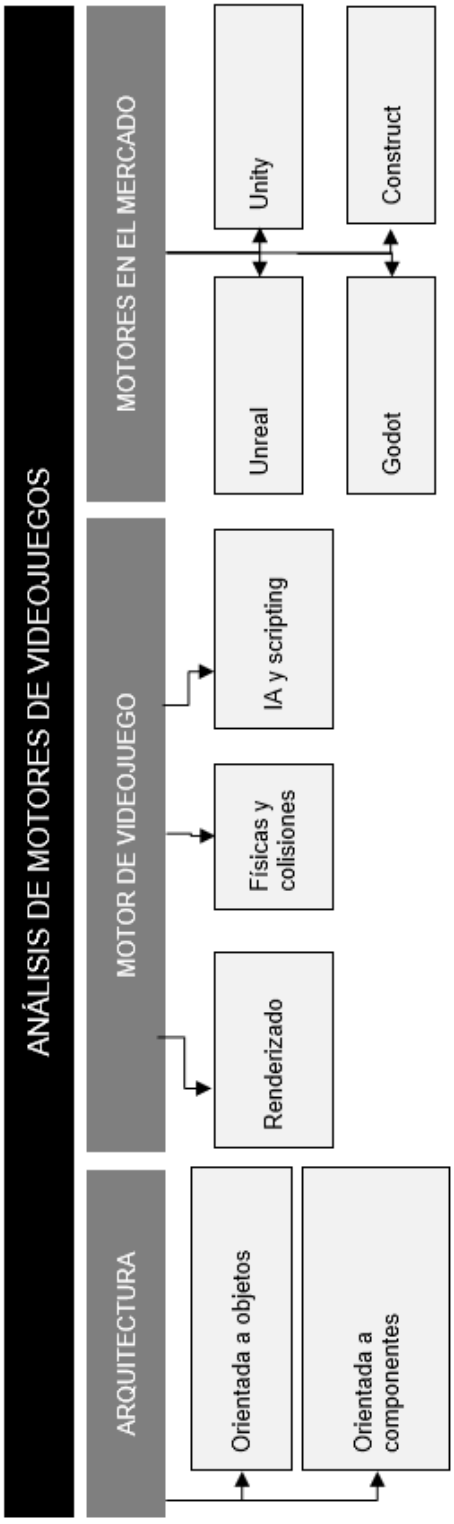
Programación multimedia y dispositivos móviles

# Análisis de motores de videojuegos

# Índice

# Esquema


ANÁLISIS DE MOTORES DE VIDEOJUEGOS
ARQUITECTURA
Orientada a objetos
Orientada a componentes
MOTOR DE VIDEOJUEGO
Renderizado
Físicas y colisiones
IA y scripting
MOTORES EN EL MERCADO
Unreal
Unity
Godot
Construct

# Material de estudio

## 8.1. Introducción y objetivos

Los objetivos que se persiguen en el siguiente tema son:

- Conocer las distintas arquitecturas de software que tenemos disponibles para el desarrollo de videojuegos.
- Aprender qué son los motores gráficos de videojuegos.
- Comparar distintos motores gráficos de videojuegos.
- Conocer Unity, sus características y su importancia en el mercado.

## 8.2. Conceptos de arquitectura y diseño de videojuegos

La arquitectura de software representa la estructura del sistema, que consta de los componentes de software, las propiedades visibles externamente y las relaciones entre ellas. En este apartado vamos a mostrar qué es la arquitectura de software en los videojuegos.

**1.** Arquitectura orientada a objetos

En un diseño orientado a objetos cada cosa en el juego suele estar representada por una clase única, como puede ser un jugador, una bala, un enemigo o un coche.

Cada clase hereda de otras; por ejemplo, en un juego puede existir la clase avión y la clase helicóptero. Ambos son similares y en nuestra jerarquía de clases heredan de una clase abstracta, llamada vehículo, que representa a un vehículo. Esta, a su vez, hereda de una clase PhysicObject que representa a un objeto físico del mundo, y

todas ellas hereden de una clase superior, llamada GameObject que las identi ca dentro del juego.

Todo este sistema de herencia hace que cada vez tengamos clases más especializadas, que cumplen su función aprovechando el código genérico lo máximo posible.

**2.** Arquitectura orientada a componentes

Uno de los principios básicos del diseño de software es la modularidad, es decir, hacer que cada sistema o componente sea lo más independiente del resto posible. Ese es el principio en el que se basa la programación orientada a entidades y componentes.

En el paradigma Entidad-Componente, cada elemento del juego es una entidad independiente, ya se trate del jugador, una bala, un enemigo, un coche o un tanque. Cada uno de ellos se debe representar como una entidad única, sin herencia.

Luego, cada entidad está compuesta, a su vez, de una serie de componentes. Por ejemplo, la entidad Avión puede tener componentes como Velocidad, Posición o Modelo. En cualquier momento podemos quitar o añadir un componente a una entidad sin que afecte al resto de componentes, ya que son independientes y uno no necesita conocer la existencia del otro.

Los componentes son el lugar donde guardamos "datos relevantes". Por ejemplo, un componente de posición en un sistema 2D tendría dos datos: una coordenada X y una coordenada Y.

## 8.3. Qué es un motor de videojuegos y sus componentes

Un motor de videojuego es una serie de librerías de programación que permiten el diseño, la creación y la representación de un videojuego. Entre las herramientas que

pueden incluir los motores de videojuegos se encuentran: motor de renderizado, física y detección de colisiones, scripting, inteligencia artificial y administradores de memoria, entre otros.

Muchos motores de videojuegos proporcionan abstracción de hardware, de manera que nos permiten desarrollar videojuegos sin la necesidad de conocer la arquitectura del hardware de la plataforma donde van a ejecutarse.

La abstracción de hardware es esencial para el desarrollo de motores con características multiplataforma, que puedan exportarse a distintas plataformas y dispositivos.

**Motor de renderizado**

El motor de renderizado de un motor de videojuegos es el componente responsable de generar las imágenes que los jugadores ven en pantalla a partir de los datos del juego, como modelos 3D, texturas, luces, sombras y efectos especiales. Es una parte fundamental de un motor de videojuegos, y su diseño y optimización tienen un impacto significativo en la calidad visual, el rendimiento y la experiencia del jugador.

Es común que esté implementado a nivel de API como DirectX u OpenGL, puesto que estas aplicaciones proveen de abstracciones de la GPU y acceso a algunos componentes de hardware, independientemente de la plataforma que se utilice.

Componentes principales del motor de renderizado

- Pipeline de renderizado: El pipeline de renderizado es el proceso que convierte los datos 3D en una imagen 2D que se muestra en la pantalla. Este proceso se divide en varias etapas:
    - Transformaciones geométricas: Convierte las coordenadas del modelo desde su espacio local al espacio de la cámara y, finalmente, al espacio de pantalla.
    - Rasterización: Convierte los triángulos 3D en píxeles 2D.

- Sombras (shading): Aplica cálculos de iluminación y materiales para determinar el color final de cada píxel.
- Postprocesamiento: Aplica efectos visuales como desenfoque, profundidad de campo, reflejos, oclusión ambiental y corrección de color.

- Shaders: Los shaders son pequeños programas ejecutados en la GPU que controlan aspectos específicos del renderizado:
  - Vertex Shader: Procesa los vértices del modelo, aplicando transformaciones geométricas y calculando atributos como normales o coordenadas de textura.
  - Fragment Shader: Calcula el color de cada píxel basado en la iluminación, las texturas y otros parámetros.
  - Geometry Shader (opcional): Permite modificar la geometría en tiempo real, como generar partículas o alterar vértices.

- Iluminación:
  - Los motores de renderizado implementan técnicas de iluminación para simular cómo interactúa la luz con los objetos. Esto incluye modelos de iluminación como Phong, Blinn-Phong, PBR (Physically-Based Rendering), entre otros.
  - Métodos como iluminación directa, iluminación global (Global Illumination) y técnicas avanzadas como trazado de rayos (Ray Tracing).

- Texturas y materiales:
  - Las texturas son imágenes aplicadas a los modelos 3D para proporcionar detalles visuales, como colores, rugosidad o normal maps.
  - Los materiales combinan texturas y propiedades físicas (brillo, reflectividad) para simular diferentes superficies.

- Culling y optimización:
    - Frustum Culling: Se descartan objetos que no están dentro del campo de visión de la cámara.
    - Occlusion Culling: Evita renderizar objetos ocultos detrás de otros.
    - Level of Detail (LOD): Utiliza modelos menos detallados para objetos lejanos.

- Postprocesado: Después de renderizar una escena, el motor aplica efectos visuales adicionales para mejorar la calidad estética, como anti-aliasing, efectos de partículas, o desenfoque de movimiento.

**Física de videojuegos y detección de colisiones**

La física en videojuegos tiene como objetivo simular las leyes del mundo real (o versiones estilizadas de ellas) para dar credibilidad y consistencia a los movimientos e interacciones en el juego. El motor del videojuego será el que se encargue de realizar cálculos para que un objeto simule tener atributos físicos como peso, volumen, estado físico, gravedad, etc.

Los componentes principales de la física en videojuegos:

**1.** Dinámica de cuerpos rígidos:

- Simula el movimiento de objetos sólidos que no cambian de forma.
- Basado en las leyes del movimiento de Newton (posición, velocidad, aceleración, fuerza, torque).

**2.** Cuerpos blandos:

- Simula objetos que pueden deformarse, como gelatinas, ropa o líquidos.
- Requiere simulaciones más avanzadas, como redes de partículas o mallas deformables.

**3.** Física de partículas:

- Simula sistemas que constan de múltiples partículas individuales, como humo, fuego, o lluvia.
- Los cálculos son menos complejos porque no hay interacciones rígidas entre las partículas.

**4.** Fricción y elasticidad:

- La fricción afecta cómo los objetos se deslizan o ruedan sobre superficies.
- La elasticidad determina cómo los objetos rebotan al chocar (coeficiente de restitución).

**5.** Gravedad: Generalmente, se simula una gravedad constante que afecta a todos los objetos hacia una dirección específica (e.g., g=9.8 m/s2g gravedad en la Tierra).

**6.** Física de fluidos: Simula líquidos y gases, a menudo usando sistemas de partículas o simulaciones volumétricas.

La detección de colisiones es una subparte de la física de videojuegos que identifica cuándo y dónde los objetos interactúan entre sí.
Tareas clave en la detección de colisiones:

**1.** Detección de intersección:

- Determinar si dos objetos están colisionando.
- Esto se basa en la geometría de los objetos: formas simples (esferas, cubos) o complejas (mallas).

**2.** Resolución de colisiones:

- Calcular cómo reaccionan los objetos después de una colisión.
- Puede incluir ajustes en posición, fuerza, velocidad o rotación.

3. Prevención de penetración: Garantiza que los objetos no se superpongan físicamente después de una colisión.

**Scripting**

Los motores de videojuegos nos van a permitir añadir nuestros propios scripts para disponer de funcionalidades que no existan en el motor.

En algunos casos no será necesario añadirlas, sino simplemente modificar algún comportamiento ya existente en el motor.

Cada motor de videojuego utiliza lenguajes de programación diferentes, en función del público objetivo, tecnología con la que esta construida, etc. Entre los más destacados podemos encontrar C#, C++, JavaScript, Python o lenguajes propios del motor.

**Inteligencia artificial**

La inteligencia artificial (IA) ha transformado de manera significativa el desarrollo de videojuegos, mejorando la experiencia del jugador y expandiendo las posibilidades creativas para los desarrolladores. Podemos destacar las siguientes ramas cómo las principales en las que la IA influye de forma notoria en el desarrollo de videojuegos:

1. Personajes no jugables (NPCs) inteligentes: Los NPCs utilizan algoritmos de IA para reaccionar de manera creíble ante las acciones del jugador.

- Árboles de decisión: Una estructura lógica para tomar decisiones en base a condiciones.
- Máquinas de estados finitos: Los NPCs cambian entre estados (patrullar, atacar, huir) dependiendo de las circunstancias.
- Pathfinding: Algoritmos como A* permiten que los personajes encuentren rutas óptimas para moverse en el entorno.

**2.** Generación procedimental: La IA permite crear contenido dinámico y único en tiempo real.

- Mapas y niveles: Herramientas como el Perlin noise o algoritmos basados en IA generan mundos virtuales únicos y expansivos, como en Minecraft o No Man's Sky.
- Misiones y narrativas: La IA puede generar historias o eventos secundarios que se adaptan al progreso del jugador, lo que incrementa la rejugabilidad.

**3.** Aprendizaje automático en videojuegos

- Entrenamiento de NPCs: Algoritmos de aprendizaje por refuerzo permiten que los NPCs aprendan y mejoren con el tiempo. Por ejemplo, un enemigo podría "aprender" tácticas para contrarrestar al jugador según su estilo de juego.
- IA para bots avanzados: Juegos como Dota 2 han implementado bots entrenados con redes neuronales para competir incluso contra jugadores profesionales.

**4.** Experiencias personalizadas

- Adaptación al jugador: La IA puede ajustar dinámicamente la dificultad y el contenido del juego según el comportamiento del usuario. Por ejemplo, en Left 4 Dead, el sistema de IA conocido como "Director" ajusta la intensidad de los ataques enemigos para mantener un ritmo equilibrado.
- Recomendaciones de juego: Basado en el análisis del comportamiento del jugador, algunos juegos sugieren estrategias, personajes o configuraciones específicas.

**5.** Animaciones y simulación

- Animaciones realistas: La IA mejora las animaciones de personajes, permitiendo transiciones suaves entre acciones. Juegos como Red Dead Redemption 2 usan simulaciones avanzadas para que las interacciones físicas y comportamientos sean naturales.
- Simulación de multitudes: La IA gestiona grandes grupos de personajes, como en juegos de mundo abierto, para que las ciudades y entornos parezcan vivos y auténticos.

**6.** Creación de diálogos y narrativa

- Diálogos generativos: Con el uso de modelos de lenguaje como GPT, los videojuegos pueden generar conversaciones dinámicas y únicas entre NPCs o entre NPCs y jugadores.
- Mundos narrativos emergentes: La IA puede crear narrativas en tiempo real que reaccionen al progreso del jugador, adaptando la historia según sus decisiones.

**Administración de memoria**

La administración de memoria en los motores de videojuegos es un aspecto crítico que impacta tanto en el rendimiento como en la experiencia del jugador. En videojuegos, los recursos suelen ser limitados (RAM, VRAM, almacenamiento) y los motores deben gestionar de manera eficiente el uso de memoria para garantizar que el juego funcione de manera fluida y estable. La tarea del administrador de memoria es saber cuándo liberar componentes del juego que no están en uso, o que ya han desaparecido, por ejemplo.

Una buena gestión del administrador de memoria es fundamental para aportar fluidez al juego y, en definitiva, a la experiencia del usuario.

**1.** Recolección de basura (Garbage Collection)

Motores como Unity usan garbage collection para liberar memoria automáticamente de objetos que ya no son referenciados. Sin embargo, esto puede causar caídas de rendimiento si no se maneja adecuadamente. Los desarrolladores deben minimizar los picos de recolección al optimizar cómo y cuándo se crean/destruyen los objetos.

**2.** Problemas comunes en la administración de memoria

- Fragmentación de memoria: Ocurre cuando se crean y destruyen objetos constantemente, dejando espacios pequeños en la memoria que no se pueden reutilizar eficientemente.

- Fugas de memoria: Datos que se asignan, pero nunca se liberan, causando que el uso de memoria aumente progresivamente. Se deben de usar herramientas de monitoreo para identificar y corregir fugas.
- Sobrecarga de memoria: Cuando un juego usa más memoria de la disponible, puede causar "crasheos" o reducir el rendimiento. Como posible solución se pueden implementar límites y estrategias como el streaming de datos.

## 8.4. Estudio de motores de videojuegos disponibles en el mercado

El mercado ofrece una variedad de motores de videojuegos que permiten desarrollar y exportar proyectos a dispositivos móviles, tanto Android como iOS. Estos motores ofrecen herramientas avanzadas para crear experiencias atractivas, optimizadas para pantallas táctiles y dispositivos con recursos limitados.

**1.** Unity

Características clave:

- Plataformas soportadas: Android, iOS, Windows, macOS, consolas y más.
- Lenguaje de programación: C#.
- Editor visual: Interfaz intuitiva con soporte para programación visual (mediante complementos como Bolt).
- Gráficos: Amplio soporte para 2D y 3D, con tecnologías avanzadas como URP (Universal Render Pipeline) y HDRP (High Definition Render Pipeline).
- Optimización para móviles:
    - Compresión de texturas.
    - Herramientas de análisis como Unity Profiler.
    - Gestión de assets mediante Addressables para reducir tiempos de carga.

Ventajas:

- Comunidad extensa y abundancia de tutoriales.
- Amplia biblioteca de assets en la Unity Asset Store.
- Fácil integración con herramientas de monetización y análisis (como Unity Ads o Firebase).

Desventajas:

- El sistema de Garbage Collection puede afectar el rendimiento en dispositivos con recursos limitados.
- Costos adicionales para licencias profesionales en proyectos comerciales.

**2.** Unreal Engine

Características clave:

- Plataformas soportadas: Android, iOS, consolas, PC, y más.
- Lenguaje de programación: C++ (además de Blueprints para programación visual).
- Gráficos: Reconocido por su calidad gráfica, con tecnologías como Nanite y Lumen.
- Optimización para móviles:
    - Herramientas avanzadas de ajuste de LOD (Level of Detail).
    - Capacidades de compresión y reducción de assets específicos para móviles.

Ventajas:

- Calidad gráfica excepcional, ideal para juegos de alta fidelidad.
- Soporte para grandes mundos abiertos y simulaciones físicas.
- Herramientas de monetización y soporte integrado para compras dentro de la aplicación.

Desventajas:

- Mayor curva de aprendizaje, especialmente para principiantes.
- Requisitos de hardware más altos para el desarrollo y pruebas.

**3.** Godot Engine

Características clave:

- Plataformas soportadas: Android, iOS, HTML5, Windows, Linux, macOS.
- Lenguaje de programación: GDScript (similar a Python), C# y soporte para lenguajes de bajo nivel como C++.
- Gráficos: Compatible con 2D y 3D, con un enfoque particularmente fuerte en el desarrollo de juegos 2D.
- Optimización para móviles:
    - Uso eficiente de recursos para juegos ligeros.
    - Herramientas de depuración integradas para ajustar el rendimiento.

Ventajas:

- Open source y completamente gratuito.
- Ligero y rápido para proyectos pequeños y medianos.
- Ideal para desarrolladores independientes y prototipado rápido.

Desventajas:

- Menos recursos y tutoriales en comparación con Unity o Unreal.
- Gráficos y capacidades 3D menos avanzados que Unreal Engine.

**4.** Cocos Creator

Características clave:

- Plataformas soportadas: Android, iOS, Windows, HTML5 y más.
- Lenguaje de programación: TypeScript y JavaScript (C++ para desarrolladores avanzados).
- Gráficos: Orientado a 2D, con soporte básico para 3D.
- Optimización para móviles:
    - Tamaño reducido de los ejecutables.
    - Enfoque en juegos ligeros, ideales para mercados emergentes.

Ventajas:

- Ligero y eficiente, ideal para juegos casuales.

- Código abierto con personalización avanzada.
- Gran soporte en mercados asiáticos.

Desventajas:

- Funcionalidades 3D limitadas.
- Comunidad menos extensa que Unity o Unreal.

**5.** Construct

Características clave:

- Plataformas soportadas: Android, iOS, HTML5, Windows.
- Lenguaje de programación: Basado en lógica visual, ideal para desarrolladores sin experiencia en programación.
- Gráficos: Enfocado en juegos 2D.
- Optimización para móviles:
    - Exportación directa a plataformas móviles.
    - Soporte para juegos ligeros.

Ventajas:

- Excelente para principiantes y juegos casuales.
- Rápido tiempo de desarrollo gracias a su editor visual.
- No requiere conocimientos avanzados de programación.

Desventajas:

- Menos flexibilidad para proyectos complejos.
- Menor soporte para gráficos avanzados.

Existen multitud de opciones, y continuamente están apareciendo nuevos motores, tanto gratuitos como con algún sistema de pago, por lo que la lista podría ser interminable.

Es importante, a la hora de elegir un motor en el que desarrollar, tener en cuenta factores como:

- La duración en el tiempo, ya que no queremos desarrollar un videojuego en un motor que deje de actualizarse y haga obsoleto nuestro juego o sus funciones.
- La comunidad de usuarios que posee, que nos servirá para resolver dudas y aprender a usar el motor.
- La curva de aprendizaje, porque es importante no perder las ganas o gastar todas las fuerzas en aprender a usar el motor, sin haber empezado a realizar el videojuego.

## 8.5. Estudio de juegos y aplicaciones creadas en Unity

Unity es un motor extremadamente versátil, utilizado en una amplia gama de proyectos, desde juegos AAA hasta aplicaciones educativas, simuladores y experiencias de realidad virtual/aumentada. Su flexibilidad, extensibilidad y soporte para múltiples plataformas lo convierten en una opción preferida tanto para desarrolladores independientes como para grandes estudios. Si bien existen desafíos, su robusta comunidad y herramientas integradas lo posicionan como uno de los motores más influyentes en la actualidad.

**1.** Juegos AAA

- CupHead: Juego de acción y plataformas con un estilo visual inspirado en los dibujos animados de los años 30, con animaciones dibujadas a mano. Unity permitió gestionar eficientemente los recursos gráficos y la compleja lógica de combate.
- Ori and Blind Forest: Juego de tipo metroidvania que combina gráficos de alta calidad con una narrativa emocional. Unity facilitó la creación de un juego visualmente impactante y con una experiencia fluida.

**2.** Juegos independientes

- Hollow Knight: Unity ofreció herramientas flexibles para desarrollar un mundo 2D inmersivo.
- Among Us: Unity permitió una rápida iteración en mecánicas y soporte para múltiples plataformas.

**3.** Juegos móviles

- Monument Valley: Juego de puzles con un diseño artístico minimalista y rompecabezas basados en perspectivas imposibles. Unity permitió combinar gráficos estilizados con una experiencia interactiva fluida.
- Pokemon Go: Unity facilitó la implementación de características de AR y compatibilidad multiplataforma.

**4.** Realidad Virtual (VR) y realidad aumentada (AR)

- Beat Saber: Juego de ritmo mediante interacción precisa con controladores de movimiento. Unity permitió optimizar el rendimiento en dispositivos VR.
- Superhot VR: Shooter en primera persona cuya donde el tiempo se mueve solo cuando el jugador lo hace. Unity facilitó la implementación de la lógica de tiempo y la interacción inmersiva.
- Ikea Place: Unity facilitó la implementación de AR con precisión espacial.
- Google Earth VR: Unity ofreció soporte para renderizar mapas 3D interactivos.

**5.** Aplicaciones educativas

- Codecombat: Unity permitió crear un entorno visualmente atractivo para aprender habilidades técnicas. Enseña programación a través de un juego interactivo
- Anatomy 4D: Unity fue esencial para integrar gráficos interactivos y funcionalidad de AR.

**6.** Simulaciones y entrenamientos

- Flight Simulators: Unity se utiliza para crear simuladores de vuelo interactivos y realistas. Permite entrenar pilotos en un entorno seguro y económico.

- Medical Simulators: Entrenar procedimientos médicos utilizando modelos interactivos en 3D. Unity soporta modelos de alta precisión y físicas avanzadas.

# A fondo

**Unreal Engine**

https://www.unrealengine.com/es-ES

Documentación en línea. (2024)

Documentación oficial sobre Unreal Engine, el motor de videojuegos referente.

**Construct**

https://www.construct.net/en

Documentación en línea. (2024)

Documentación oficial sobre el motor de videojuegos multiplataforma Construct.

**Godot Engine**

https://godotengine.org/

Documentación en línea. (2024)

Documentación oficial sobre sobre el motor de videojuegos multiplataforma Godot.

**Unity showcase**

https://www.youtube.com/watch?v=ucalG1dsvL4&ab_channel=Unity

Video de Youtube. (2023)

Video sobre videojuegos creados con Unity en 2023.

# Entrenamientos

**Entrenamiento 1**

- Realiza un estudio de las diferentes planes o licencias de Unity Engine.

- Desarrollo paso a paso y solución: https://unity.com/products/compare-plans

**Entrenamiento 2**

- Crear una cuenta en Unity.com y descargar la versión Unity Hub y la versión Unity personal.

- Desarrollo paso a paso y solución: https://unity.com/es/download

**Entrenamiento 3**

- Instalar Unity 6 en nuestro equipo.

- Desarrollo paso a paso y solución: https://www.youtube.com/watch?v=eTWU4rWAO6k&ab_channel=LuisCanary

**Entrenamiento 4**

- Instalación y configuración de Visual Studio Code para el desarrollo de C#.

- Desarrollo paso a paso y solución: https://learn.microsoft.com/es-es/training/modules/install-configure-visual-studio-code/

**Entrenamiento 5**

- Configurar Unity y Visual Studio Code para usar Visual Studio Code como editor de código en Unity.

- Desarrollo paso a paso y solución: https://www.youtube.com/watch?v=KVgXmdkGZS8&ab_channel=BravePixelG