Desarrollo de entornos interactivos multidispositivo

UT1.- LOS VIDEOJUEGOS

¿Qué son los videojuegos?

Podemos definir de forma muy genérica un videojuego como una “película de animación interactiva” que se modifica por la acción de una o más personas.

Por lo tanto vamos a necesitar un dispositivo que muestre las imágenes y algún tipo de controlador que permita la interacción del usuario.

Además de la imágen y la interacción, otro de los elementos clave en los videojuegos es el sonido.

También podemos utilizar dispositivos que transmitan sensaciones al usuario vibraciones, fuerzas...) y que se conocen como periféricos hápticos.

¿Qué son los videojuegos?

Los videojuegos producen la sensación de movimiento gracias a la generación de muchos fotogramas por segundo.

A diferencia de las películas de animación en las que los fotogramas, que acaban formando una secuencia, se han de calcular uno a uno (con tiempos de cálculo generalmente muy grandes), en los videojuegos los fotogramas se generan en tiempo real y teniendo en cuenta además, la interacción del usuario.

Para que la sensación de movimiento sea lo más fluida posible, es necesario generar muchos fotogramas por segundo; normalmente se consideran aceptables unos 30 fps (frames por segundo) aunque los 60fps sería una cifra óptima.

Glosario

Glosario de términos de videojuegos

En el mundo de los videojuegos se utiliza un conjunto de términos y palabras muy específico y que no vendría mal que viésemos, aunque sea de forma muy sencilla.

Sistema de juego

Un **sistema de juego** es una regla o conjunto de reglas cuyo objetivo consiste en obtener una serie de resultados coherentes en el seno de un juego.

Gameplay (Jugabilidad)

El **Gameplay** es el conjunto de acciones a las que tiene acceso el jugador, gobernado por las interacciones de uno o más **sistemas de juego**.

Glosario de términos de videojuegos

HUD (Head-Up Display)

Se llama **HUD** a la información que se muestra en pantalla durante el juego, generalmente en forma de iconos y textos. El HUD suele mostrar información al usuario como el número de vidas, los puntos, el nivel de salud, minimapas...

Assets (Recursos)

Los assets son todos los elementos que componen un juego (modelos, texturas, scripts, sonidos, animaciones...)

Frame

Cada una de las imágenes estáticas que forman parte de la sucesión de imágenes que componen una animación, y que producen sensación de movimiento.

Glosario de términos de videojuegos

Textura
Imagen en dos dimensiones (bitmap o mapa de bits) que se puede utilizar para aplicar sobre un objeto en tres dimensiones cubriéndolo y simulando una superficie (madera, piedra, metal, etc), como elemento gráfico en el HUD o como una animación de personajes en un juego 2D.

Frame rate
Frecuencia con la que un juego muestra sucesivas imágenes o cuadros en pantalla. Generalmente se mide en FPS (Frames per Second o Cuadros por segundo). A mayor frame rate, mayor fluidez y suavidad se aprecia en el movimiento.

En videojuegos se considera aceptable un frame rate de entre 30-60 FPS siendo ésta última considerada una medida óptima, aunque ciertos juegos puede superar con el hardware adecuado los 100 FPS.

Glosario de términos de videojuegos

First Person Game (Juego en Primera Persona)

Tipo de juego (normalmente de disparos o **shooter**) donde la perspectiva con la que jugamos es desde el punto de vista del propio personaje que controlamos. Normalmente sólo vemos el arma y/o el brazo del personaje.

Third Person Game (Juego en Tercera Persona)

En este caso la vista con la que jugamos es desde detrás del personaje, con un encuadre de la cámara que permite verlo completamente (o casi). En el caso de los Third Person Shooters en 3D, la cámara a menudo se sitúa a la altura del hombro para facilitar el apuntado con el arma.

Glosario de términos de videojuegos

Power-up (Potenciador)

Objeto que otorga al jugador una característica o capacidad especial, como invencibilidad, mayor velocidad, mayor poder de ataque, etc.

La lista continúa...

Aquí tenéis un listado (en castellano) mucho más completo, muy bien organizado y casi interminable...

https://www.devuego.es/gamerdic/

¿Cómo se crea?
Fases de la creación

¿Cómo se crean?

La creación de un videojuego pasa por varias fases.

Fase 1: Concept

Igual que ocurre con la mayoría de proyectos, los videojuegos comienzan con una primera idea a partir de la cual se determinan aspectos fundamentales como el género, el gameplay, el estilo de los personajes, el ambiente, la música...

Este Concept normalmente se plasma en un sencillo guión gráfico (storyboard) con todas las ideas que suelen pasar por varios procesos de adaptación.

¿Cómo se crean?

Fase 2: Pre-producción

Con el Concept decidido, podemos pasar a definir los elementos que componen el juego. Desarrollamos la historia, creamos versiones iniciales de guiones con los objetivos, decidimos los personajes principales, etc.

Utilizando estos esbozos de guiones, los artistas se ponen manos a la obra para crear diseños previos del aspecto del juego, la forma en que se visualizarán los personajes, los escenarios, objetos, etc. Su trabajo es presentar propuestas visuales para ir dando forma a la idea original.

¿Cómo se crean?

Fase 2: Pre-producción

También se describen los elementos sonoros de los que consta el juego: efectos de sonido, ambientación, música, voces...

Paralelamente se especifica el funcionamiento general del videojuego.

Finalmente se hace el diseño de la programación, que describe la manera en la que se implementará el videojuego, el lenguaje o lenguajes de programación que se utilizarán, las metodologías que se seguirán, etc.

¿Cómo se crean?

Fase 2: Pre-producción

Se realiza una planificación de todas las tareas, con el reparto del trabajo, los plazos de entrega, las reuniones de seguimiento...

Todo lo anterior tiene como objetivo generar el Documento de Diseño del Juego o **GDD (Game Design Document)** que especificará el desarrollo del arte, las mecánicas, la programación del videojuego y la planificación.

Si es posible, se realiza un prototipo muy sencillo del videojuego.

¿Cómo se crean?

Fase 3: Producción

Una vez tenemos claro lo que hay que hacer, cómo hacerlo y se ha planificado el tiempo, empezamos la producción con el objetivo de crear el juego, como mínimo en una versión inicial que se mejorará gradualmente.

Lo primero que hemos de hacer es generar **todos** los assets del juego teniendo como guía el **GDD**: programación, texturas, desarrollo de interfaces, animación, modelado, desarrollo de sonidos, etc.

¿Cómo se crean?

Fase 3: Producción

Con todos los assets preparados (o la mayoría) pasamos a “ensamblar” o editar el juego. Aquí es donde utilizamos los **motores de juegos** (game engines) como **Unity** o **Unreal**.

Si logramos ensamblar correctamente todos los assets, esta fase culmina (por ahora). Sin embargo, al igual que en el desarrollo de software tradicional, es muy difícil que todo salga bien a la primera, por lo que se entra en una fase para probar a fondo el videojuego.

¿Cómo se crean?

Fase 4: Pruebas

En esta fase se corrigen los errores del proceso de edición inicial y se mejora la jugabilidad a medida que se prueba el juego.

Generalmente encontraremos dos tipos: las **pruebas alpha**, realizadas por un pequeño grupo de personas, generalmente involucradas en el desarrollo, y las **pruebas beta**, realizadas por un equipo externo de jugadores. Las primeras tienen el objetivo de corregir defectos graves y mejorar características fundamentales no contempladas en el **GDD**, mientras que las segundas se enfocan en la detección de fallos menores y en perfilar la experiencia de usuario.

¿Cómo se crean?

Fase 5: Post-producción

Durante esta fase nos encargamos del **marketing**, **distribución** y **mantenimiento**.

La **distribución** consiste en crear copias del juego terminado y llevarlo a las tiendas (ya sean físicas o digitales) para su venta (aunque sean gratuitos).

El proceso de **marketing** es fundamental para dar a conocer el videojuego y conseguir el mayor número de jugadores posibles.

¿Cómo se crean?

Fase 5: Post-producción

Las tareas de márketing no tienen un orden concreto dentro del desarrollo, pero es habitual hacer campaña de un videojuego mucho tiempo antes de publicarlo.

Tanto la distribución como el márketing suelen delegarse en empresas externas.

Aunque el juego ya esté distribuido, su ciclo de vida todavía no ha terminado. La fase de **mantenimiento** es el momento de arreglar nuevos errores, mejorarlo, etc. Ésto se hace sacando parches o actualizaciones nuevas al mercado.

¿Cómo se crean?

Fase 5: Post-producción

El **mantenimiento** del videojuego es también una magnífica oportunidad para seguir sacándole partido.

Ya sea en forma de microtransacciones, suscripciones de pago o incluso con expansiones completas que añaden nuevas características al videojuego sin modificar en profundidad el motor del mismo, mediante las tareas de **mantenimiento** se puede aprovechar al máximo la base inicial del videojuego.

¿Cómo funciona?
Secuencia de funcionamiento

¿Cómo funcionan?

Podemos estructurar el funcionamiento de cualquier videojuego en tres partes secuenciales y bien diferenciadas:

1: Inicialización (https://docs.unity3d.com/Manual/ExecutionOrder.html)

La fase de inicialización en memoria incluye la carga de librerías, scripts, objetos y colecciones dinámicas, la declaración de variables, la asignación de valores iniciales, etc...

También se suele hacer una precarga de recursos para poder utilizarlos más rápidamente, como por ejemplo imágenes, modelos, fuentes o sonidos...

¿Cómo funcionan?

1: Inicialización (https://docs.unity3d.com/Manual/ExecutionOrder.html)

La clave para optimizar el juego consiste en precargar solamente lo necesario en cada momento. Esto se puede hacer gracias a que los juegos se organizan internamente en pantallas o escenarios.

Con todo lo necesario en la memoria pasamos a crear el espacio de juego y, a continuación, cargamos los componentes y objetos asignándoles las posiciones y valores iniciales relativos al espacio, ya sea en 2D o en 3D.

¿Cómo funcionan?

2: Bucle de juego (https://docs.unity3d.com/Manual/ExecutionOrder.html)

Esta parte es la más importante ya que controla los procesos fundamentales de la lógica del juego y el renderizado (o dibujo), creando la ilusión óptica que nos permite percibir el espacio como algo real y con el que podemos interactuar.

Durante la lógica del juego se capturan los eventos del teclado, del ratón, se redefinen las propiedades de los objetos (como su posición en el espacio, su velocidad o su apariencia), se comprueba si hay colisiones entre objetos, se cambian sus estados y un largo etcétera.

¿Cómo funcionan?

2: Bucle de juego (https://docs.unity3d.com/Manual/ExecutionOrder.html)

Una vez hemos analizado la lógica y sabemos qué debe hacer cada componente y objeto del escenario, podemos proceder a su renderizado. El resultado del proceso de renderizado será un fotograma o frame.

Este proceso de análisis de la lógica del juego y posterior renderizado de la escena, se ha de repetir muchas veces por segundo, por eso esta parte se la conoce como bucle de juego.

¿Cómo funcionan?

3: Liberación (https://docs.unity3d.com/Manual/ExecutionOrder.html)

Por último, pero no por ello menos importante, cuando decidimos que el juego debe finalizar es fundamental realizar las tareas de limpieza de la memoria y liberación de todos los datos almacenados.

Game Engine

Los Game Engines

Un **Game Engine** nos debe facilitar el desarrollo de videojuegos para múltiples plataformas, de forma que nos permita reducir tanto los costes como los tiempos de desarrollo.

Por esto, es interesante que al elegir un Game Engine, le pidamos una serie de características:

- **Editor potente:** creación de escenas 3D, importación de assets en diferentes formatos, editor de scripts, creación de animaciones, desarrollo en 2D y 3D, editor de GUI, previsualización del juego, gestión de sonidos...

Los Game Engines

- **Motor de físicas** (en 2D y 3D) y **renderizado** avanzados.
- **Motor de scripts** eficiente.
- **Distribución multiplataforma.**
- Generador de **sistemas de partículas**.
- Editor de **terrenos**.
- Gestión de **LODs** (Level Of Detail).
- **Documentación** lo más completa posible.
- **Ecosistema de terceros** grande y eficiente.
- **Coste reducido.**
- **Curva de aprendizaje** corta.

Unity

Unity es un motor de videojuegos con soporte 2D, 3D, VR (realidad virtual) y AR (realidad aumentada) creado por Unity Technologies, anunciado y lanzado por primera vez en junio de 2005 en la Conferencia Mundial de Desarrolladores de Apple como motor de juegos para Mac OS X.

Está disponible como plataforma de desarrollo para Microsoft Windows, Mac OS, Linux.

Ofrece portabilidad multiplataforma.

Es ideal para desarrolladores indies. Hasta que no generas un mínimo de ingresos, actualmente 100.000 $/año.

Coordenadas

Sistemas de coordenadas

Unity utiliza un sistema de coordenadas cartesianas que nos permite recrear el espacio y ubicar en él todos los elementos que formarán el videojuego.

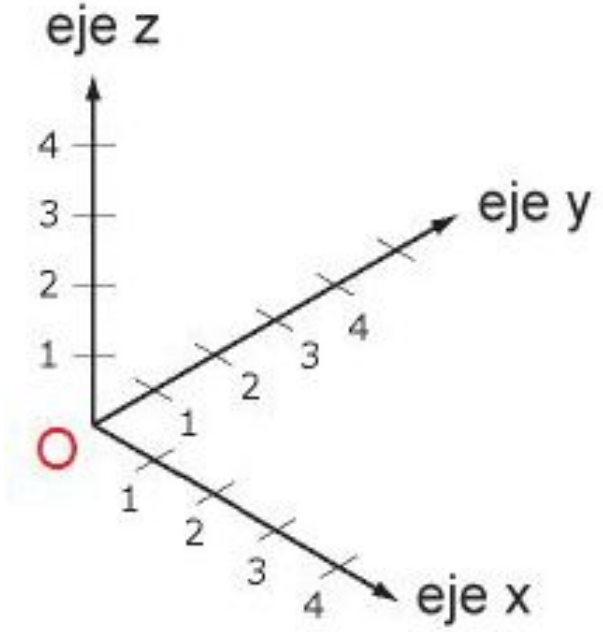
El **sistema de coordenadas cartesianas** es una parte fundamental de las matemáticas y de la física. Está compuesto por un **origen de coordenadas** y **tres ejes (X, Y y Z)**, que representan cada una de las tres dimensiones.

La mayoría de los programas de modelado 3D utilizan un sistema de coordenadas conocido como **Z-up**, en el que el eje X representa la anchura, el eje Y la profundidad y el eje Z la altura (ojo: Unity no usa este sistema de coordenadas).

Sistemas de coordenadas

Aquí podemos ver el **sistema de coordenadas cartesianas** con su **origen** de coordenadas y sus tres **ejes**.

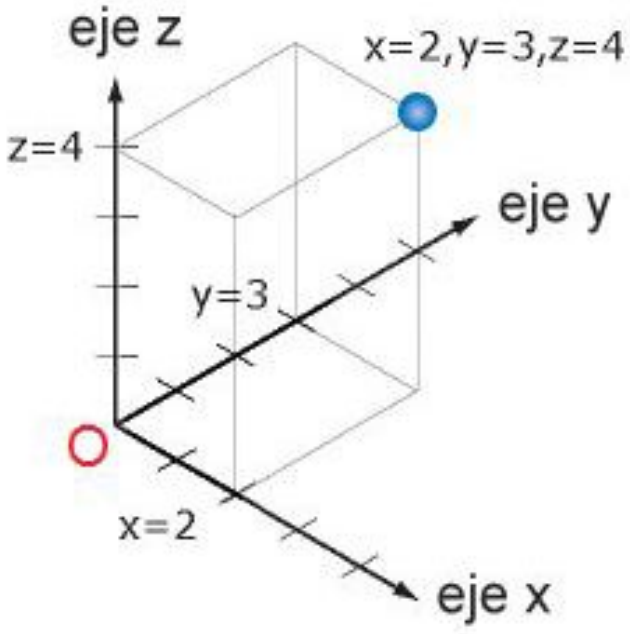
Podemos representar fácilmente la unidad básica en el espacio, el **punto**, indicando sus valores en cada uno de los ejes. El **punto p** se representa como **p = (X, Y, Z)**.



Sistemas de coordenadas

Así, el punto A= (2, 3, 4) se representaría de este modo en el espacio de coordenadas.

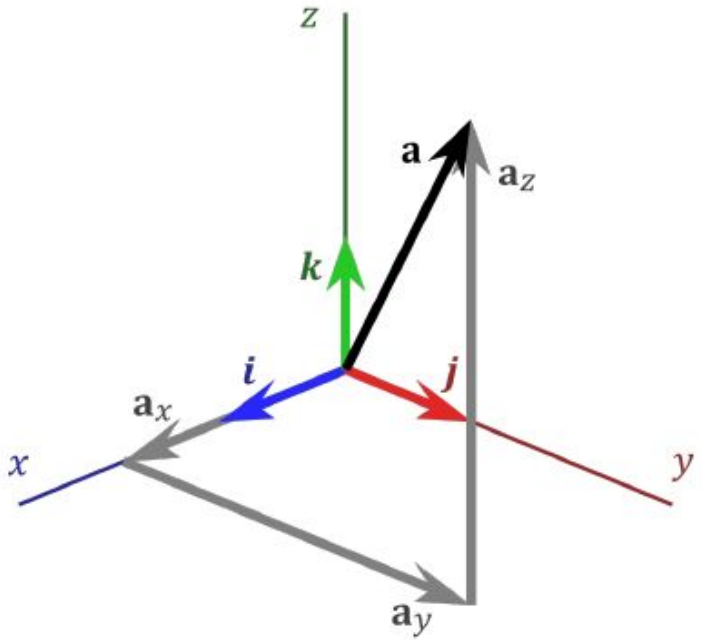
Los valores positivos en un eje se representan desde el origen en la dirección que indica el eje de coordenadas, y los valores negativos en la dirección contraria. En el caso del **eje X**, los valores positivos estarán a la derecha del origen de coordenadas y los valores negativos a la izquierda.



Sistemas de coordenadas

Otro elemento que se puede representar sobre el sistema de coordenadas es un **vector**, que no es más que la distancia entre el origen y un punto, en dirección hacia ese punto.

En los videojuegos, los vectores se utilizan para calcular distancias y ángulos relativos entre dos objetos, para representar fuerzas como la de la gravedad o para realizar el movimiento de objetos utilizando el motor físico.



Arte 2D:
Sprites y Tiles

Sprites y spritesheets

Los **Sprites** (del inglés, duendecillos) son imágenes bidimensionales únicas que representan objetos o personajes que pueden ser movidos, escalados y rotados.

Los **Spritesheets** son imágenes bidimensionales únicas que contienen múltiples frames de sprites, dispuestos en una posición dentro de la rejilla (grid), y utilizados para representar animaciones.

Sprites y spritesheets

Los **Tiles** (del inglés, azulejos) son imágenes bidimensionales que representan entornos, fondos y plataforma para construir la escena de un juego.

La colección de imágenes organizadas en cuadrículas se conoce como **Tileset**.

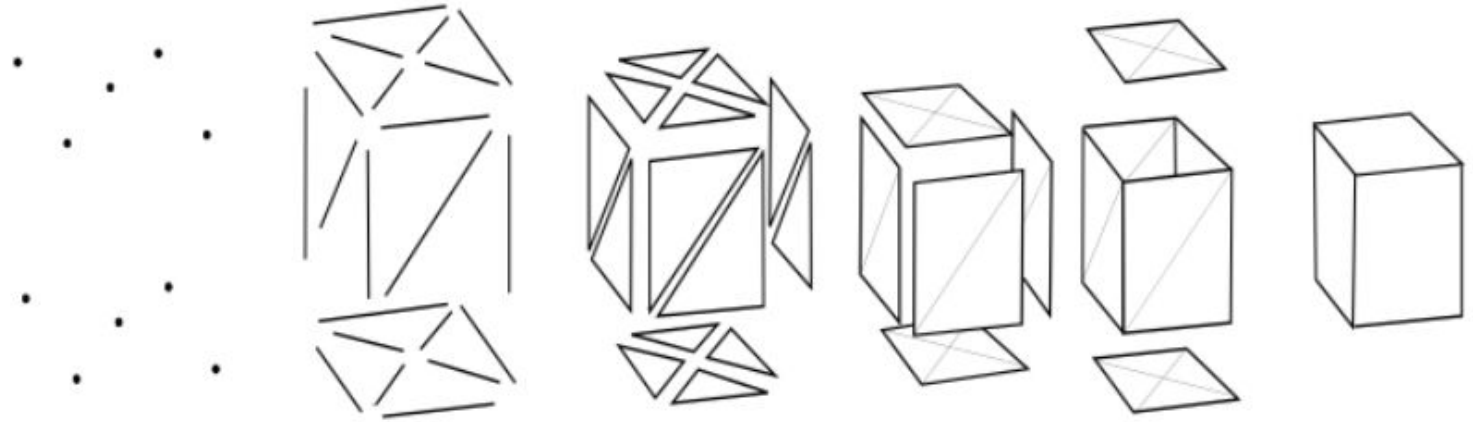
Las escenas creadas con Tiles se conoce como **Tilemap**.

Arte 3D: Vértices, aristas, y polígonos

Vértices, aristas, polígonos y mallas

Muchos de los objetos que vamos a utilizar en un videojuego serán modelos 3D. Estos objetos se definen internamente como un conjunto de muchos puntos en el sistema de coordenadas cartesianas que acabamos de ver.

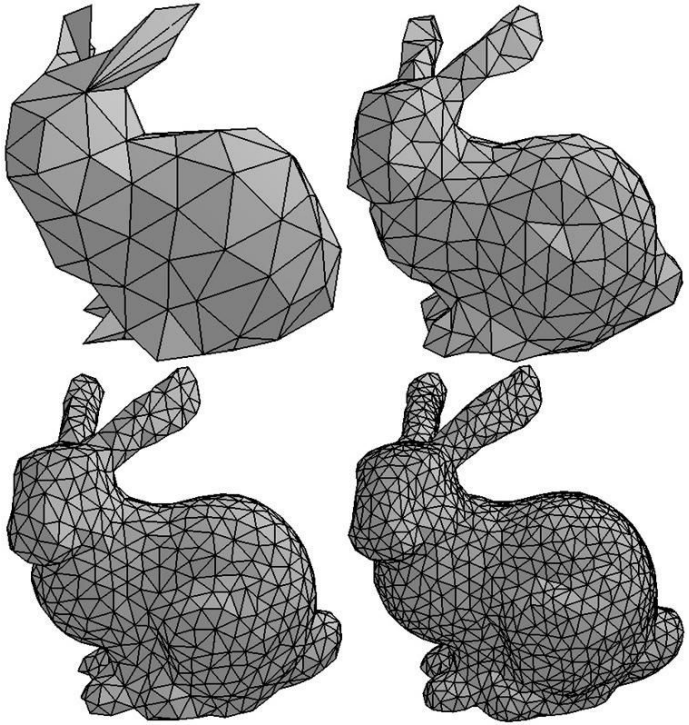
Cada uno de estos puntos representaría un **vértice** del objeto.



Vértices, aristas, polígonos y mallas

La unión de dos vértices (**vertex**), formaría una arista (**edge**), y uniendo tres aristas obtendremos un polígono (**face**).

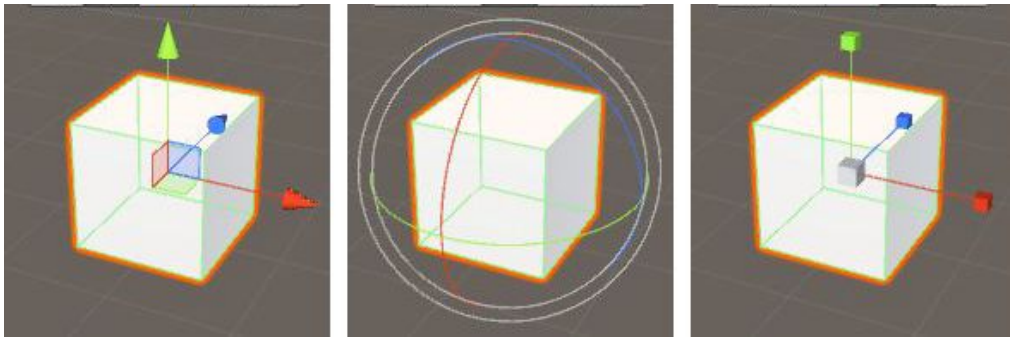
La conexión de muchos polígonos nos proporciona una malla (**mesh**) que conforma la "piel" del objeto 3D.



Vértices, aristas, polígonos y mallas

Un concepto importante en los objetos 3D (y 2D) es el punto pivote o pivot point, que indica el centro de coordenadas del propio objeto y, por tanto, el punto de referencia para sus transformaciones (posición, rotación y escala).

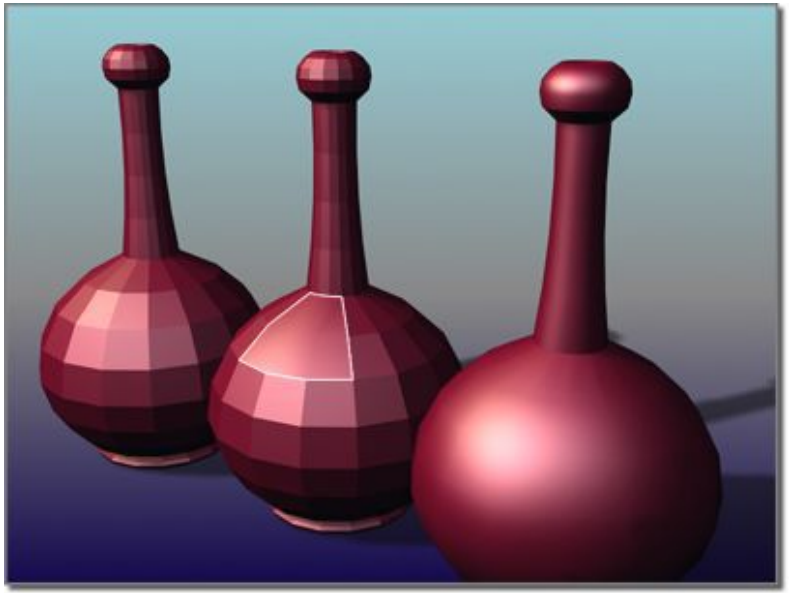
En las primitivas básicas (cubo, esfera, cilindro, plano...) suele situarse en el centro del objeto, pero en nuestros modelos lo situaremos donde más nos interese.



Vértices, aristas, polígonos y mallas

Es importante utilizar un número de polígonos pequeño para reducir los tiempos de cálculo al renderizar los objetos.

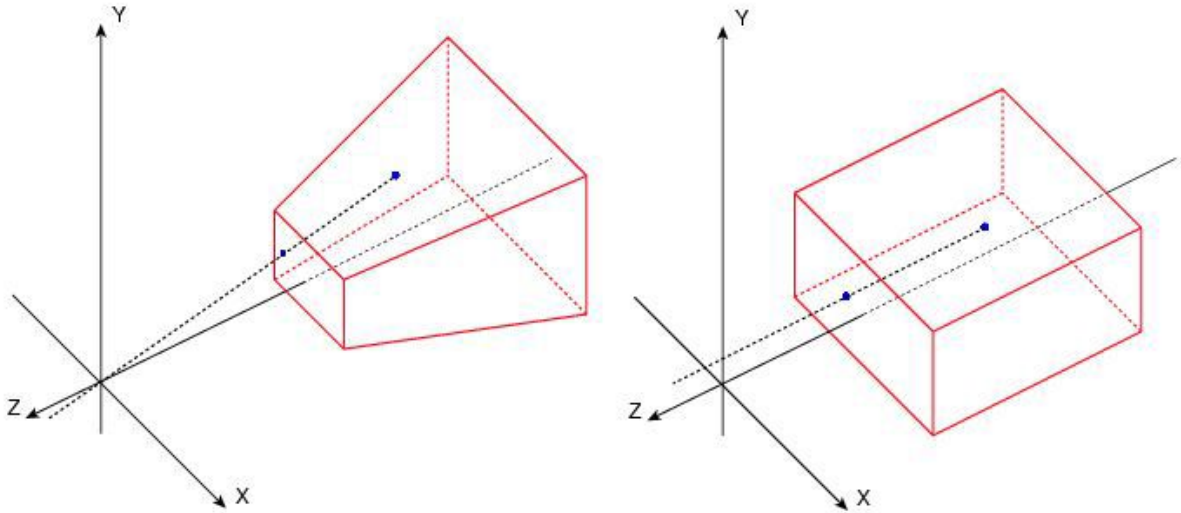
A pesar de usar pocos polígonos, podemos obtener un buen acabado de los objetos utilizando técnicas como los grupos de suavizado o **smoothing groups**.



Proyecciones

Proyecciones de cámara

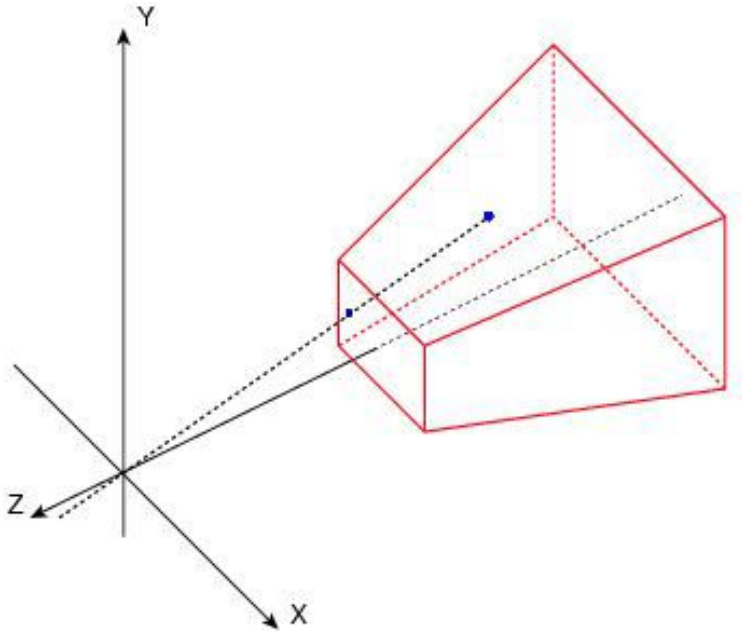
Nuestro juego siempre se va a visualizar a través de una cámara, que podremos configurar con dos tipos de proyecciones, **perspectiva** para representar el mundo en 3D u **ortográfica** para una representación en 2D.



Proyecciones de cámara

Por defecto, las cámaras se configuran con una proyección en **perspectiva**.

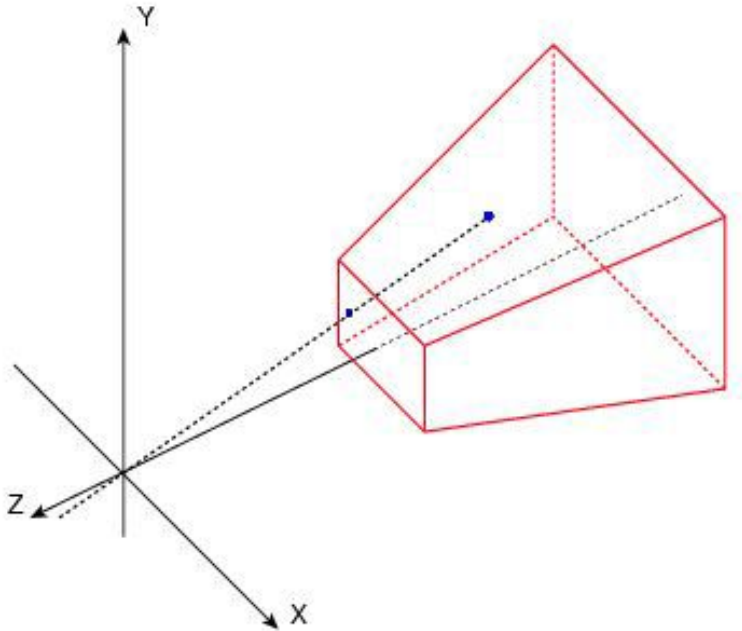
En este tipo de proyección, el campo de visión de la cámara lo forma una pirámide truncada, o **frustum**, cuya forma está determinada por el ángulo de visión (anchura de la pirámide), por la distancia máxima de visión (la base de la pirámide) y por la distancia mínima de visión (el corte de la punta de la pirámide).



Proyecciones de cámara

Cuando usamos una proyección en **perspectiva**, el espacio se representa en 3D y, por lo tanto, los objetos se verán más pequeños según se alejen de la posición de la cámara.

Los objetos que no se encuentren dentro del campo de visión de la cámara (frustum) no se visualizarán. Esta comprobación es la primera que se hace antes de renderizar una escena.

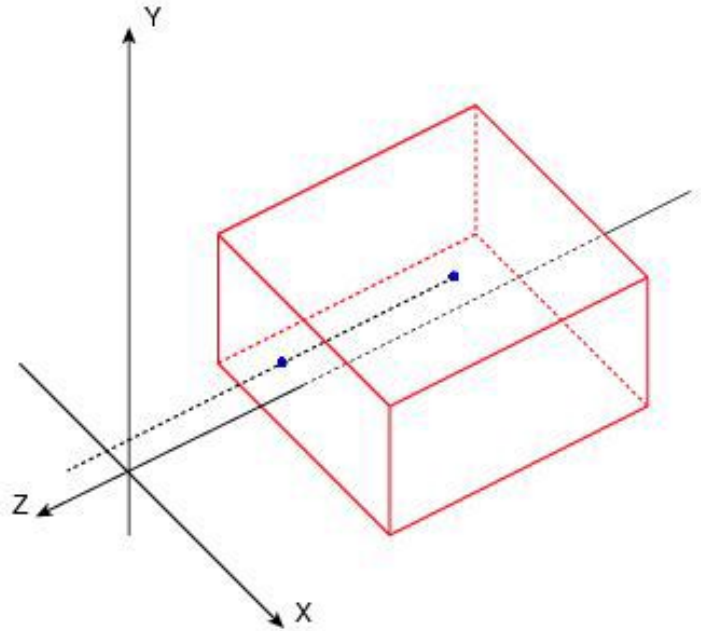


Proyecciones de cámara

Cuando usamos una proyección **ortográfica**, el campo de visión de la cámara pasará a ser un prisma rectangular.

Con este tipo de proyección el espacio se representa en 2D y, en este caso, el tamaño de los objetos se mantendrá fijo por mucho que se alejen de la posición de la cámara.

La proyección **axonométrica** es un tipo especial de proyección ortográfica.



Color, iluminación y acabados

Materiales y texturas

El primer paso para crear un objeto 3D consiste en obtener la malla o mesh, que sería como la piel del objeto.

Posteriormente pasamos a darle un acabado más realista aplicándole materiales y/o texturas.

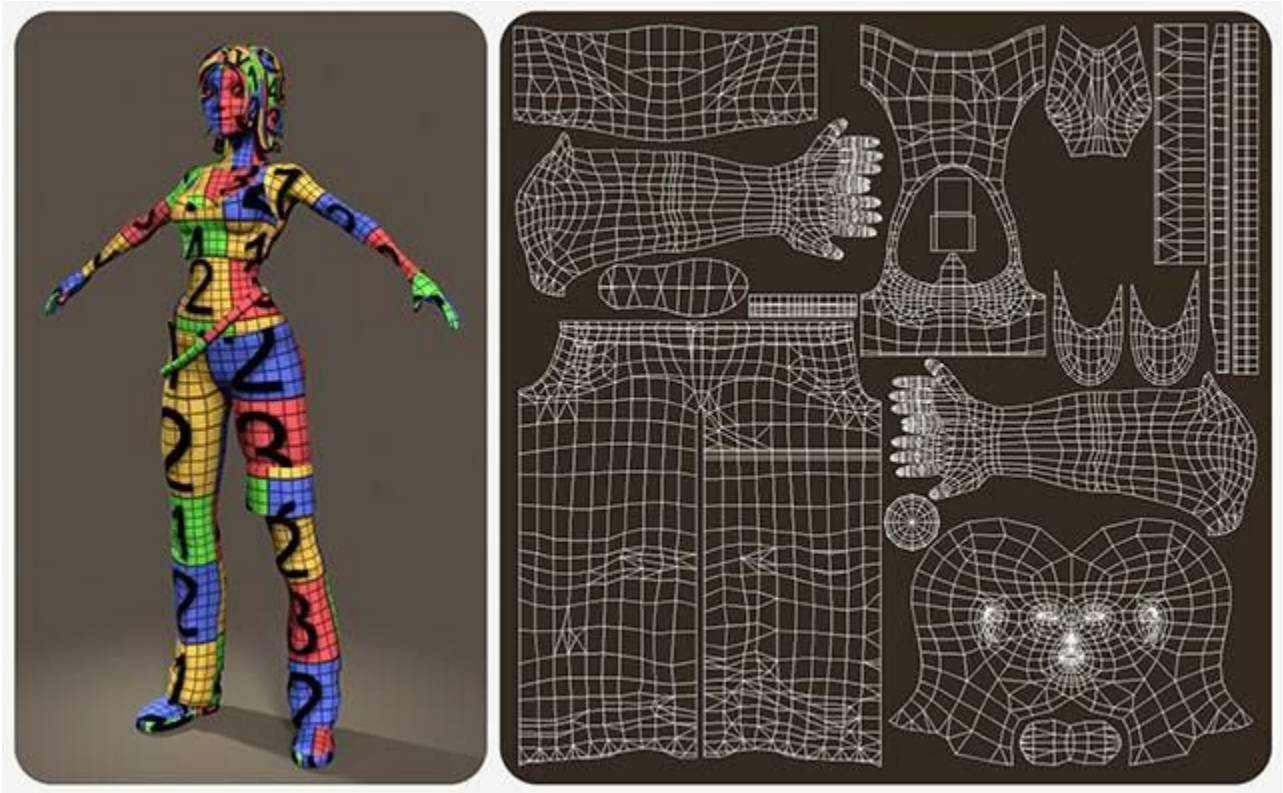
Si vamos a utilizar materiales sencillos, como metales o plásticos, podemos conseguir un buen acabado simplemente creando un material básico, con un color y unos valores de brillo o matizado.

Materiales y texturas

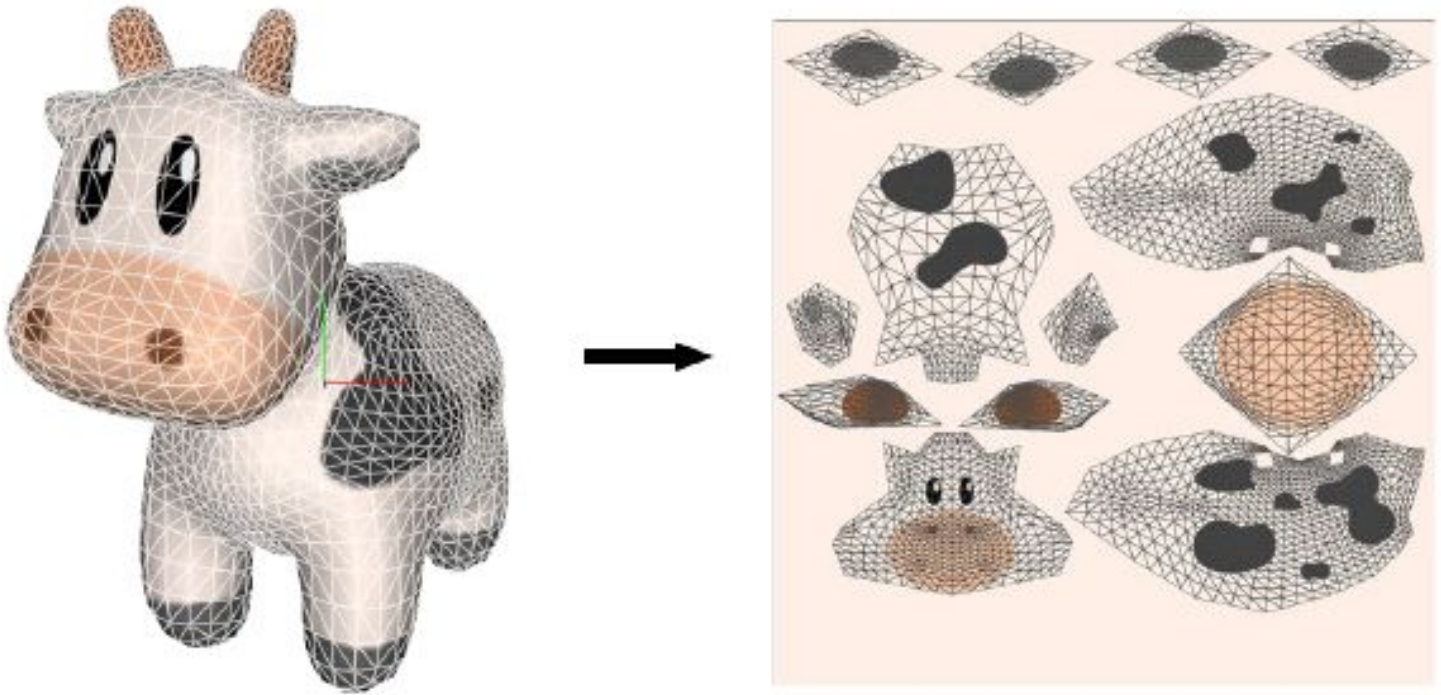
Sin embargo, si lo que necesitamos es representar un objeto con un acabado más real, hemos de cubrirlo con una textura, como si lo empapelásemos. Tenemos que proyectar una imagen 2D sobre la superficie de nuestro modelo 3D. Es lo que se conoce como **UV mapping**.

Para realizar esto hemos de “cortar” la superficie de nuestro modelo 3D y extenderla sobre un plano que será la imágen 2D que acabaremos proyectando. Esta técnica se llama **unwrapping**.

Materiales y texturas



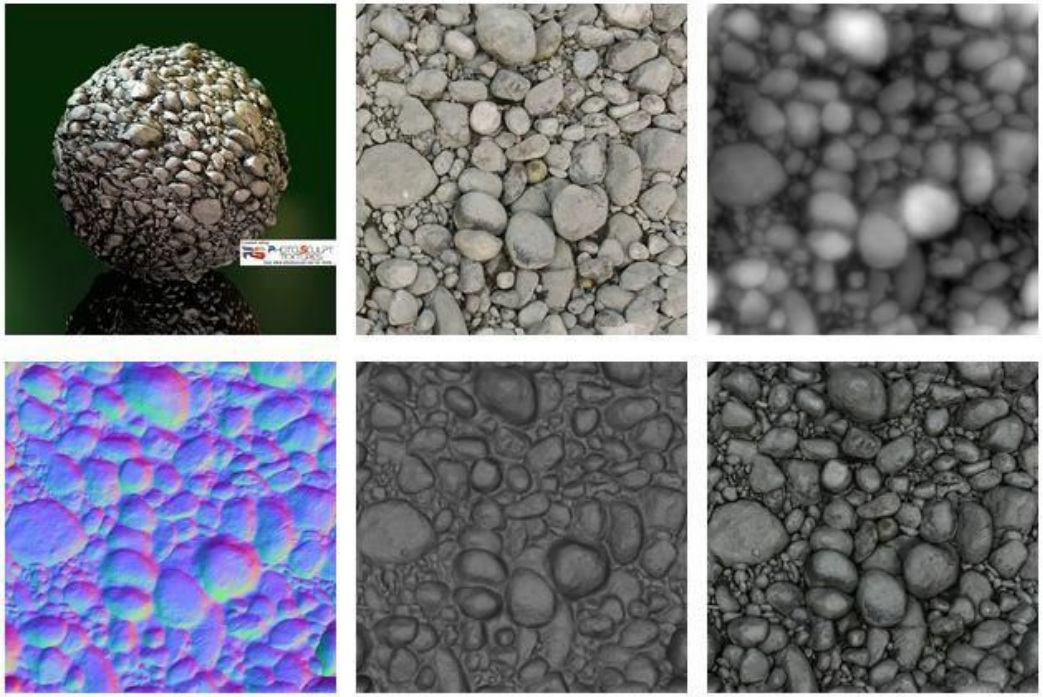
Materiales y texturas



Materiales y texturas

Utilizando diferentes mapas de textura junto al **UV mapping**, podemos obtener detalles en los objetos sin tener que crearlos en la malla.

Algunos de los mapas que se suelen utilizar son: **Albedo**, **Normal**, **Bump**, **Displacement** u **Occlusion**.



Shaders (Sombreadores)

Un shader es un algoritmo (o script) que se aplica a un objeto y que calcula su apariencia en función de las luces que hemos colocado en la escena y de los materiales que le hemos aplicado.

Un objeto puede tener diferentes apariencias utilizando diferentes shaders.

Físicas y colisiones

Física y colisiones

Una de las acciones más comunes en los videojuegos consiste en la detección de **colisiones** entre objetos y reaccionar a estas colisiones utilizando el motor físico o las acciones indicadas por un script.

Por ejemplo, cuando colisionan dos objetos, podemos emitir un sonido, hacer que desaparezca uno de los objetos o que ambos salgan despedidos en función de la fuerza de la colisión.

Para detectar colisiones, los objetos deben tener un componente conocido como **collider** que es una forma geométrica (3D o 2D) que lo envuelve.

Física y colisiones

La geometría de los colliders debe ser mucho más simplificada que la del objeto para optimizar al máximo los cálculos, ya que la detección de colisiones es una operación que utiliza mucha potencia de cálculo.

Podemos realizar acciones diferentes al iniciar la colisión, mientras se mantiene y cuando finaliza.

Física y colisiones

Para que un objeto se vea afectado por el motor físico, hemos de añadirle un componente que determine sus propiedades físicas, tales como su masa o su respuesta a la fricción.

En Unity ese componente es el **Rigidbody**.

A medida que aumenta la potencia de procesamiento de los dispositivos, la simulación que se obtiene utilizando los motores físicos es cada vez más real, permitiéndonos generar incluso movimientos de los objetos por la acción del viento o rebotes realistas en función de la masa de un objeto.

Juegos 2D y 3D

Juegos 2D y 3D

3D completo

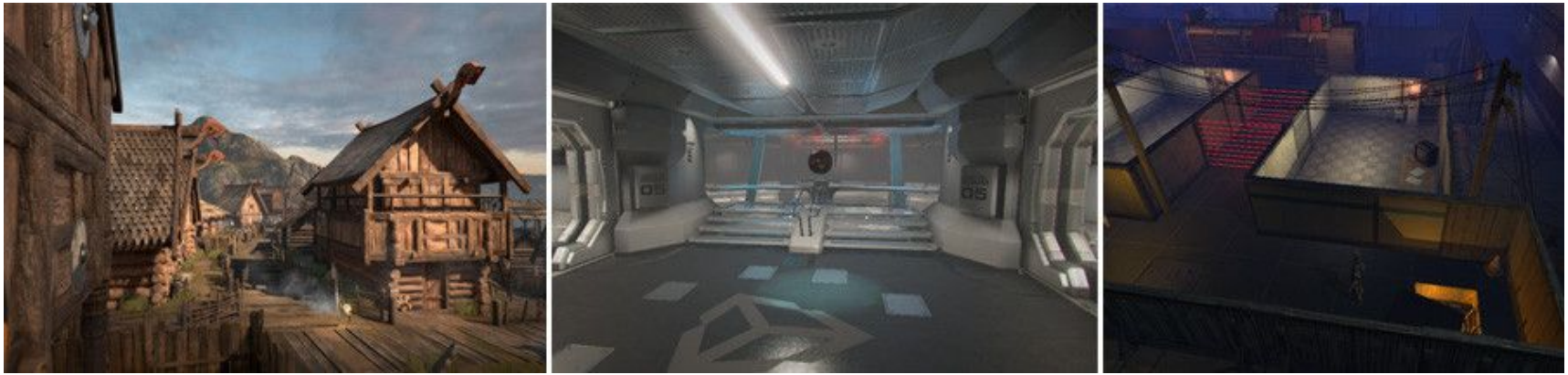
Son juegos que hacen uso de **geometría de tres dimensiones** con **materiales y texturas** renderizadas en las superficies de estos objetos para hacerlos parecer como entornos sólidos, personajes y objetos que componen su juego del mundo.

La cámara puede moverse alrededor de la escena libremente, con luces y sombras emitiéndose de manera realista en el mundo.

Normalmente renderizan la escena utilizando **perspectiva**, por lo que los objetos aparecen más grande en la pantalla a medida que se acercan a la cámara (como en la vida real).

Juegos 2D y 3D

3D completo



Para todos los juegos que encajen en esta descripción, escogeremos el **modo 3D**.

Juegos 2D y 3D

3D ortográfico

A veces los juegos utilizan **geometría 3D**, pero utilizan una **cámara ortográfica** en vez de una perspectiva. Esto es una técnica común utilizada en juegos que dan una vista panorámica de la acción, y a veces se llama “2.5D”. El ejemplo más clásico es el Crossy Road.

En este caso utilizaremos el editor en modo 3D, ya que aunque no haya una perspectiva, vamos a trabajar con modelos 3D y assets.

Juegos 2D y 3D

3D ortográfico

Para todos los juegos que encajen en esta descripción, escogeremos el **modo 3D,** pero con la **vista ortográfica**.

Juegos 2D y 3D

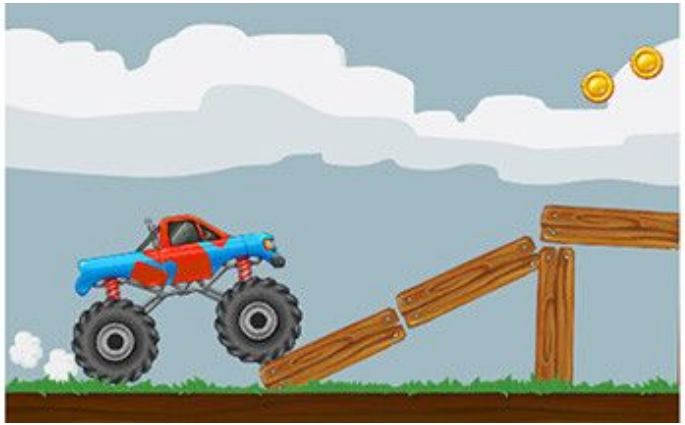
2D completo

Muchos otros juegos utilizan gráficas planas, a veces llamadas **sprites**, las cuales no tienen geometría en tres dimensiones en absoluto.

Estas son dibujadas a la pantalla como **imágenes planas**, y la cámara del juego no tiene perspectiva.

Juegos 2D y 3D

2D completo



Para este tipo de juego, usaremos el **modo 2D**.

Juegos 2D y 3D

Gameplay 2D con gráficos 3D

Algunos juegos 3D utilizan geometría 3D para el entorno y personajes, pero se limita el gameplay a dos dimensiones.

Por ejemplo, la cámara puede mostrarse como una “vista de desplazamiento lateral” y el jugador puede solamente moverse en dos dimensiones, pero el juego todavía utiliza modelos 3D para los obstáculos y un perspectiva 3D para la cámara.

Para estos juegos, el efecto 3D puede servir como algo de estilo en vez de un propósito funcional. Este tipo de juego también es a veces referido como “2.5D”.

Juegos 2D y 3D

Gameplay 2D con gráficas 3D

Aunque el gameplay es en 2D, trabajaremos con modelos 3D para construir el juego por lo que debemos escoger el **modo 3D**.

Juegos 2D y 3D

Gameplay 3D y gráficos 2D, con una cámara en perspectiva

Este es otro estilo popular de un juego 2D, utilizando gráficos 2D pero con una cámara de perspectiva para obtener un efecto de scroll parallax.

Este es un estilo de escena de “teatro de cartón”, donde todos los gráficos son planos, pero arreglados en diferentes distancias de la cámara.

Juegos 2D y 3D

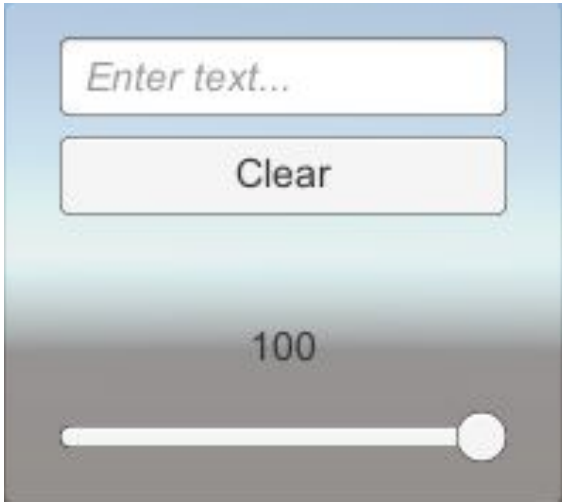
Gameplay 3D y gráficas 2D, con una cámara en perspectiva

Utilizaremos el **modo 2D**, aunque con la cámara en perspectiva.

Visualización de usuario

GUI (Graphical User Interface)

El GUI son pantallas o indicaciones visuales con las que el usuario puede interactuar con el fin de controlar el juego o aplicación. Esto incluye botones, paneles, barras de progreso, menús, y otros componentes visuales.



Canvas

Área donde se dibujan todos los elementos de la UI.

UI Elements

Elementos como texto, imágenes, botones o sliders que se utilizan para crear la interacción con el usuario.

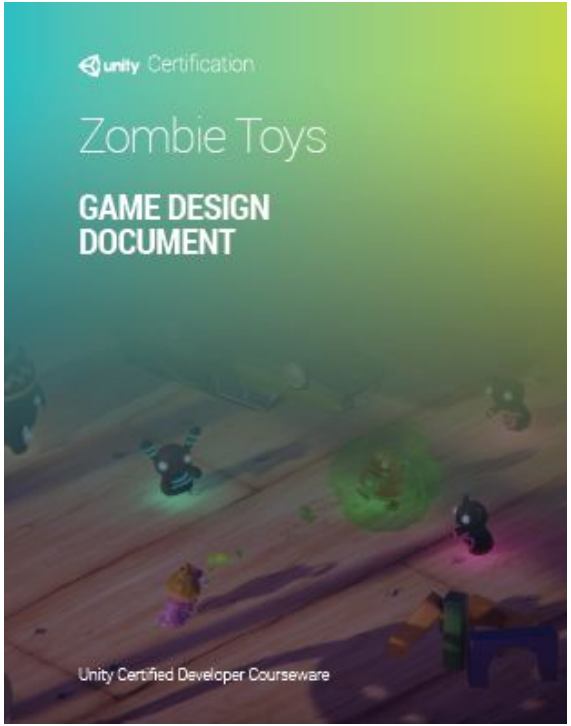
Documentos de diseño

GDD (Game Design Document)

Es un documento de diseño hecho a medida para el equipo de desarrollo.

Contiene toda la información que el equipo de desarrollo debe conocer, desde cómo se juega hasta su atmósfera, historia y la experiencia que se intenta crear. Puede tener unas pocas páginas o varios cientos.

+info



TDD (Technical Design Document)

Es una guía esencial para los programadores y otros miembros del equipo técnico, asegurando que todos trabajen con la misma visión y entendimiento de los aspectos técnicos del proyecto.

Este documento se enfoca en los aspectos técnicos del juego, desde el hardware en el que se ejecutará hasta cómo deben construirse las clases y la arquitectura del programa.