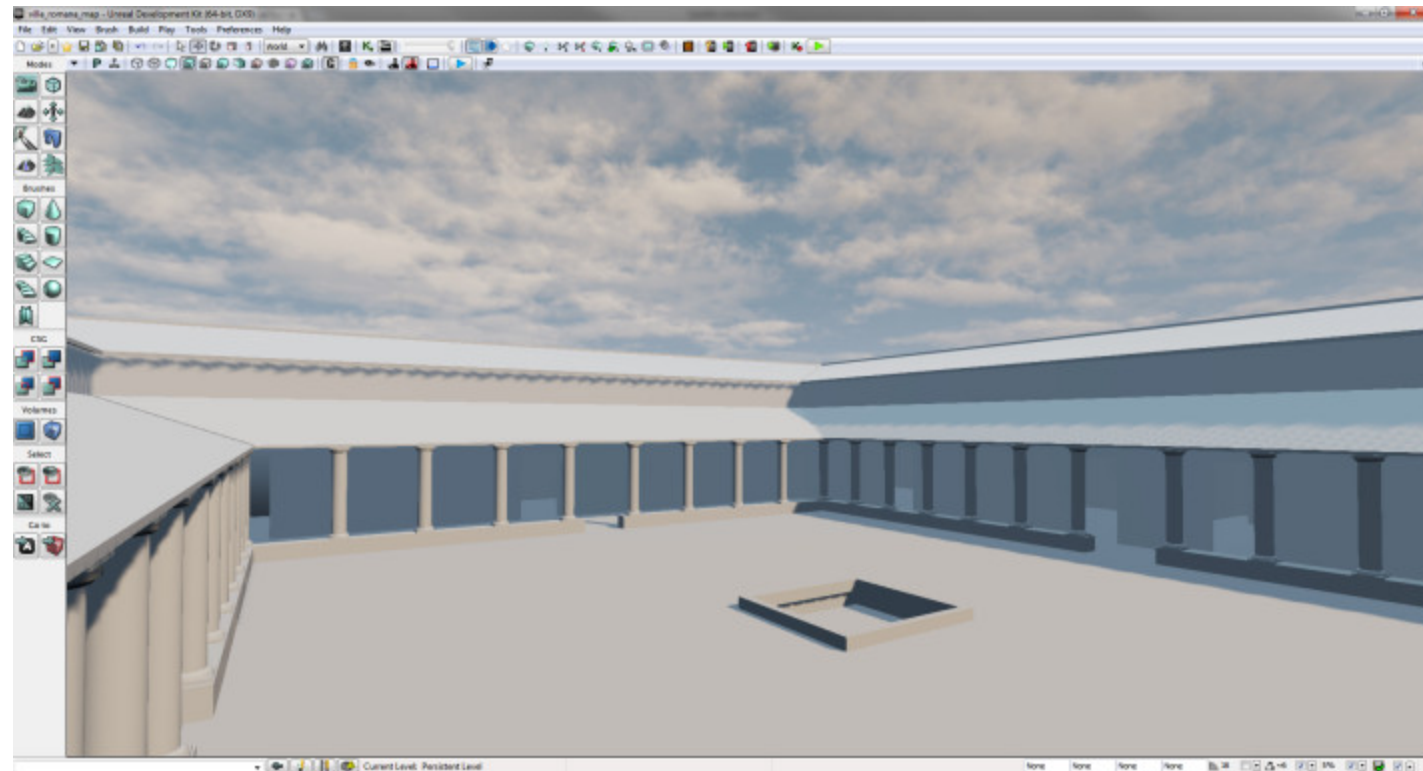
Igual que para el desarrollo de interfaces gráficas utilizábamos otras librerías, como Swing en Java, para el desarrollo de videojuegos utilizaremos motores de videojuegos.

Estos motores incluyen diferentes rutinas de programación que permiten la creación, diseño y funcionamiento de un videojuego. Entre las funcionalidades típicas, se incluye:

- Motor gráfico con capacidad para renderizar gráficos.
- Motor físico que simule las leyes de la física
- Animación de gráficos
- Posibilidad de creación de scripts para los elementos del juego
- Motor de sonido
- Inteligencia artificial
- Gestión de memoria
- Gestión de red
- [...]

1. INTRODUCCIÓN

Los motores de videojuegos suelen estar compuestos por un conjunto de herramientas de desarrollo visual y componentes software que pueden ser reutilizables. En la siguiente imagen, tenemos como ejemplo el entorno de desarrollo Unreal Engine 4, utilizado en juegos como Fortnite, Gears of War 4 y Unreal Tournament.



2. UNITY

Motor de videojuegos

- Sistema creado para hacer videojuegos de una forma rápida y sencilla.

Editor visual

- Permite cambios en tiempo real. Muy interactivo.

Arquitectura basada en componentes

- Permite ser modular y extensible.

Multiplataforma

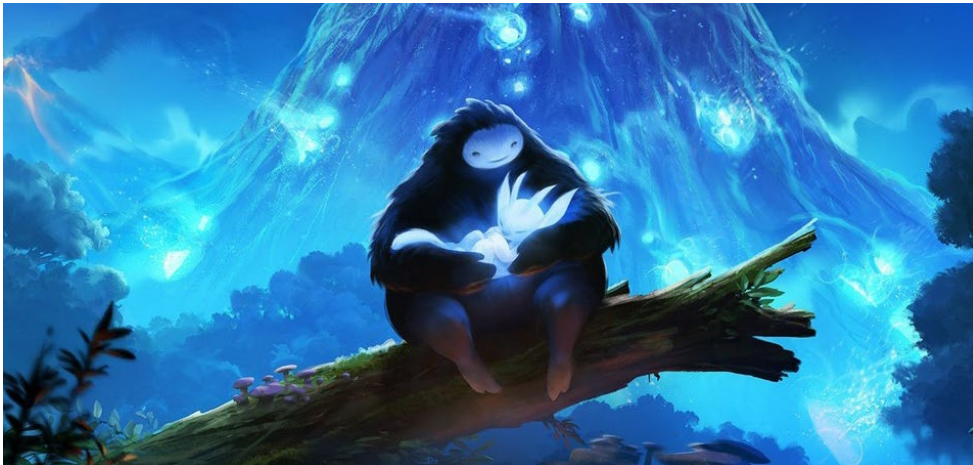
- Permite ser usado y desarrollar para múltiples plataformas



Sesión 1. Contenidos teóricos

2. UNITY

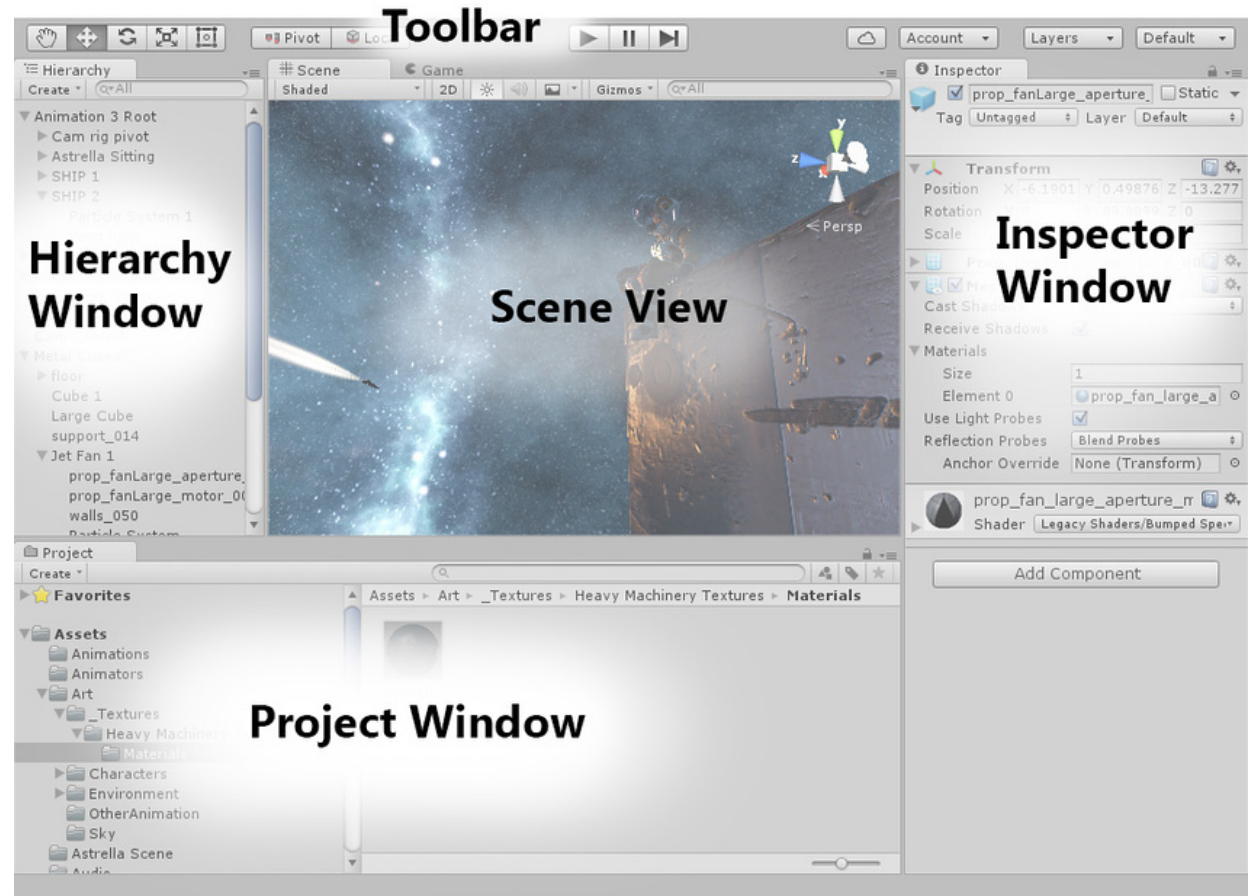
- Se utilizan Scripts para desarrollar el comportamiento de los distintos componentes.
- La lógica del juego se basa en la plataforma Open Source .NET: Mono.
- Lenguajes compatibles: C# y JavaScript.
- Fácil de usar.
- Visual scripting.
- Sistemas de eventos.



Sesión 1. Contenidos teóricos

3. INTERFAZ DE UNITY

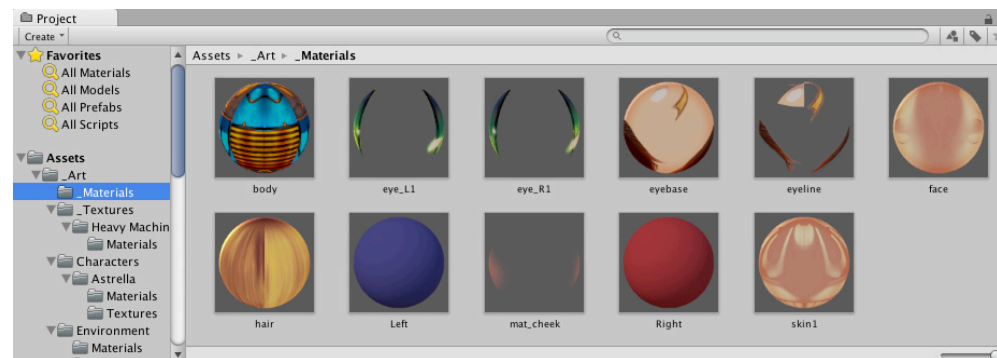
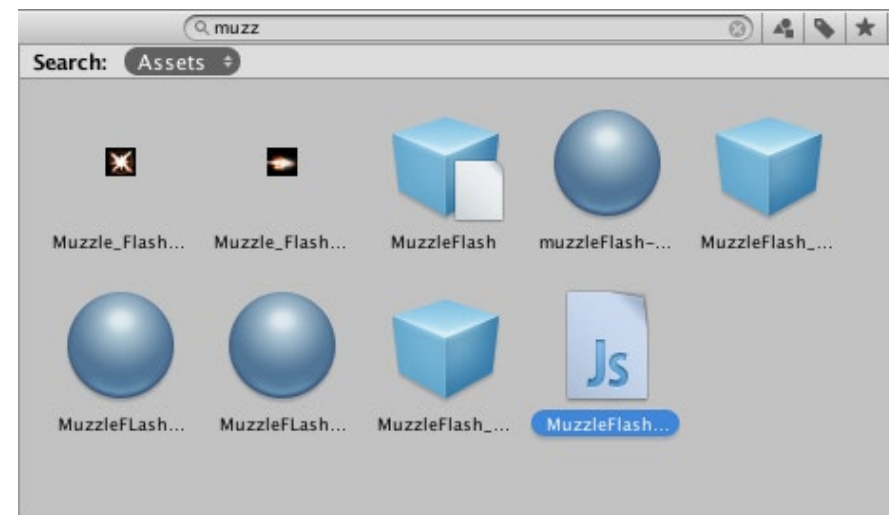
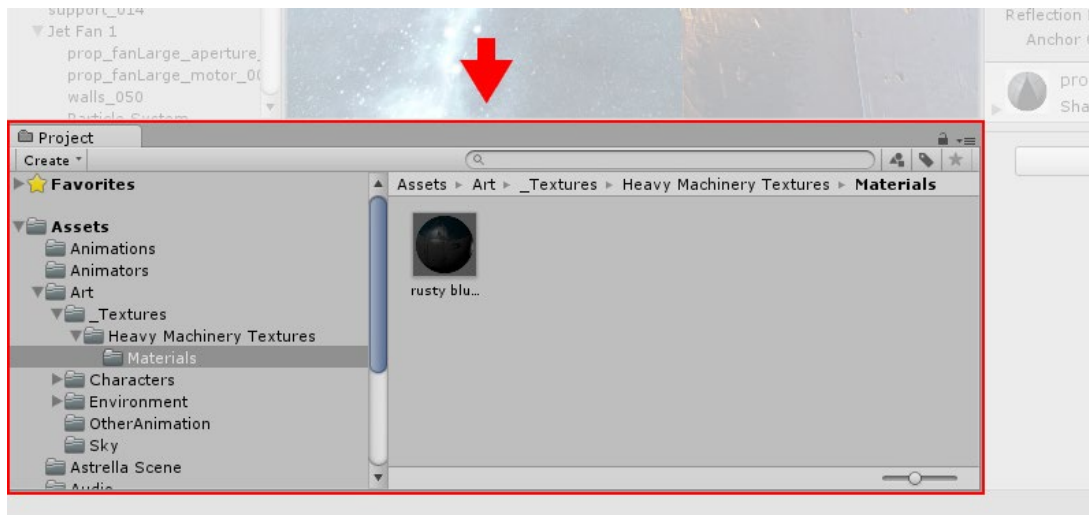
- [Ventana del Proyecto](#)
- [Visión de Escena](#)
- [Ventana de Jerarquía](#)
- [Ventana del Inspector](#)
- [Barra de herramientas](#)
- [Ventana del juego](#)



Sesión 1. Contenidos teóricos

3. INTERFAZ DE UNITY

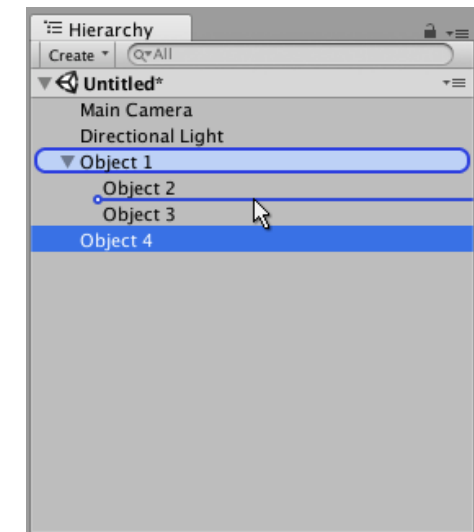
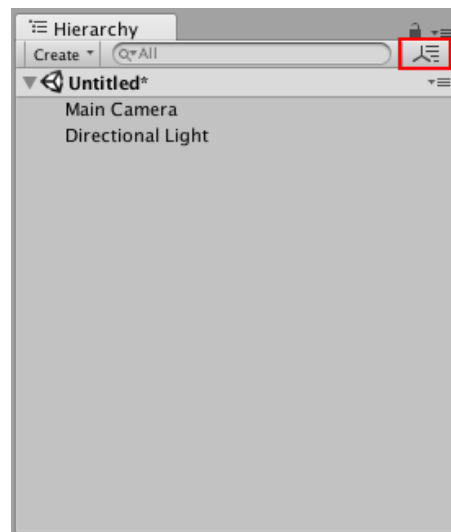
Ventana de Proyecto



Sesión 1. Contenidos teóricos

3. INTERFAZ DE UNITY

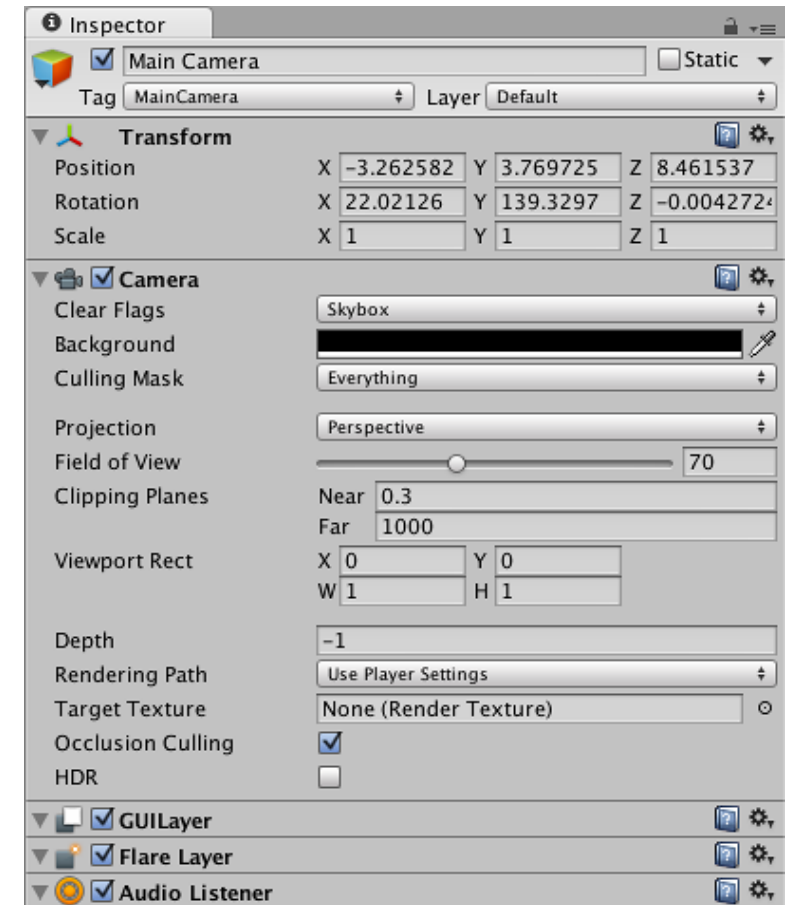
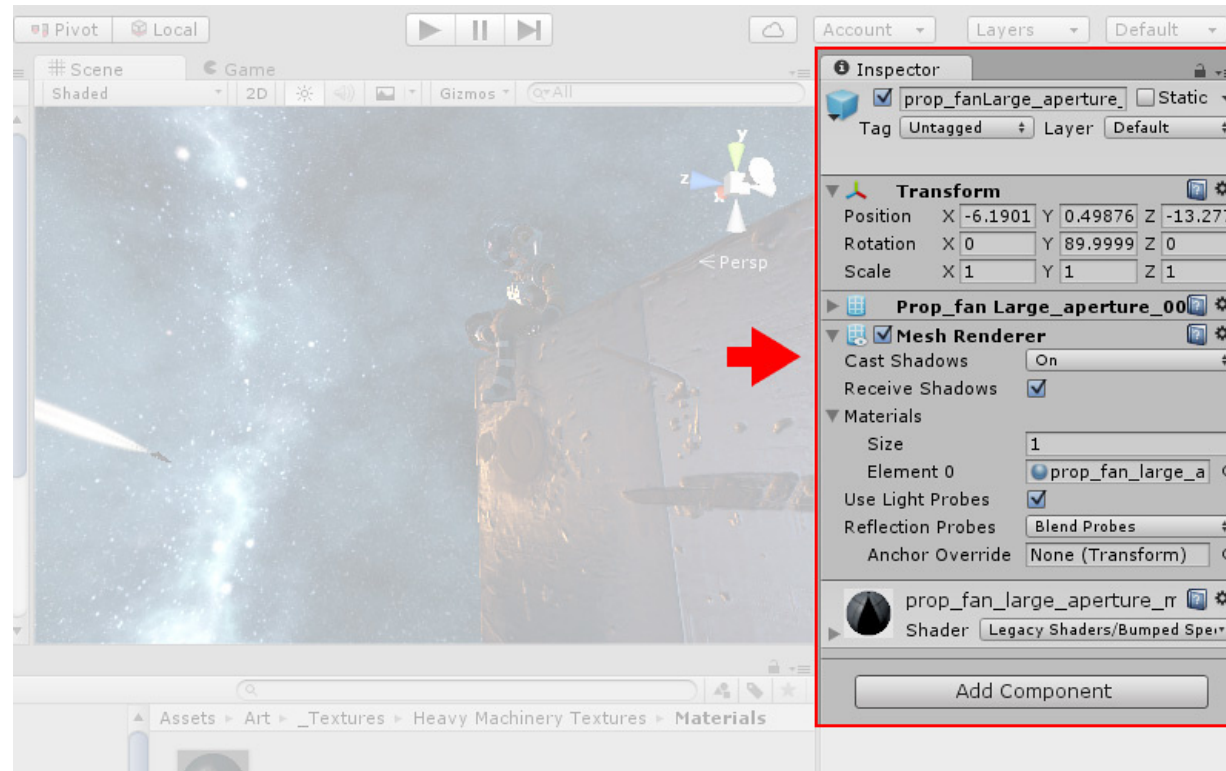
Ventana de Jerarquía



Sesión 1. Contenidos teóricos

3. INTERFAZ DE UNITY

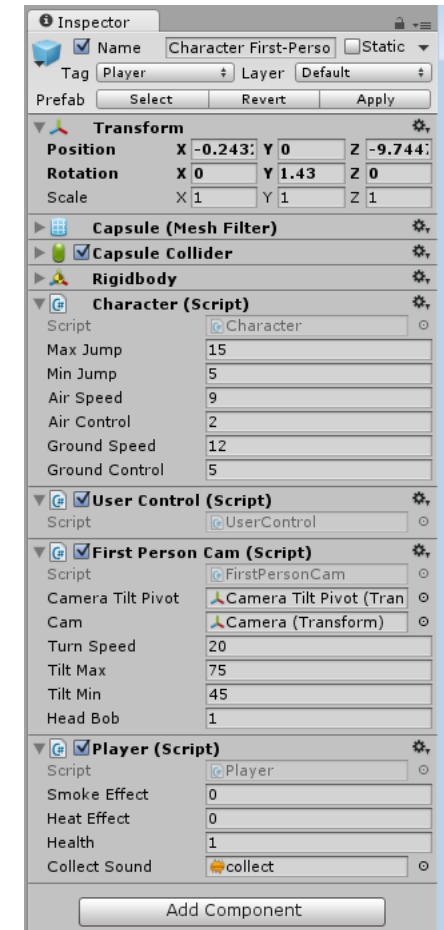
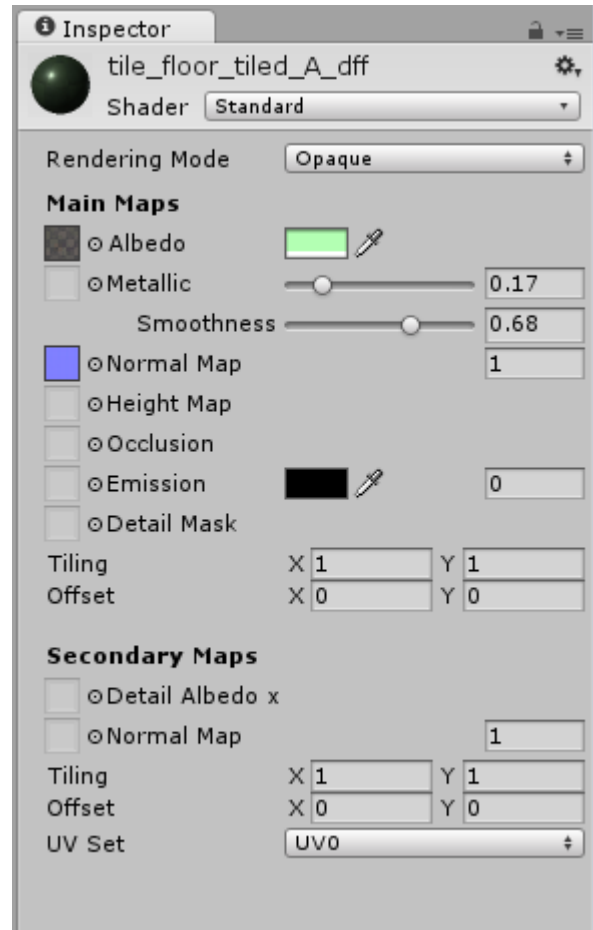
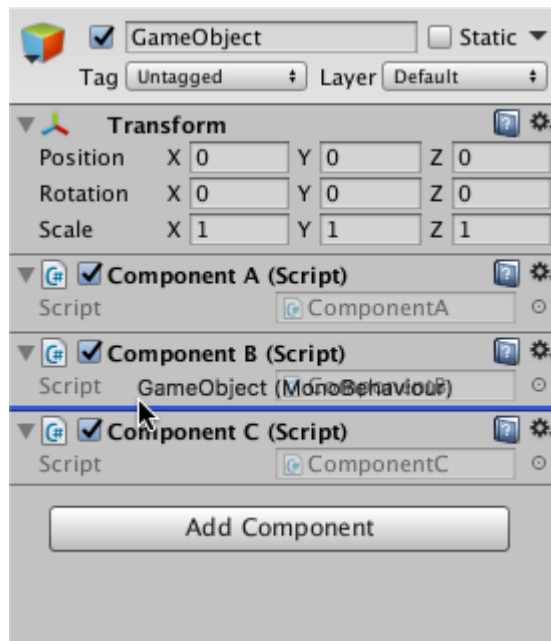
Ventana de Inspector



Sesión 1. Contenidos teóricos

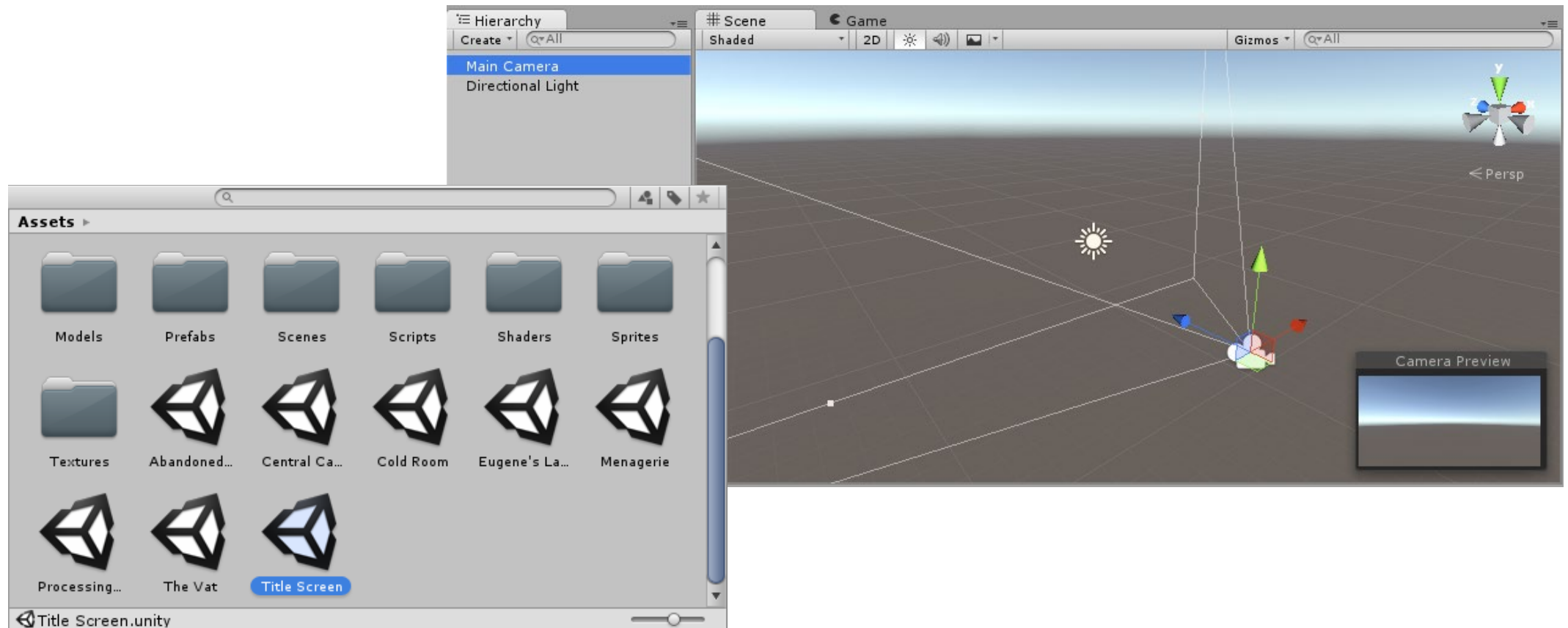
3. INTERFAZ DE UNITY

Ventana de Inspector



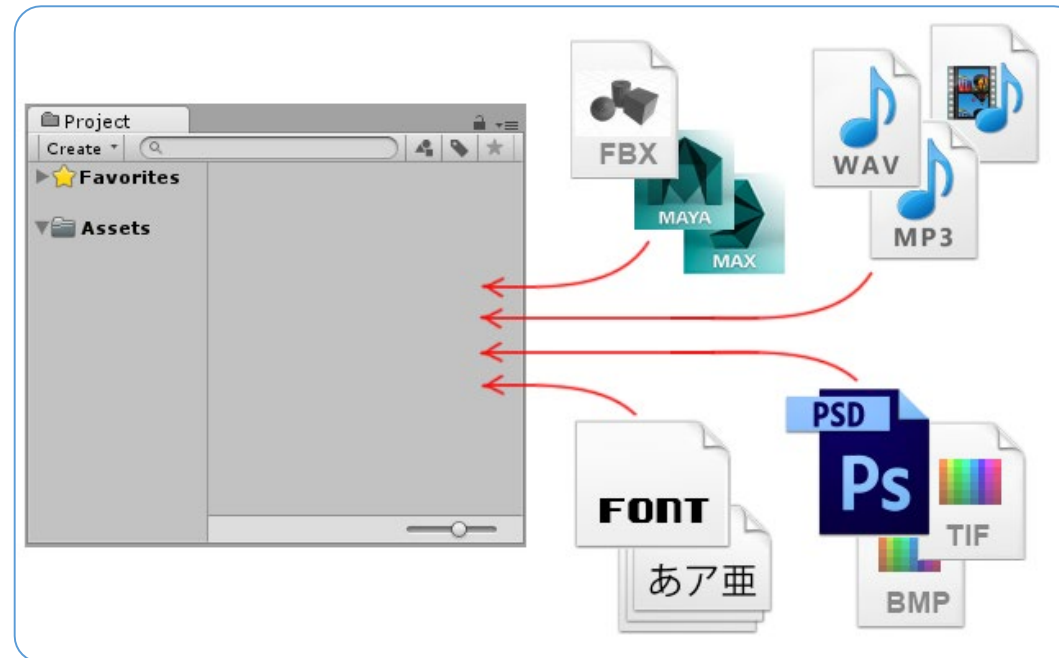
3. INTERFAZ DE UNITY

Escenas: contienen los entornos y menús del juego.



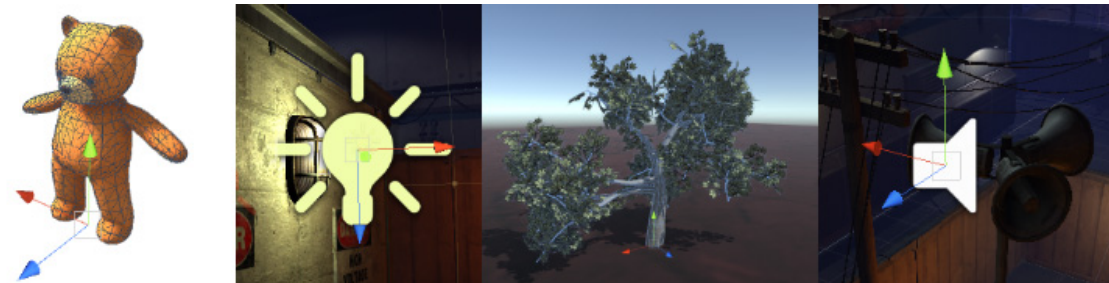
1. ASSETS

- Representación de cualquier ítem que puede ser utilizado en su juego o proyecto.
- Puede venir de un archivo creado fuera de Unity, tal como un modelo 3D, un archivo de audio, una imagen, o cualquiera de los otros tipos de archivos que Unity soporta.
- También hay otros tipos que pueden ser creados dentro de Unity, tal como un Animator Controller, un Audio Mixer o una Render Texture.



2. GAMEOBJECTS

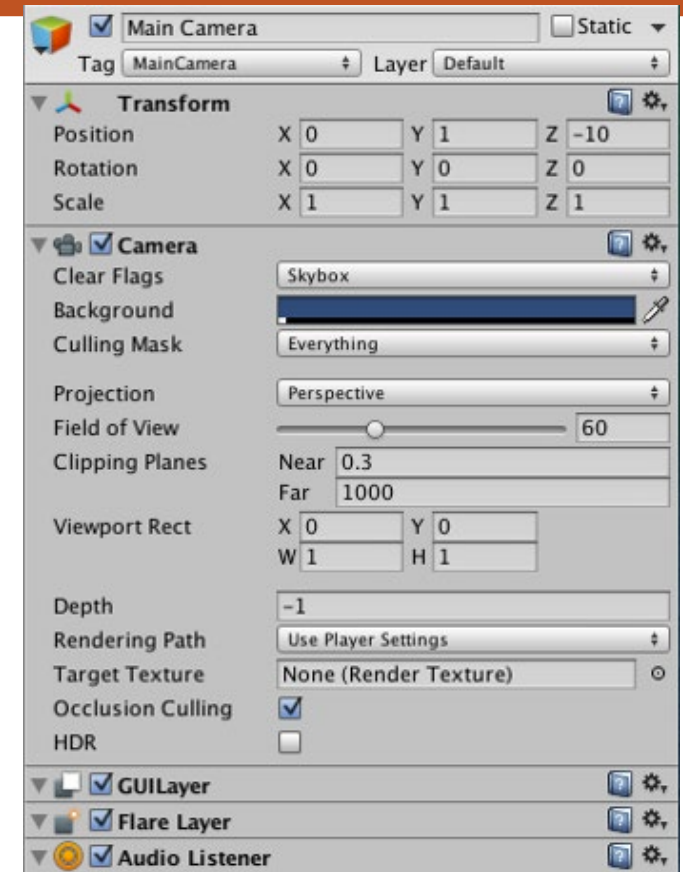
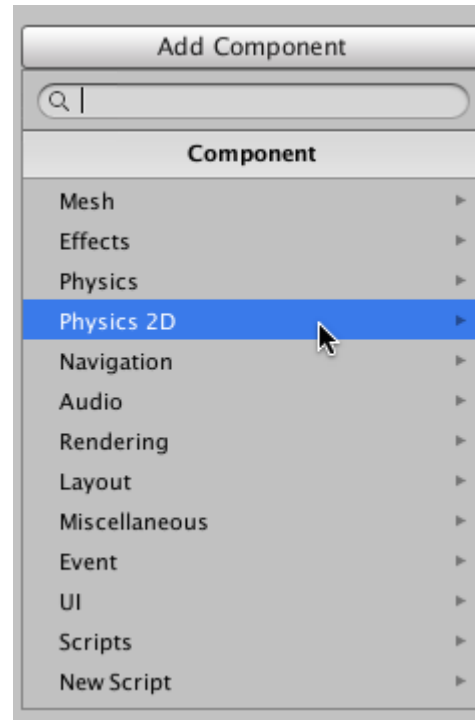
- Cada objeto en su juego es un GameObject, desde personajes y objetos coleccionables hasta luces, cámaras y efectos especiales.
- Sin embargo, un GameObject no puede hacer nada por sí mismo; se necesita darle propiedades antes de que pueda convertirse en un personaje, un entorno o un efecto especial.
- Para darle las propiedades que necesita para convertirse en una luz, un árbol o una cámara, se deben agregar componentes.



Sesión 2. Contenidos teóricos

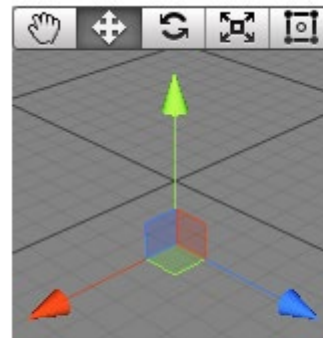
3. COMPONENTES

- [Rigidbody](#)
- [Collider](#)
- [Particle System](#)
- [Audio](#)

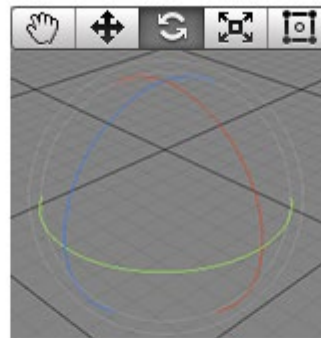


4. TRANSFORM

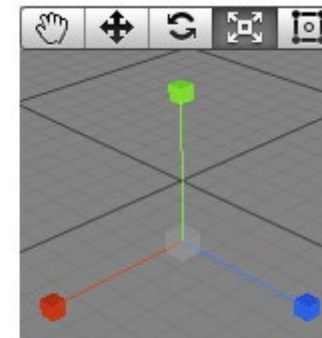
- El Transform es usado para almacenar la posición, rotación, escala y el estado de parentesco de un GameObject.
- Un GameObject siempre va a tener adjunto un Transform component - no es posible quitar un Transform o crear un GameObject sin uno.
- Más información: <https://docs.unity3d.com/es/current/Manual/Transforms.html>



Translate (W)



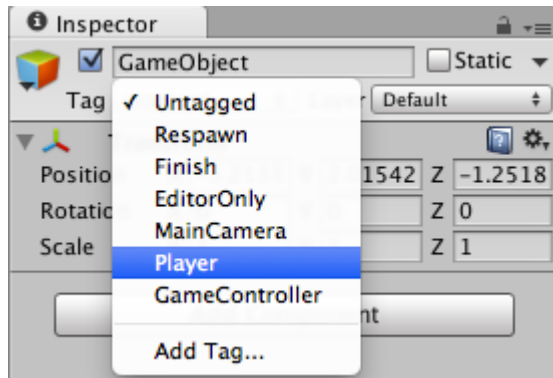
Rotate (E)



Scale (R)

5. TAGS EN GAMEOBJECTS

- Un Tag es una palabra de referencia que puede asignar a uno o más GameObjects.
- Por ejemplo, puede definir Tags de "Jugador" para los personajes controlados por el jugador y una Tag de "Enemigo" para los personajes no controlados por el jugador.



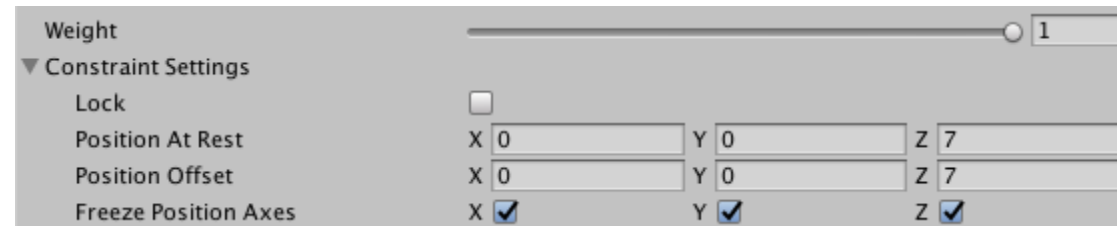
```
using UnityEngine;
using System.Collections;

public class Example : MonoBehaviour {
    public GameObject respawnPrefab;
    public GameObject respawn;
    void Start() {
        if (respawn == null)
            respawn = GameObject.FindWithTag("Respawn");

        Instantiate(respawnPrefab, respawn.transform.position, respawn.transform.rotation) as GameObject;
    }
}
```

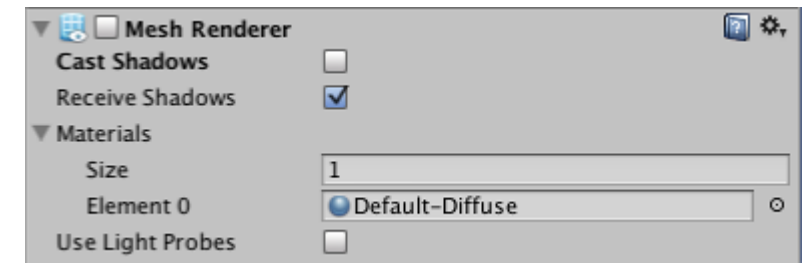
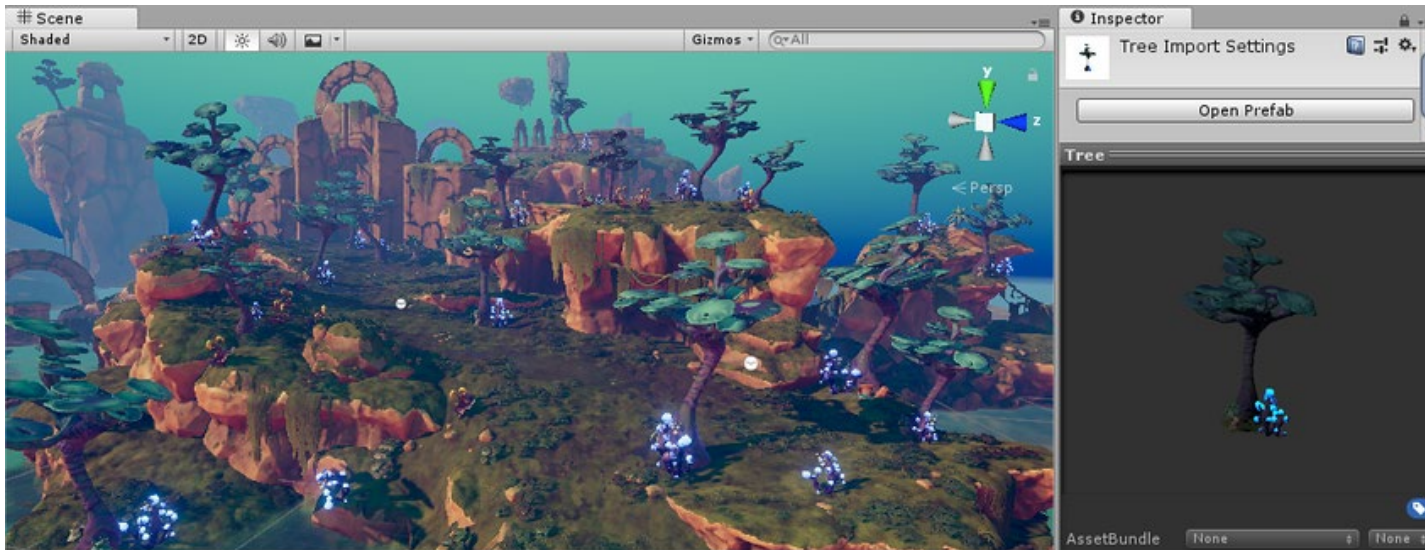
6. RESTRICCIONES

- Es posible enlazar el valor de un componente o gameObject al valor de otro, de tal manera que si cambiamos uno, se cambien ambos.
- Hay diferentes tipos de restricciones: aim, parent, position, rotation y scale.
- Más información: <https://docs.unity3d.com/es/current/Manual/Constraints.html>



7. PREFABS

- Plantilla a partir de la cual se pueden crear nuevas instancias del objeto en la escena.
- Cualquier edición hecha a un prefab asset será inmediatamente reflejado en todas las instancias producidas por él, pero, también se puede anular componentes y ajustes para cada instancia individualmente.



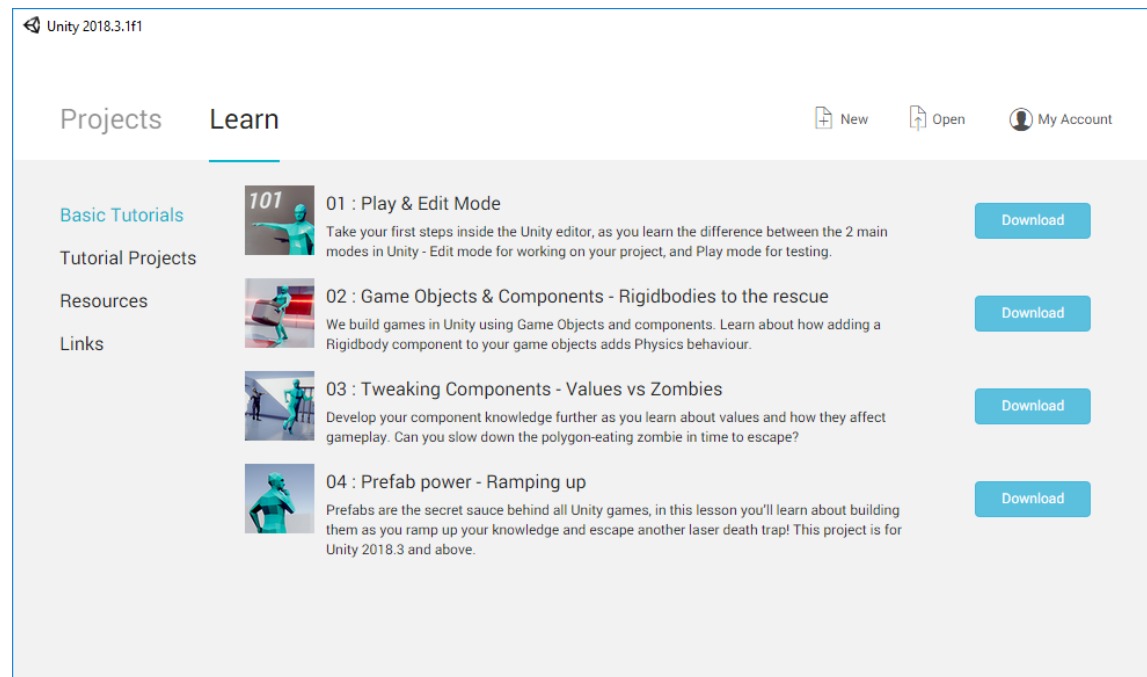
8. INPUTS

- Unity soporta teclado, ratón y mando para manejar nuestro videojuego.
- Se configura en el Input Manager:
- Configuración para PC: <https://docs.unity3d.com/es/current/Manual/ConventionalGameInput.html>
- Configuración para dispositivos móviles:
<https://docs.unity3d.com/es/current/Manual/MobileInput.html>

ACTIVIDAD 3. Recapitulación dentro del entorno

Para realizar una prueba más exhaustiva del entorno y aplicar lo visto en teoría, descarga y realiza los "Basic Tutorials" de Unity. Puedes acceder a ellos en la siguiente página:

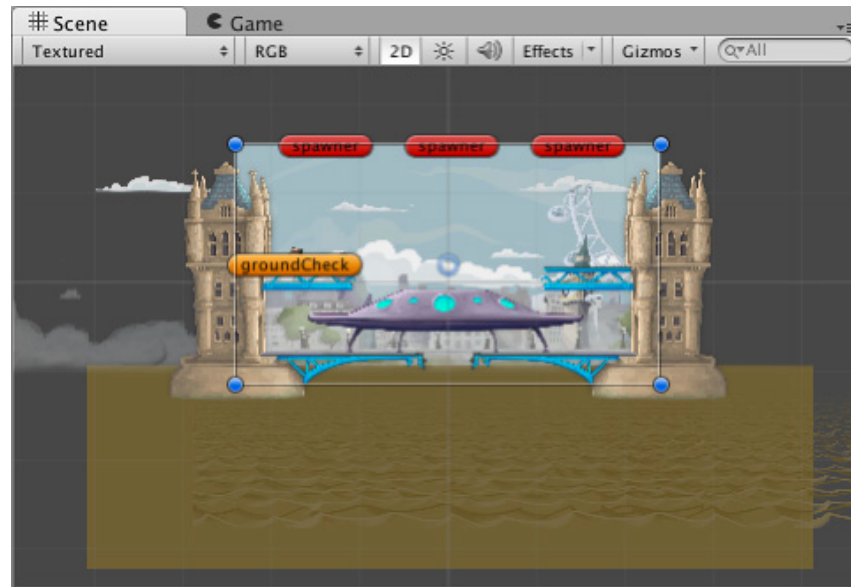
<https://unity3d.com/es/learn/tutorials/s/interactive-tutorials>



1. VISIÓN GENERAL DE UN JUEGO 2D

La característica que más se nota es el botón del modo en vista 2D en la barra de herramientas del Scene View.

Cuando el modo 2D es activado, una vista ortográfica (por ejemplo, libre de perspectiva) va a ser establecida; la cámara ve a lo largo del eje X con el eje Y aumentando hacia arriba. Esto le permite visualizar la escena y poner objetos 2D fácilmente.



2. GRÁFICOS 2D

Los objetos gráficos en 2D son conocidos como Sprites.

Los Sprites son nada más que unas texturas estándar pero hay varias técnicas para combinar y manejar las texturas sprites por rendimiento y conveniencia durante el desarrollo.

Para la configuración de sprites, se puede utilizar el Sprite Editor:

<https://docs.unity3d.com/es/current/Manual/SpriteEditor.html>



3. FÍSICAS 2D

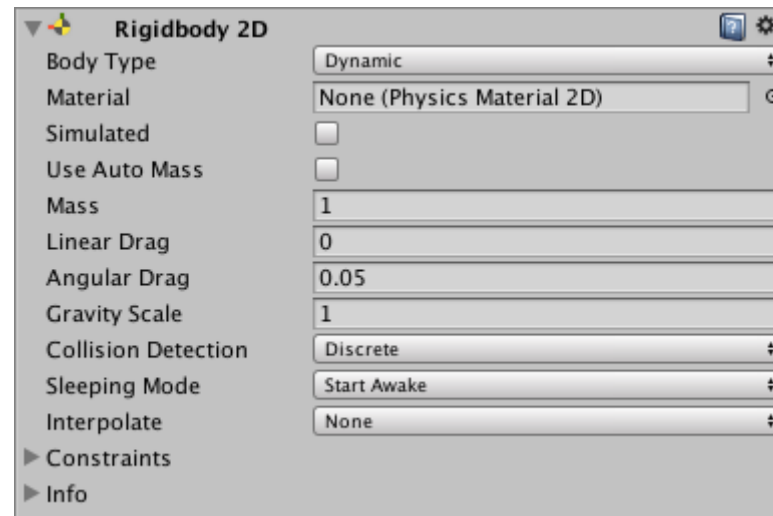
Unity cuenta con un motor de física separado para el manejo de la física en 2D con el fin de hacer uso de optimizaciones disponibles únicamente en 2D.

Los componentes específicos son:

- Rigidbody 2D: <https://docs.unity3d.com/es/current/Manual/class-Rigidbody2D.html>
- Box Collider 2D: <https://docs.unity3d.com/es/current/Manual/class-BoxCollider2D.html>
- Hinge Joint 2D: <https://docs.unity3d.com/es/current/Manual/class-HingeJoint2D.html>

3. FÍSICAS 2D

Un componente Rigidbody 2D coloca un objeto bajo el control del motor de física.

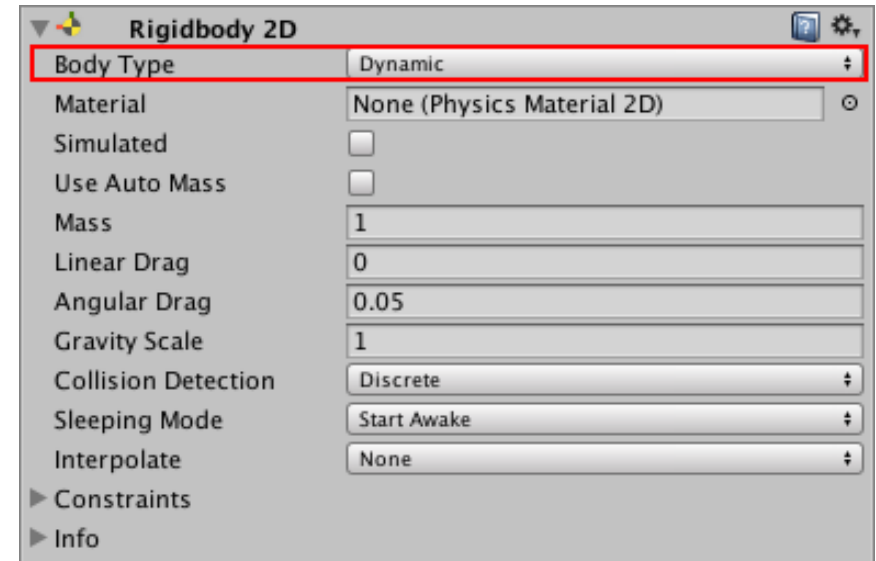
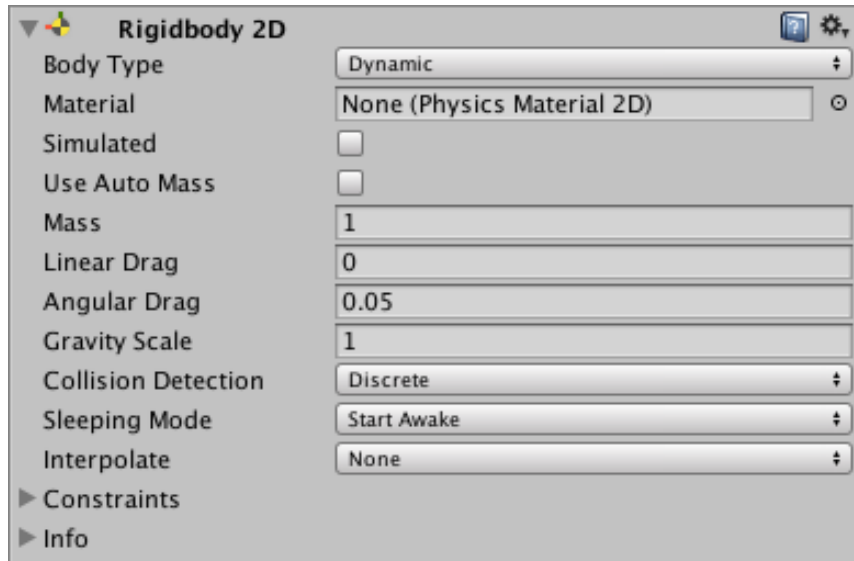


3. FÍSICAS 2D

Body Type: Dependiendo de este, veremos cambios en el comportamiento de movimiento (posición y rotación) y en la interacción con el Collider. Puede ser Dynamic, Kinematic y Static.

- **Dynamic:** Un Dynamic Rigidbody 2D está diseñado para moverse bajo simulación. Tiene todo el conjunto de propiedades disponibles, como la masa finita y el arrastre, y se ve afectado por la gravedad y las fuerzas. Un cuerpo dinámico colisionará con cualquier otro tipo de cuerpo y es el más interactivo de los tipos de cuerpo. Este es el tipo de cuerpo predeterminado para un Rigidbody 2D, porque es el tipo de cuerpo más común para las cosas que necesitan moverse.
- **Kinematic:** Un Kinematic Rigidbody 2D está diseñado para moverse bajo simulación, pero solo bajo un control muy explícito por parte del usuario. Mientras que los Rigidbody 2D Dynamic se ven afectados por la gravedad y las fuerzas, un Rigidbody 2D Kinematic no lo es.
- **Static:** Un Static Rigidbody 2D está diseñado para no moverse en absoluto en la simulación; si algo colisiona con él, un Static Rigidbody 2D se comporta como un objeto inamovible (como si tuviera una masa infinita).

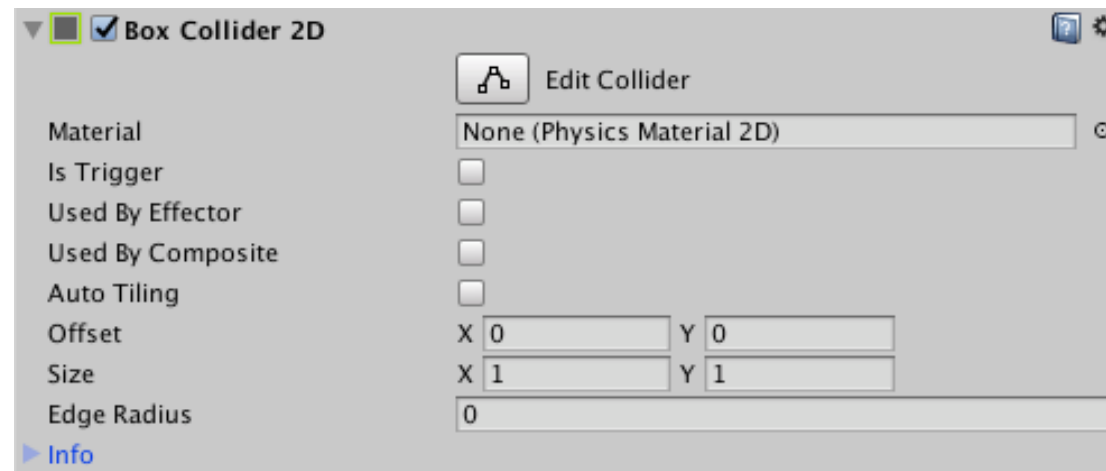
3. FÍSICAS 2D



3. FÍSICAS 2D

El componente Box Collider 2D es un Collider para usar con la física 2D. Su forma es un rectángulo con una posición, anchura y altura definidas en el espacio de coordenadas local de un Sprite.

Está alineado con el eje, es decir, sus bordes son paralelos a los ejes X o Y del espacio local.

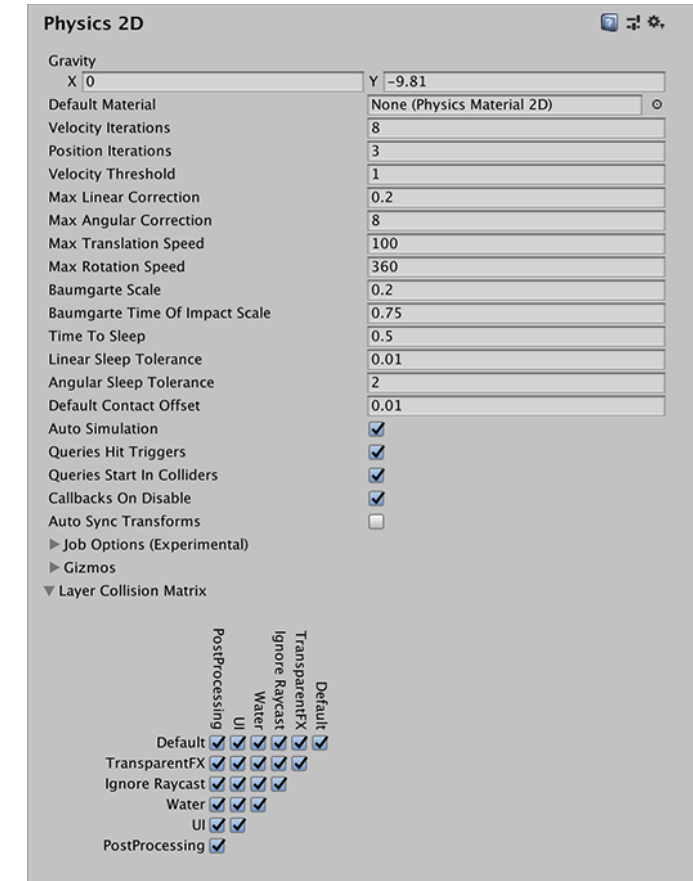


Sesión 3. Contenidos teóricos

3. FÍSICAS 2D

La configuración de las físicas 2D está en:
Project Settings → Physics 2D.

Propiedades: [Enlace](#)



4. SCRIPTING

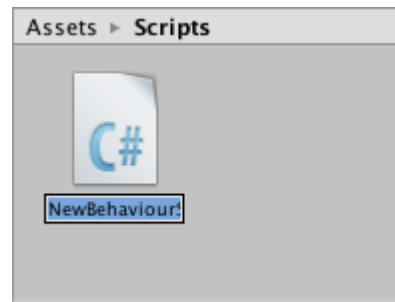
El flujo de ejecución de los scripts está guiado por el motor, de forma que para seguir ese flujo de ejecución debemos heredar de MonoBehaviour.

MonoBehaviour es una clase interna de Unity que será la encargada de llamar a los métodos en el orden preciso, actualizar, dar acceso al GameObject al que está añadido el componente del script, etc.

Creando scripts

Podemos crear scripts con la opción Assets > Create > C# Script.

El nuevo script será creado en la carpeta que se haya seleccionado en el Panel del Proyecto.



4. SCRIPTING

Métodos MonoBehaviour

- **void Awake():** llamado en el mismo momento que se cargue la instancia del script. Será llamado solamente una vez.
- **void Start():** llamado cuando el objeto se active y justo antes de llamar a cualquier Update(). Las acciones que se harían en un constructor, hay que hacerlas en este método.
- **void Update():** llamado continuamente por el motor. Muy útil para actualizar el estado del GameObject.
- **void LateUpdate():** se llama después de cada Update().
- **void FixedUpdate():** igual que Update(), pero nos aseguramos de que tenemos un número fijo de llamadas a este método.

4. SCRIPTING

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

}
```

4. SCRIPTING

Otros métodos MonoBehaviour

void OnEnable(): llamado cuando el GameObject se activa.

void OnDisable(): llamado cuando el GameObject se desactiva.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

}
```

4. SCRIPTING

Búsqueda de componentes

```
// Búsqueda de un componente de un tipo  
Component c1 = GetComponent("Transform");  
Transform c2 = GetComponent<Transform>();  
  
GameObject go1 = GameObject.Find("nombre_GameObjet");  
GameObject[] goArr = GameObject.FindGameObjectsWithTag("tag");
```

Más información: <https://docs.unity3d.com/es/2018.3/Manual/ScriptingSection.html>