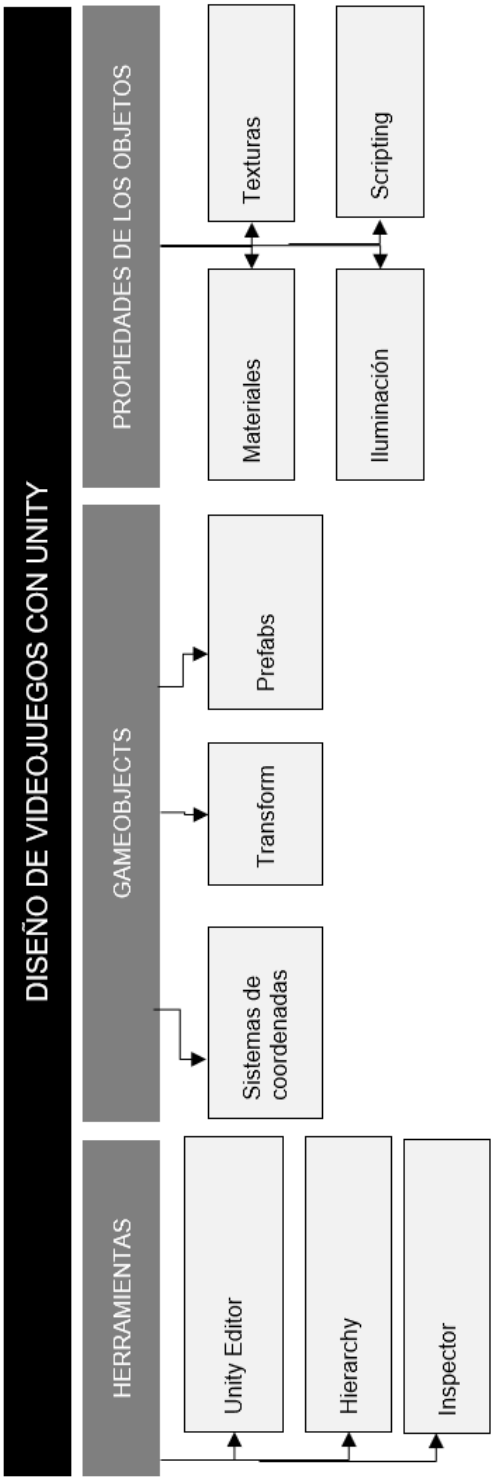
Programación multimedia y dispositivos móviles

Diseño de videojuegos con Unity

Índice

Esquema


DISEÑO DE VIDEOJUEGOS CON UNITY
HERRAMIENTAS
Unity Editor
Hierarchy
Inspector
GAMEOBJECTS
Sistemas de coordenadas
Transform
Prefabs
PROPIEDADES DE LOS OBJETOS
Materiales
Iluminación
Texturas
Scripting

Material de estudio

9.1. Introducción y objetivos

En Unity, los GameObjects son los elementos básicos que componen cualquier escena. Sin embargo, por sí mismos no tienen ninguna funcionalidad ni propiedades más allá de su posición en el espacio. Es mediante la incorporación de componentes que los GameObjects adquieren comportamientos, características y funcionalidades específicas. Estos componentes son módulos que se pueden agregar, personalizar y combinar para definir cómo interactúan los objetos con el entorno y con otros elementos del juego.

Desde componentes visuales como Renderers y Materiales, hasta elementos de interacción física como Colliders y Rigidbodies, cada componente cumple un propósito único dentro del motor de Unity. Además, los desarrolladores tienen la posibilidad de añadir scripts personalizados, ampliando las capacidades de los GameObjects y permitiendo la creación de experiencias interactivas y dinámicas.

Este tema busca explorar en detalle el papel fundamental de los componentes en Unity, cómo se estructuran y su relación directa con los GameObjects, así como las mejores prácticas para integrarlos de manera efectiva en el desarrollo de juegos.
Los objetivos que se persiguen en el siguiente tema son:

- Comprender la función de los componentes.
- Explorar los componentes predeterminados.
- Saber usar los principales tipos de componentes que Unity ofrece, como el Transform, los Renderers, Colliders, Rigidbodies y luces, entre otros.
- Relacionar componentes con jerarquías y transformaciones.
- Fomentar la experimentación creativa.

9.2. Fases de desarrollo

El desarrollo de un videojuego en Unity implica varias fases que ayudan a organizar y estructurar el proceso creativo y técnico. Estas fases pueden variar dependiendo del tamaño del equipo, el alcance del proyecto y los objetivos específicos, pero generalmente incluyen las siguientes etapas.

1. Conceptualización

En esta etapa, se define la idea central del juego y sus objetivos principales.

- Definición del concepto: ¿De qué trata el juego? ¿Cuál es su género? ¿Qué lo hace único?
- Público objetivo: Identificación del público al que va dirigido.
- Creación de un documento de diseño: Se elabora el Game Design Document (GDD), que describe mecánicas, narrativa, arte y otros aspectos clave.
- Prototipo en papel: Esbozos de niveles, mecánicas y otros elementos para validar ideas rápidamente.

2. Preproducción

Aquí se realiza la planificación del proyecto y se crean los cimientos técnicos y artísticos.

- Prototipado inicial: Se desarrolla un prototipo básico en Unity para probar mecánicas clave, usando assets temporales (placeholders).
- Planificación del proyecto: Creación de un cronograma y división del trabajo en sprints (si se usa una metodología ágil como Scrum).
- Herramientas y tecnología: Selección de los paquetes de Unity, plugins y tecnologías necesarias (por ejemplo, sistemas de partículas, shaders o inteligencia artificial).
- Estilo artístico: Desarrollo de bocetos y modelos iniciales para definir el estilo visual.

3. Producción

La fase más extensa, donde se crea el contenido principal del juego.

- Desarrollo de mecánicas: Programación de las funciones principales, como movimiento, interacción y sistemas de juego.
- Creación de arte: Diseño de personajes, entornos, animaciones, efectos visuales y otros recursos gráficos.
- Diseño de niveles: Uso de Unity para construir niveles jugables y equilibrados.
- Sonido: Incorporación de música, efectos de sonido y voces.
- Pruebas y ajustes: Realización de pruebas continuas para encontrar errores y optimizar las mecánicas y el rendimiento.

4. Pruebas

Una etapa crucial para garantizar la calidad del juego.

- Pruebas funcionales: Verificación de que todas las mecánicas y sistemas funcionan correctamente.
- Pruebas de usabilidad: Evaluación de la experiencia del usuario para asegurar que sea intuitiva y agradable.
- Corrección de errores: Reparación de bugs y problemas identificados durante las pruebas.
- Optimización: Ajustes en el rendimiento para garantizar que el juego funcione bien en diferentes dispositivos.

5. Lanzamiento

El momento en que el juego se hace público.

- Preparación para el lanzamiento: Creación de builds finales para las plataformas objetivo (PC, consola, móvil).
- Marketing: Campañas para promocionar el juego, como tráileres, redes sociales y eventos.

- Distribución: Publicación en tiendas digitales como Steam, Google Play, Apple Store o consolas específicas.

6. Postlanzamiento

El desarrollo no termina con el lanzamiento. En esta fase se mantiene y mejora el juego.

- Corrección de errores: Resolución de problemas reportados por los jugadores.
- Actualizaciones y contenido adicional: Nuevas funciones, niveles o expansiones para mantener el interés del público.
- Soporte a la comunidad: Interacción con los jugadores para recibir retroalimentación y resolver dudas.

9.3. Herramientas de trabajo

Unity es una plataforma muy completa que ofrece diversas herramientas y funcionalidades para crear videojuegos y aplicaciones interactivas. Estas herramientas están diseñadas para facilitar las tareas de diseño, programación, animación, prueba y publicación. A continuación, se describen las principales herramientas de trabajo en Unity.

1. Interfaz del Editor de Unity

El editor principal de Unity es donde se realiza la mayor parte del trabajo. Algunas de sus secciones clave incluyen:

- Scene View: Vista principal donde se construyen y editan los niveles del juego. Permite posicionar, rotar y escalar objetos en el mundo 3D o 2D.
- Game View: Muestra cómo se verá el juego durante su ejecución, simulando la experiencia del jugador.

- Hierarchy: Lista jerárquica de todos los objetos en la escena actual, organizados en un sistema de árbol.
- Inspector: Panel que muestra y permite modificar las propiedades del objeto seleccionado, como transformaciones, componentes o scripts.
- Project: Biblioteca de recursos del proyecto (modelos, scripts, sonidos, texturas, etc.).
- Console: Panel para mostrar mensajes, advertencias y errores del sistema

2. Herramientas de diseño

- Tilemap: Para crear y editar niveles 2D mediante un sistema de mosaicos.
- ProBuilder: Permite diseñar y modificar geometrías 3D directamente en Unity, ideal para prototipos rápidos de niveles.
- Terrain Editor: Herramienta para diseñar terrenos grandes, con funciones para esculpir, pintar texturas y añadir vegetación.
- NavMesh: Sistema de navegación para crear caminos y áreas transitables por NPCs o personajes controlados por IA.
- Shader Graph: Editor visual para crear shaders personalizados sin necesidad de escribir código.

3. Herramientas de programación

- Scripts: Se escriben en C# y se integran fácilmente en objetos del juego. Unity usa Visual Studio o Rider como entornos de desarrollo integrados (IDE).
- MonoBehaviour: Clase base para la creación de scripts que permiten manejar eventos y comportamientos en el juego.
- Unity Events y Delegates: Sistemas para manejar interacciones y eventos personalizados entre objetos.
- Debugger: Herramienta para depurar scripts directamente desde el editor o IDE.

4. Animación

- Animator: Sistema de animación basado en gráficos de estado, ideal para gestionar transiciones y comportamientos complejos.

- Animation Timeline: Herramienta para editar y previsualizar animaciones cuadro por cuadro.
- Rigging y Mecanim: Integración con sistemas de rigging y controladores de animación, especialmente útiles para personajes 3D.
- Cinemachine: Sistema de cámaras avanzado para crear transiciones suaves, movimientos dinámicos y efectos cinematográficos.

5. Pruebas y optimización

- Profiler: Muestra datos sobre el rendimiento del juego, como uso de CPU, GPU, memoria y red.
- Frame Debugger: Permite inspeccionar los procesos de renderizado cuadro a cuadro.
- Physics Debugger: Visualiza colisiones, interacciones físicas y fuerzas en la escena.
- Device Simulator: Simula cómo se verá y funcionará el juego en diferentes dispositivos y resoluciones.

6. Publicación y distribución

- Build Settings: Configura los parámetros para exportar el juego a diferentes plataformas (Windows, macOS, Android, iOS, consolas, WebGL, etc.).
- Unity Asset Store: Mercado para adquirir recursos como modelos, texturas, sonidos, scripts y herramientas.
- Cloud Build: Sistema automatizado para compilar proyectos en la nube y reducir el tiempo de integración continua.
- Unity Analytics y Ads: Herramientas para implementar análisis de datos del jugador y monetización mediante anuncios.

9.4. Escenas

Las escenas en Unity son un componente esencial en el desarrollo de juegos y aplicaciones interactivas. Una escena es un entorno virtual que contiene todos los

elementos necesarios para una parte específica del juego, como objetos, iluminación, cámaras y scripts. Se utilizan para dividir el juego en secciones manejables, como niveles, menús o pantallas de inicio. Pueden ser usadas para crear un menú principal, niveles individuales y cualquier otra cosa. Para utilizar las funciones de escenas en Unity se utiliza la clase SceneManager.

Características principales de las Escenas en Unity

1. Estructura Jerárquica: Cada escena está compuesta por una jerarquía de objetos llamada Hierarchy. Los objetos de la escena se organizan en un sistema de árbol, donde los objetos principales (padres) pueden tener objetos secundarios (hijos).

2. Archivo de Escena: Cada escena se guarda como un archivo con extensión .unity. Este archivo contiene toda la información sobre los objetos y configuraciones de la escena.

3. Independencia: Las escenas son independientes entre sí. Esto permite cargar y descargar escenas según sea necesario, ahorrando recursos y mejorando el rendimiento.

4. Uso de Múltiples Escenas: Unity permite usar varias escenas simultáneamente gracias al Scene Manager. Esto es útil para dividir contenido grande, como niveles y elementos persistentes (por ejemplo, un HUD o música de fondo).

Elementos Principales en una Escena

Objetos del Juego (GameObjects): Los GameObjects son los bloques básicos de construcción en una escena. Pueden ser objetos 3D, 2D, luces, cámaras, vacíos (vacuum) o prefabs. Se les pueden añadir componentes (scripts, físicas, materiales, etc.) para definir su comportamiento.

1. Cámara: Cada escena necesita al menos una cámara para que el jugador pueda ver el contenido. La cámara define el ángulo, perspectiva y campo de visión de la escena.

2. Iluminación: Incluye luces para dar realismo al entorno (como luces direccionales, puntuales o de área). Unity permite configurar la iluminación global y el horneado de luces (light baking) para optimizar el rendimiento.

3. Terrenos y Entornos: Terrains y elementos del entorno, como árboles, rocas o edificios, se pueden añadir para construir niveles realistas.

4. Scripts: Añaden lógica y comportamientos específicos a los objetos de la escena.

Gestión de Escenas con el Scene Manager

Unity utiliza el Scene Manager para controlar cómo se cargan y descargan las escenas durante el juego. Algunas funciones clave del Scene Manager son:

1. Carga de Escenas: Puedes cargar una nueva escena reemplazando la actual o añadiéndola encima de la existente. Métodos comunes:

- SceneManager.LoadScene("SceneName") - Carga una escena reemplazando la actual utilizando su nombre o su indice.
- SceneManager.LoadScene("SceneName", LoadSceneMode.Additive) - Carga una escena adicionalmente sin reemplazar la actual.

2. Cambio de Escenas: Es común cambiar de escena al terminar un nivel o al acceder a un menú. Esto se gestiona a través de scripts.

3. Persistencia de Datos entre Escenas: El método DontDestroyOnLoad se utiliza para mantener objetos persistentes al cambiar de escena (por ejemplo, música o estadísticas del jugador).

9.5. GameObjects y cámara

Un GameObject es una entidad básica en Unity que actúa como un contenedor vacío. Por sí solo, no tiene ninguna funcionalidad ni apariencia, pero se convierte en algo útil cuando se le agregan componentes. Los GameObjects pueden representar cualquier cosa en el juego, desde personajes y enemigos hasta luces, cámaras y elementos del entorno.

Cada GameObject tiene al menos los siguientes elementos:

1. Transform:

Es el único componente que todos los GameObjects tienen por defecto. Define la posición, rotación y escala del GameObject en el espacio 3D o 2D.

2. Componentes:

Los componentes son lo que define el comportamiento y las propiedades de un GameObject. Por ejemplo:

- Un Mesh Renderer permite que el objeto sea visible en la escena.
- Un Collider define su área física para interacciones (Box Collider, Sphere Collider y Capsule Collider, Mesh Collider).
- Un Rigidbody le da propiedades físicas como gravedad y colisiones.
- Scripts: Añadir comportamientos personalizados, interactuar con otros GameObjects o controlar sus transformaciones y componentes

Se pueden agregar, modificar o eliminar estos componentes y muchos más desde el Inspector en Unity.

Los GameObjects pueden **activarse o desactivarse** desde el Inspector o mediante scripts (`gameObject.SetActive(true/false)`). Cuando están desactivados, no participan en la escena ni en los cálculos del juego.

Jerarquías de GameObjects

Los GameObjects pueden estar organizados en jerarquías, donde un GameObject puede ser "padre" de otros. Las Jerarquías de GameObjects en Unity son una

característica fundamental que permite organizar y estructurar los objetos de una escena de forma lógica y funcional. A través de jerarquías, puedes agrupar GameObjects y establecer relaciones de dependencia entre ellos, lo que facilita su manipulación, control y organización.

Una jerarquía de GameObjects en Unity tiene una estructura tipo árbol en la que un GameObject puede ser un "padre" de otros GameObjects llamados "hijos". Cada GameObject hijo puede tener a su vez otros hijos, creando una jerarquía de múltiples niveles.

Esto tiene varias implicaciones:

1. Transformación heredada:
Los GameObjects hijos heredan las transformaciones del padre. Esto significa que, si mueves, rotas o escalas al GameObject padre, esos cambios afectarán automáticamente a los hijos.
Los hijos también pueden tener sus propias transformaciones locales, que son relativas a la posición, rotación y escala del padre.

2. Organización lógica:
Las jerarquías te permiten agrupar GameObjects que están relacionados. Por ejemplo, un coche puede ser el padre, y las ruedas, puertas y luces pueden ser sus hijos.

3. Manipulación más sencilla:
Al trabajar con jerarquías, puedes mover, rotar o escalar un grupo completo simplemente manipulando el GameObject padre.

4. Activación y desactivación grupal:
Si desactivas un GameObject padre, todos sus hijos también se desactivan automáticamente. Esto es útil para ocultar o desactivar grupos de objetos relacionados en la escena.

Prefabs

Un Prefab (abreviatura de prefabricated object) es una plantilla de un GameObject con todos sus componentes, propiedades y jerarquías asociados, que se guarda como un recurso en el proyecto. Puedes instanciar este Prefab varias veces en diferentes escenas o en la misma escena, y mantener todos los objetos sincronizados con los cambios realizados en el Prefab original. Sus características principales son:

- Reutilización: Los Prefabs son ideales para crear múltiples copias de un objeto sin necesidad de repetir su configuración.
- Actualización en tiempo real: Si modificas el Prefab original, todos los objetos que lo utilizan (instancias del Prefab) se actualizan automáticamente.
- Instanciación dinámica: Puedes instanciar Prefabs mediante scripting para generar objetos en tiempo de ejecución.
- Jerarquías: Los Prefabs pueden incluir jerarquías completas de GameObjects, lo que permite que objetos complejos sean tratados como una sola unidad.

Cuando un GameObject forma parte de un Prefab, sus componentes y propiedades pueden gestionarse de diferentes maneras:

- Propiedades vinculadas al Prefab: Cambios realizados en el Prefab original afectarán automáticamente a todas las instancias.
- Propiedades personalizadas en instancias: Puedes personalizar componentes o propiedades específicas de una instancia sin afectar el Prefab original. Estas modificaciones se llaman overrides.
- Revertir cambios: Si una instancia tiene cambios que ya no deseas, puedes revertirla a su estado original desde el menú de Inspector.

Unity ofrece diferentes tipos de Prefabs para cubrir diversas necesidades:

- Prefabs estándar: Son plantillas de GameObjects comunes.
- Variants (Variantes): Son Prefabs que heredan de un Prefab base pero pueden tener diferencias específicas. Esto es útil si necesitas diferentes versiones de un

objeto que comparten una base común. Ejemplo: Un Prefab base de Enemy con variantes como FastEnemy y StrongEnemy.

- Nested Prefabs (Prefabs anidados): Puedes incluir Prefabs dentro de otros Prefabs para estructurar objetos complejos.

El uso de los Prefabs en Unity nos aporta una serie de ventajas:

- Eficiencia en el desarrollo: Puedes ahorrar tiempo al reutilizar Prefabs en lugar de recrear objetos desde cero.
- Consistencia: Garantizas que todos los objetos similares en el juego mantengan una configuración uniforme.
- Gestión dinámica: Los Prefabs son esenciales para sistemas como generación procedural, manejo de enemigos, proyectiles y más.
- Facilidad de iteración: Cambiar el diseño o comportamiento de un objeto en el juego es sencillo, ya que solo necesitas modificar el Prefab.

La cámara en Unity

La cámara es una herramienta indispensable en Unity que define cómo los jugadores interactúan visualmente con el mundo del juego. Su configuración, manejo mediante scripts y la integración de efectos visuales pueden transformar completamente la experiencia del jugador. La cámara en Unity es un GameObject especial que actúa como el ojo del jugador. Renderiza la escena desde su posición y orientación en el espacio 3D o 2D, mostrando lo que se ve en la pantalla.

Unity permite tener múltiples cámaras en una escena, cada una con configuraciones específicas para diferentes propósitos (por ejemplo, vista en primera persona, vistas de seguridad, mapas pequeños, etc.).

Componentes principales de una cámara

1. Transform: Al igual que cualquier GameObject, la cámara tiene un Transform que define su posición, rotación y escala en el espacio.

2. Camera Component: Es el componente principal que convierte un GameObject en una cámara funcional. Este componente incluye múltiples configuraciones:

- Field of View (FOV): Define el ángulo de visión en perspectiva. Un valor más alto da un efecto de "gran angular", mientras que un valor más bajo se asemeja a un teleobjetivo.

- Clipping Planes:
    - Near: Distancia mínima a la que la cámara puede renderizar.
    - Far: Distancia máxima que la cámara puede renderizar.

- Clear Flags: Define lo que la cámara debe renderizar como fondo.
    - Skybox: Renderiza el skybox asignado.
    - Solid Color: Renderiza un color sólido como fondo.
    - Depth Only: Renderiza solo objetos sin un fondo visible.
- Culling Mask: Permite seleccionar qué capas de objetos serán renderizadas por la cámara.

Projection: Perspective: Simula cómo vemos en el mundo real, con objetos más lejanos viéndose más pequeños.

Orthographic: Renderiza los objetos sin perspectiva, útil para juegos en 2D o vistas técnicas.

3. Audio Listener: Todas las cámaras tienen un Audio Listener por defecto, que actúa como el oído del jugador. Solo puede haber un Audio Listener activo en la escena.

Configuración básica de la cámara y tipos de cámara

Cuando creas un nuevo proyecto, Unity incluye automáticamente una cámara predeterminada. Puedes ajustar sus configuraciones desde el Inspector para adaptarlas a tus necesidades. Algunos ajustes básicos incluyen:

- Cambiar el FOV para ampliar o reducir el ángulo de visión.
- Configurar el plano cercano y lejano (clipping planes) para optimizar el rendimiento.

- Asignar un Skybox personalizado o establecer un color de fondo sólido.
- Ajustar la resolución y proporción de la vista desde el menú de la ventana de juego.

Tipos de cámaras comunes

- Cámara en primera persona: Sigue la perspectiva del jugador como si estuviera viendo desde sus ojos. Se utiliza en juegos FPS (First-Person Shooter).
- Cámara en tercera persona: Muestra al personaje desde un ángulo detrás o sobre él.
- Suele estar vinculada al movimiento del personaje, con ajustes para seguir su posición.
- Cámara fija: Permanece en una posición fija, útil para juegos con perspectivas isométricas o cámaras de seguridad.
- Cámara cinemática: Usada para crear escenas de corte o animaciones en las que la cámara sigue una ruta predefinida.

Unity permite tener varias cámaras en una escena. Puedes usarlas de diferentes maneras:
Dividir la pantalla: Utiliza cámaras para mostrar diferentes perspectivas al mismo tiempo.

- Cámaras de UI: Dedica una cámara exclusivamente a renderizar elementos de la interfaz de usuario.
- Cámaras de efectos: Agrega cámaras secundarias para renderizar efectos específicos como reflejos o texturas especiales.
- Puedes gestionar qué cámara es la principal y el orden en que se renderizan utilizando la propiedad Depth de cada cámara.

Las cámaras pueden ser controladas mediante scripts para crear efectos dinámicos, como seguimiento del jugador o movimientos cinemáticos.

9.6. Sistemas de coordenadas, transformaciones

El sistema de coordenadas y el componente Transform en Unity son fundamentales para posicionar, rotar y escalar objetos dentro de una escena. Estos conceptos son esenciales para comprender cómo interactúan los objetos en un entorno 3D o 2D.

Sistema de coordenadas en Unity

Unity utiliza un sistema de coordenadas cartesianas para definir la posición de los objetos en el espacio. Este sistema es tridimensional (3D), aunque también funciona en 2D.

Ejes principales

- Eje X: Representa la dirección horizontal.
    - Valores positivos: Hacia la derecha.
    - Valores negativos: Hacia la izquierda.
- Eje Y: Representa la dirección vertical.
    - Valores positivos: Hacia arriba.
    - Valores negativos: Hacia abajo.

- Eje Z: Representa la profundidad.
    - Valores positivos: Hacia adelante (lejos de la cámara).
    - Valores negativos: Hacia atrás (cerca de la cámara).

En una escena 2D, generalmente solo se utilizan los ejes X e Y, mientras que el eje Z se utiliza para el orden de renderizado o la separación de capas.

Tipos de sistemas de coordenadas

Unity trabaja con dos sistemas de coordenadas principales:

1. Coordenadas globales (World Coordinates):

- Son absolutas y representan la posición, rotación y escala de un objeto respecto al origen del mundo (0, 0, 0).
- Se accede mediante transform.position.

2. Coordenadas locales (Local Coordinates):

- Son relativas a la posición, rotación y escala del GameObject padre.

- Se accede mediante transform.localPosition.

Por ejemplo:
Un objeto en (2, 1, 0) en coordenadas locales, con un padre en (5, 5, 0) en coordenadas globales, tendrá una posición global de (7, 6, 0).

El componente Transform

El Transform es el componente que define la posición, rotación y escala de un GameObject en el espacio. Es un componente que todos los GameObjects tienen por defecto y que no se puede eliminar.

Propiedades principales del Transform

1. Position (posición):

Define dónde se encuentra el GameObject en el espacio global o local.
Se accede mediante:

- transform.position (global).
- transform.localPosition (local).

2. Rotation (rotación):

- Define la orientación del GameObject.
- Se mide en ángulos de Euler (grados) o cuaterniones.
- Se accede mediante:
    - transform.rotation (global, como un quaternion).
    - transform.localRotation (local, como un quaternion).
    - transform.eulerAngles (global, como ángulos de Euler).

3. Scale (escala):

- Define el tamaño del GameObject en cada eje.
- Se accede mediante: transform.localScale.
- La escala afecta a todos los hijos del objeto, ya que se hereda jerárquicamente.

4. Parent (padre):

- Define el GameObject padre al que está vinculado.
- Se accede mediante transform.parent.

5. Hierarchy (jerarquía):

Puedes acceder a los hijos del GameObject usando transform.GetChild(index) y al número de hijos con transform.childCount.

Unity ofrece varias funciones y propiedades para manipular la posición, rotación y escala de los objetos. Aquí tienes las más comunes:

- Mover un objeto

```
// Cambiar directamente la posición global
transform.position = new Vector3(1, 2, 3);
// Mover de forma relativa
transform.Translate(Vector3.forward * Time.deltaTime);
```

- Rotar un objeto

```
// Cambiar la rotación directamente
transform.rotation = Quaternion.Euler(0, 90, 0);
// Rotar de forma incremental
transform.Rotate(Vector3.up * 45 * Time.deltaTime);
```

- Cambiar la escala

```
// Escalar uniformemente
transform.localScale = new Vector3(2, 2, 2);
// Cambiar solo un eje
transform.localScale += new Vector3(0, 1, 0) * Time.deltaTime;
```

Cuando trabajas con transformaciones, es importante distinguir entre espacio global y espacio local:

- Global: Las transformaciones se aplican respecto al sistema de coordenadas del mundo. Ejemplo: Mover un objeto hacia arriba en el eje global Y.

- Local: Las transformaciones se aplican respecto al sistema de coordenadas del GameObject. Ejemplo: Mover un objeto hacia adelante en su eje Z local, sin importar cómo esté rotado.

Las jerarquías afectan directamente al componente Transform:

- Si un objeto tiene un padre, sus transformaciones locales son relativas a las del padre.
- Si se elimina el padre de un objeto, sus transformaciones locales se convierten en globales.

Unity proporciona varias funciones prácticas para manipular transformaciones:

- LookAt: Hace que un GameObject mire hacia otro.

```
transform.LookAt(target.position);
```

- SetParent: Cambia el padre de un objeto en tiempo de ejecución.

```
transform.SetParent(newParent);
```

- DetachChildren: Desvincula todos los hijos del GameObject actual.

```
transform.DetachChildren();
```

- TransformDirection y InverseTransformDirection: Convierte direcciones de local a global y viceversa.

```
Vector3 globalDirection = transform.TransformDirection(Vector3.forward);
```

- TransformPoint y InverseTransformPoint: Convierte posiciones de local a global y viceversa.

```
Vector3 globalPoint = transform.TransformPoint(new Vector3(1, 1, 1));
```

9.7. Propiedades de los objetos: luz, texturas, materiales, iluminación

En Unity, los GameObjects pueden personalizarse y extenderse mediante diversos componentes que definen sus propiedades y comportamientos visuales. Estas propiedades incluyen elementos como luces, texturas, materiales e iluminación, que en conjunto dan vida a los objetos en el juego. Un componente en Unity es un script o módulo que se adjunta a un GameObject para añadirle propiedades o funcionalidades específicas. Por ejemplo:

- Un Rigidbody le da al GameObject propiedades físicas como gravedad y colisiones.
- Un Light lo convierte en una fuente de luz.
- Un Collider define su área física para detectar colisiones.

Iluminación

La iluminación es crucial para crear un entorno realista y envolvente. Unity ofrece un sistema de iluminación robusto que incluye luces dinámicas, luces estáticas, iluminación global y efectos avanzados.

1. Tipos de iluminación

- Iluminación directa: Proporcionada por las luces en la escena (Directional, Point, Spot, etc.).
- Iluminación global (Global Illumination): Simula la luz que rebota entre superficies, mejorando el realismo. Puede ser en tiempo real o precalculada (baked).

2. Características avanzadas

- Baked Lighting: La iluminación precalculada se guarda en Lightmaps, lo que mejora el rendimiento.
- Realtime Lighting: La iluminación se calcula en tiempo real, adecuada para objetos dinámicos.
- Mixed Lighting: Combina iluminación precalculada y en tiempo real.
- Ambient Occlusion: Simula sombras suaves en los bordes de los objetos.

3. Configuración de la iluminación

Se gestiona en el menú Window > Rendering > Lighting:

- Skybox Material: Define el fondo y la iluminación ambiental.
- Ambient Light: Controla el brillo general.
- Reflection Probes: Capturan reflejos para superficies brillantes.
- Lightmaps: Mapa de luz precalculado para objetos estáticos.

Luz en GameObjects

Las luces son componentes que simulan fuentes de iluminación en el mundo del juego. Pueden influir en cómo se ven los objetos al interactuar con sus materiales y texturas.

1. Tipos de luces en Unity

- Directional Light: Simula una luz infinita que emite rayos paralelos, como el sol. Ilumina toda la escena de manera uniforme.
- Point Light: Emite luz en todas las direcciones desde un punto específico. Útil para lámparas o bombillas.
- Spot Light: Proyecta un haz de luz en forma de cono. Ideal para focos o linternas.
- Area Light: Emite luz desde una superficie rectangular. Se usa principalmente en renderizado avanzado (HDRP).
- Ambient Light: No es un tipo de luz directa, pero afecta el brillo general de los objetos en la escena. Se configura en las Settings de iluminación.

2. Propiedades principales de las luces

- Color: Define el color de la luz.
- Intensity: Ajusta la fuerza de la luz.
- Range: (Para Point y Spot Lights) Define hasta qué distancia afecta la luz.
- Spot Angle: (Para Spot Lights) Controla el ángulo del cono de luz.
- Shadows: Configura si la luz proyecta sombras y de qué tipo (reales o suaves).

Materiales

Los materiales son componentes que definen cómo un objeto interactúa con la luz. Un material combina texturas y propiedades físicas para determinar su apariencia.

1. Shader:

El shader de un material define cómo procesa Unity la información del material. Hablaremos más en profundidaz en el próximo tema. Algunos shaders comunes en Unity incluyen:

- Standard Shader (universal para PBR).
- Unlit Shader (sin iluminación).
- Transparent Shader (materiales transparentes).
- Custom Shaders (creados con Shader Graph o mediante código).

2. Propiedades del material:

- Albedo: Color base o textura del material.
- Metallic: Controla el nivel de reflectividad metálica.
- Smoothness: Controla qué tan suave o rugosa es la superficie.
- Emission: Agrega un brillo autosuficiente al material.
- Transparency: Define si el material es opaco, transparente o semitransparente.
- Tiling y Offset: Ajustan cómo se repite y posiciona la textura.

Texturas

Una textura es una imagen 2D que se aplica a un material para definir el aspecto superficial de un GameObject. Las texturas pueden representar colores, patrones, detalles de superficies, entre otros. En Unity disponemos de los siguientes tipos de texturas:

- Albedo Texture: Representa el color base o el mapa de color de un material.
- Normal Map: Da la ilusión de profundidad o detalles en la superficie, sin alterar la geometría real.
- Metallic Map: Controla qué partes de un material se ven metálicas.
- Roughness/Glossiness Map: Define qué tan brillante o rugosa es la superficie.
- Height Map: Utilizado para simulaciones más detalladas de superficies rugosas.
- Emission Map: Hace que ciertas áreas de un objeto emitan luz.

Relación entre texturas, materiales e iluminación

La relación entre texturas, materiales e iluminación en Unity es clave para lograr gráficos realistas y visualmente atractivos. Estos tres elementos trabajan juntos para definir cómo se ven los objetos en una escena y cómo interactúan con la luz del entorno. La interacción entre estos elementos es esencial para determinar el aspecto final de un objeto en Unity. Veamos cómo se conectan:

1. Las texturas y su rol en los materiales

Las texturas proporcionan información visual al material, definiendo cómo se ve la superficie de un objeto.

Ejemplo:

- Albedo Map: Una textura que define el color base.
- Normal Map: Simula baches o irregularidades para que la superficie parezca más detallada sin modificar la geometría real.

Un material puede usar varias texturas simultáneamente para combinar diferentes propiedades visuales:

- Albedo (color base).
- Metallic (reflejos metálicos).
- Roughness/Glossiness (rugosidad y brillo).
- Emission (partes que emiten luz).

2. Los materiales y su interacción con la iluminación

Los materiales determinan cómo un objeto responde a la luz del entorno. Esto incluye:

- Qué tanto refleja la luz (metallic).
- Qué tan brillante o difusa es la superficie (smoothness/roughness).
- Si el objeto emite luz (emission).

Por ejemplo:

- Un material con un Metallic Map alto reflejará luces y objetos circundantes.

- Un material con Roughness alto dispersará la luz, haciendo que la superficie parezca más opaca.

3. La iluminación y su efecto sobre texturas y materiales

La iluminación influye en la apariencia de los materiales y las texturas:

- Luces dinámicas (reales) afectan cómo se proyectan sombras y reflejos en tiempo real.
- Global Illumination mejora el realismo al simular el rebote de la luz entre superficies.
- Reflection Probes generan reflejos realistas en materiales reflectantes.

Supongamos que estás estás diseñando una mesa de madera con un material detallado en Unity:

- Texturas: Usas una textura de Albedo que tiene un patrón de vetas de madera. Agregas un Normal Map para que las vetas tengan un relieve aparente. Incluyes un Roughness Map que hace que la superficie se vea más mate en ciertas áreas.

- Material: Configuras el material para que tenga un valor de Metallic bajo, ya que la madera no es reflectante. Ajustas el Smoothness para que la mesa tenga un acabado ligeramente pulido, simulando barniz.

- Iluminación: Una luz puntual se coloca sobre la mesa. Esto resalta las vetas de la madera y crea un brillo tenue en las áreas más lisas. Un Reflection Probe captura el entorno para proporcionar reflejos suaves en la superficie de la mesa.

El resultado es una mesa de madera que parece realista, con detalles visibles gracias a las texturas, propiedades físicas del material y efectos de iluminación.

9.8. Assets

En Unity, los assets son los elementos o recursos que utilizas para construir tu proyecto. Estos pueden incluir modelos 3D, texturas, materiales, sonidos, scripts, animaciones y más. Unity organiza los assets en la carpeta Assets del proyecto, y puedes administrarlos a través del Inspector y la ventana de Proyecto.

Tipos Comunes de Assets

- Modelos 3D: Formatos como .fbx, .obj o .blend que representan objetos tridimensionales.
- Texturas: Imágenes utilizadas para añadir detalles visuales a superficies, como archivos .png, .jpg, .tiff.
- Materiales: Configuraciones que determinan cómo las texturas se aplican a un objeto (propiedades de color, brillo, transparencia, etc.).
- Scripts: Archivos de código en C# que controlan el comportamiento de los objetos en tu juego.
- Audio: Clips de sonido en formatos como .wav, .mp3 o .ogg.
- Animaciones: Archivos que definen movimiento o transformaciones para personajes u objetos.
- Prefabs: Plantillas de objetos que puedes reutilizar en tu escena.
- Shaders: Programas que determinan cómo se renderizan los materiales.
- Scenes: Archivos que representan niveles o áreas específicas de tu juego.

Gestión de Assets

- Ventana de Proyecto: Aquí es donde puedes ver, organizar y buscar tus assets.
- Carpeta Assets: Es la carpeta principal donde se almacenan todos los recursos del proyecto.
- Inspector: Permite editar las propiedades de los assets seleccionados.
- Unity permite agrupar y categorizar assets en carpetas personalizadas para mantener el proyecto organizado. Además, puedes renombrar y mover assets dentro del editor, pero ten cuidado de no romper referencias entre objetos.

Importación de Assets

Puedes importar assets a tu proyecto arrastrándolos desde el sistema de archivos a la ventana de Proyecto o usando el menú Assets > Import New Asset. Unity optimiza los recursos al importarlos y permite ajustar configuraciones específicas:

- Configuración de Texturas: Resolución, tipo (albedo, normal, etc.), compresión.
- Modelos 3D: Escala, animaciones, materiales.
- Audio: Calidad, formato de compresión (PCM, Vorbis, ADPCM).

Prefabs

Los prefabs son uno de los recursos más importantes. Permiten crear plantillas reutilizables de objetos con todas sus configuraciones, componentes y jerarquías. Por ejemplo, un enemigo en un juego puede ser un prefab con su modelo 3D, comportamiento y animaciones preconfigurados. Al actualizar un prefab, todos los objetos que usan ese prefab se actualizan automáticamente.

Asset Store y Unity Package Manager

- Unity Asset Store: Una tienda integrada donde puedes descargar assets gratuitos y pagos, como modelos, scripts y herramientas.
- Unity Package Manager (UPM): Maneja paquetes oficiales y personalizados para añadir funcionalidad al proyecto.

A fondo

Unity Assets Store

https://assetstore.unity.com/

Portal web de recursos. (2024)

Portal web oficial para la adquisición de Assets en Unity.

Docs Unity

https://docs.unity.com/

Documentación en línea. (2024)

Documentación oficial sobre el motor de videojuegos Unity.

Unity Learn

https://learn.unity.com/

Documentación en línea. (2024)

Documentación oficial sobre sobre el programa y los recursos de aprendizaje de Unity.

Entrenamientos

Entrenamiento 1

- Desarrollar el movimiento de un objeto 3D en una escena.

- Desarrollo paso a paso y solución: https://www.youtube.com/watch?v=nmGM0SZW8ac

Entrenamiento 2

- Desarrollar un UI-HUD Canvas en Unity.

- Desarrollo paso a paso y solución: https://www.youtube.com/watch?v=rXMeDUJGsOo

Entrenamiento 3

- Explorar el kit de desarrollo 2D de Unity.

- Desarrollo paso a paso y solución: https://learn.unity.com/tutorial/video-guia-para-el-kit-de-juego-en-2d#

Entrenamiento 4

- Desarrollo de juego Roll a Ball: Utilizarás el Editor de Unity y sus capacidades integradas para configurar un entorno de juego sencillo. Escribirás tus propios scripts personalizados para crear la funcionalidad del juego. Crearás una interfaz de usuario básica para mejorar la experiencia de juego.

- Desarrollo paso a paso y solución: https://learn.unity.com/project/roll-a-ball?language=en&utm_source=learn_recommendation

Entrenamiento 5

- Desarrollar una IA muy sencilla que persiga al jugador.

- Desarrollo paso a paso y solución: https://www.youtube.com/watch?v=llMrzq1r61w