Componente formativo

Documentación de integración de Assets

Breve descripción:

Se explican conceptos básicos de optimización de *assets* y documentación de integración.

Área ocupacional:

Ciencias Naturales.

Mayo 2023

Grupo de Ejecución de la Formación Virtual

Tabla de contenido

Introducción

En este componente formativo se abordarán conceptos básicos de optimización de assets y documentación de integración, a continuación, la ruta que se trazará:

Video 1. Documentación de integración de assets

Documentación de integración de assets

Síntesis del video: Documentación de integración de assets
En el desarrollo de los videojuegos es muy importante el aprovechamiento de los recursos de un sistema que permita mejorar el rendimiento de un juego lo que significa, optimizar el código, los gráficos, los sonidos, las animaciones, entre otros aspectos. En los juegos 3D, parte de la responsabilidad de optimización recae en los artistas por el diseño de los gráficos, ya que la base de optimización radica en el número de polígonos, sombras y la calidad de las texturas que utiliza.

Trabajar sobre las mejoras medibles en el uso de recursos, los tiempos de carga, el uso de memoria o reducir el tamaño de la aplicación permite tener una mayor eficiencia y mejor rendimiento en un videojuego.

1. Verificación y optimización de assets

Las partes gráficas de un videojuego pueden tener injerencia en dos sistemas del cómputo: el GPU o el CPU; y la primera regla de optimización es en encontrar dónde está el problema de rendimiento para determinar las estrategias que se utilizarán en la optimización de todos los recursos del videojuego.

1.1. Métricas de rendimiento

En términos generales, las métricas de rendimiento involucran procedimientos, técnicas y tecnologías que permiten realizar seguimiento del rendimiento del sistema, entendiendo este último, en el desarrollo de videojuegos, como el videojuego en sí mismo, como aplicación de software.

En el motor de videojuegos Unity3D, se puede encontrar una herramienta global denominada Unity Profiler que reúne diferentes subherramientas para determinar métricas de rendimiento en tiempo real, mientras se ejecuta un videojuego en tiempo de ejecución en modo edición. Cada una de estas herramientas brindan información detallada del comportamiento de diversos elementos y aspectos a manera de indicadores que permiten determinar un perfil general de rendimiento.

Unity Profiler: es una herramienta que se puede utilizar para obtener información sobre el rendimiento de la aplicación. Se puede conectar a dispositivos en la red o conectados a la máquina para probar cómo se ejecuta la aplicación en la plataforma de lanzamiento prevista. También puede ejecutarlo en el editor para obtener una descripción general de la asignación de recursos mientras desarrolla la aplicación.

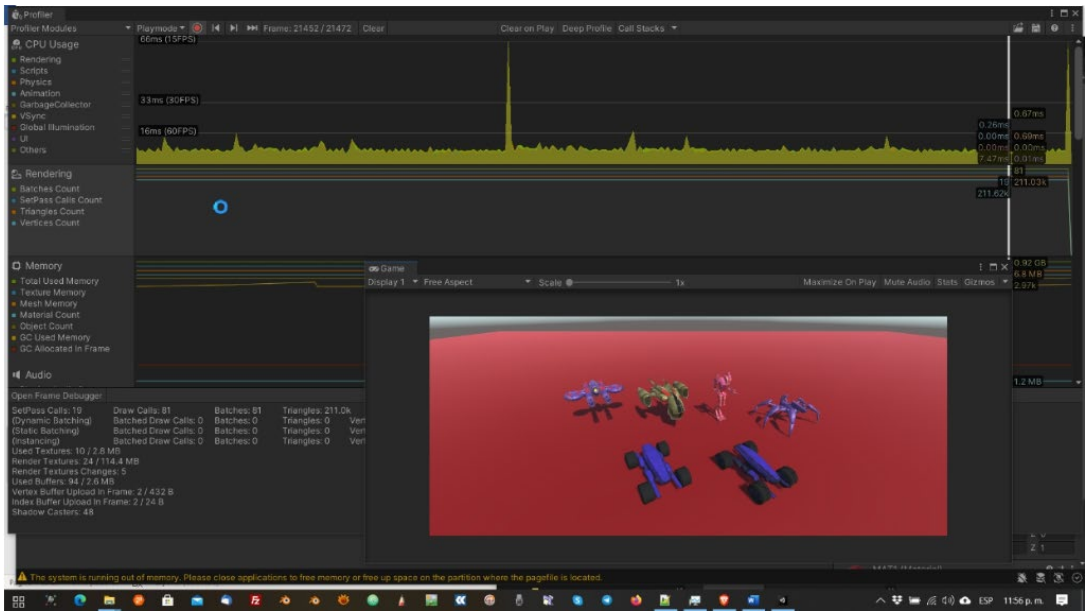
Unity Profiler recopila y muestra datos sobre el rendimiento de su aplicación en áreas como la CPU, la memoria, el renderizador y el audio. Es una herramienta útil para identificar áreas de mejora del rendimiento en su aplicación e iterar en esas áreas. Puede identificar cosas como su código, recursos activos, configuraciones de escena, renderizado de cámara y configuraciones de compilación que pueden y afectan el rendimiento de su aplicación. Muestra los resultados en una serie de gráficos para visualizar dónde ocurren los picos en el rendimiento de su aplicación.

Además de utilizar Unity Profiler integrado, puede utilizar la API Profiler de complemento nativo de bajo nivel para exportar datos de creación de perfiles a herramientas de creación de perfiles de terceros, y el paquete Profiling Core para personalizar su análisis de creación de perfiles. También puede agregar potentes herramientas de creación de perfiles, como Memory Profiler y Profile Analyzer a su proyecto para analizar los datos de rendimiento con más detalle.

Para acceder a la ventana Unity Profiler vaya al menú:

Window > Analysis > Profiler

Figura 1. Ventana Profiler



El generador de perfiles registra varias áreas del rendimiento de su aplicación y le muestra esa información. Puede utilizar esta información para tomar decisiones informadas sobre lo que podría necesitar optimizar en su aplicación y para confirmar que sus optimizaciones producen los resultados que espera.

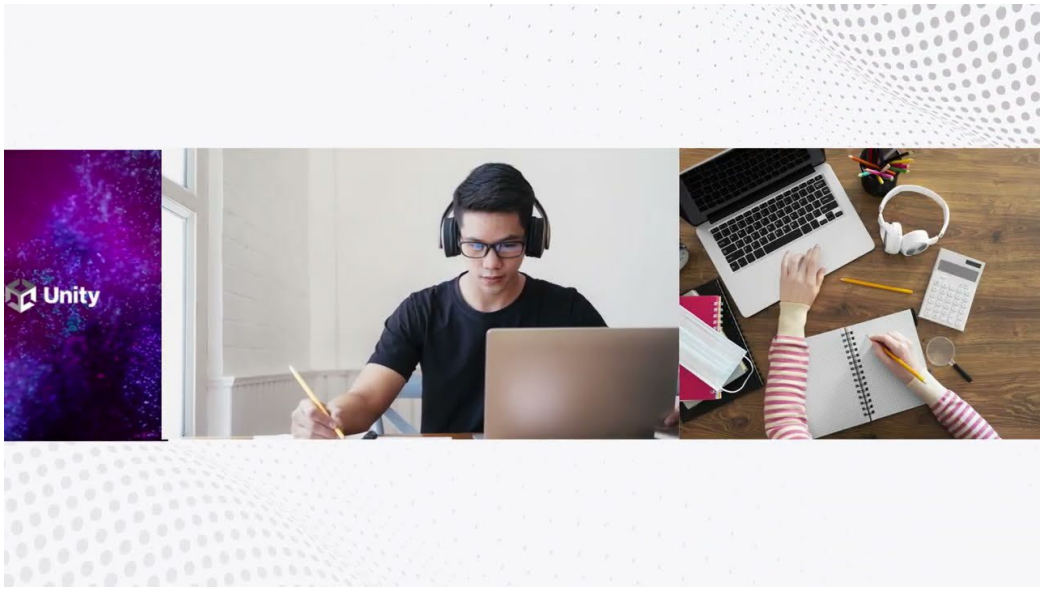
De forma predeterminada, Unity Profiler registra y mantiene los últimos 300 fotogramas del juego y muestra información detallada sobre cada fotograma. Puede aumentar el número de fotogramas que registra en la ventana de Preferencias (menú: Edit > Preferences), hasta 2.000 fotogramas.

Nota: si aumenta esta configuración a una gran cantidad de fotogramas, la sobrecarga del generador de perfiles y el uso de la memoria pueden aumentar el rendimiento.

Puede inspeccionar el código de la secuencia de comandos y cómo su aplicación usa ciertos activos y recursos que podrían estar ralentizándola. También puede comparar el rendimiento de su aplicación en diferentes dispositivos. Esta herramienta tiene varios Módulos de Perfilado diferentes que puede agregar a su sesión de creación de perfiles para obtener más información sobre áreas como renderizado, memoria y audio.

Para saber más acerca de la ventana de Unity Profiler, el siguiente video da información relevante:

Video 2. Ventana de Profiler

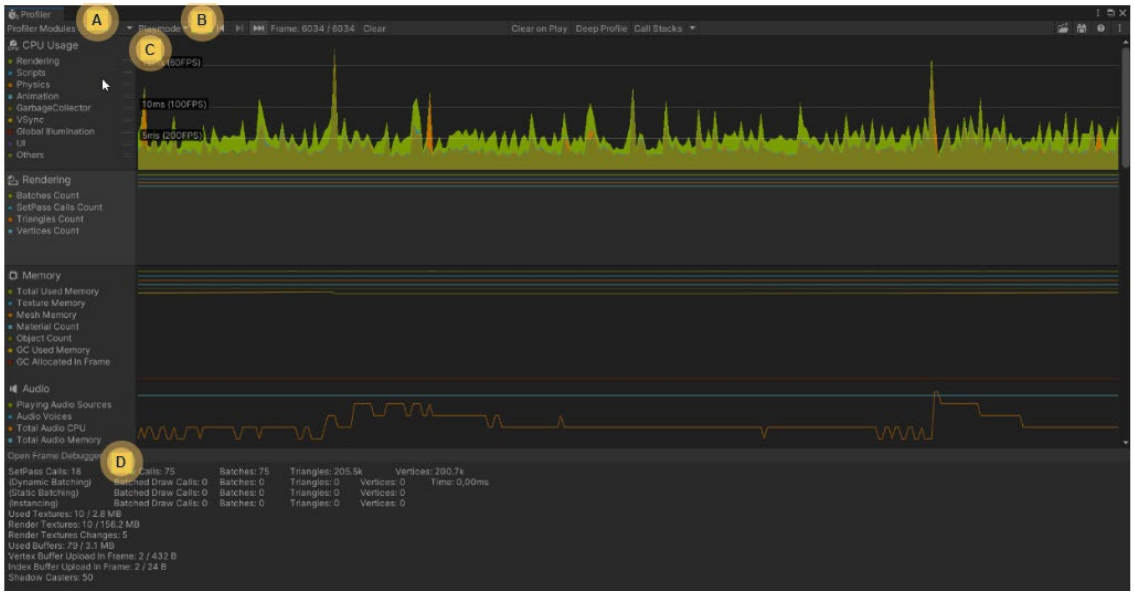


Ventana de Profiler

Síntesis del video: Ventana de Profiler
Es un video tutorial, que muestra el paso a paso para la creación de la Ventana de Profiler.

a) Diseño de la ventana del generador de perfiles

Figura 2. Ventana del generador de perfiles



A. Módulos de perfilado.

Esta es una lista de todos los módulos que puede perfilar en su aplicación. Utilice el menú desplegable en la parte superior de esta área para agregar y eliminar módulos de la ventana.

B. Controles del generador de perfiles.

Utilice estos controles para establecer desde qué dispositivo crear un perfil y qué tipo de creación de perfiles debe realizar Unity, navegar entre los fotogramas y comenzar a registrar datos.

C. Diagramas de cuadros.

Esta área contiene gráficos de cada módulo, los perfiles de Unity Profiler. Esta área está en blanco cuando abre el generador de perfiles por primera vez y se llena de información cuando comienza a crear perfiles de su aplicación.

D. Panel de detalles del módulo.

La información en esta área de la ventana cambia según el módulo que haya seleccionado. Por ejemplo, cuando selecciona el módulo CPU Usage Profiler, contiene una

línea de tiempo detallada y la opción de cambiar a una vista de jerarquía. Cuando selecciona el módulo Rendering Profiler, esta área muestra una lista de información de depuración. Esta área está en blanco cuando abre el generador de perfiles por primera vez y se llena de información al crear perfiles de su aplicación.

b) Preferencias del generador de perfiles

La ventana Preferencias contiene configuraciones adicionales de la ventana Unity Profiler de la siguiente manera:

Tabla 1. Ventana de preferencias

Preferencia	Descripción
Frame count	Establece el número máximo de fotogramas que debe capturar Profiler, que puede ser entre 300 y 2.000.
Show stats for 'current frame'	De forma predeterminada, cuando se selecciona el botón Cuadro actual e ingresa al modo Cuadro actual, la línea indicadora del cuadro no tiene anotaciones con las estadísticas del cuadro actual. Esto se debe a que las anotaciones de estadísticas pueden dificultar la visualización de datos en tiempo real. Para mostrar las anotaciones, se debe habilitar esta configuración.
Default recording state	Se selecciona en qué estado de grabación debe abrirse el generador de perfiles, se puede elegir entre Activado, Desactivado o Recordar. Habilitado mantiene el botón Grabar habilitado entre sesiones, mientras que Deshabilitado lo deshabilita, independientemente de si se enciende o apaga durante la sesión de creación de perfiles. El estado Recordar indica si se ha habilitado o deshabilitado el botón Grabar durante la sesión y lo mantiene en su último estado la próxima vez que se abra la ventana del Generador de perfiles.

Preferencia	Descripción
Default editor target mode	Se establece el modo al que debe dirigirse el menú desplegable Adjuntar al jugador de forma predeterminada. Elegir entre el Modo de reproducción o el Editor.

c) Módulos de perfilador

La parte superior de la ventana Profiler está dividida en módulos de perfilador que perfilan áreas específicas de tu juego. Cuando crea un perfil de la aplicación, Unity muestra los datos relacionados con cada módulo en los gráficos correspondientes.

El módulo de uso de CPU proporciona la mejor descripción general de cuánto tiempo pasa la aplicación en cada marco. Los otros módulos recopilan datos más específicos y pueden ayudar a inspeccionar áreas más específicas o monitorear los aspectos vitales de la aplicación, como el consumo de memoria, la representación o las estadísticas de audio.

A continuación, se presentan estos módulos en la siguiente tabla.

Tabla 2. Módulos de perfilador

Profiler Module	Función
CPU Usage	Muestra una descripción general de dónde pasa la mayor parte del tiempo la aplicación, en áreas como física, scripts, animación y recolección de basura. Este módulo contiene información amplia de perfiles sobre la aplicación y se puede usar para decidir qué módulos adicionales usar para investigar problemas más específicos en la aplicación. Este módulo siempre está activo, incluso si se cierra; se debe consultar el módulo de perfilador de uso de CPU.
GPU Usage	Muestra información relacionada con el procesamiento de gráficos. De forma predeterminada, este módulo no está activo, porque tiene una sobrecarga alta.

Profiler Module	Función
Rendering	Muestra información sobre cómo Unity renderiza gráficos en la aplicación, incluida información sobre lotes estáticos y dinámicos, llamadas SetPass y Draw, triángulos y vértices.
Memory	Indica la información sobre cómo Unity asigna memoria en la aplicación. Esto es particularmente útil para ver cómo las asignaciones de secuencias de comandos (GC.Alloc) conducen a la recolección de basura, o cómo las tendencias de uso de la memoria de activos de la aplicación a lo largo del tiempo.
Audio	Muestra información relacionada con el audio en la aplicación, como cuándo y cuántas fuentes de audio se reproducen, cuánto uso de CPU requiere el sistema de audio y cuánta memoria le asigna Unity.
Video	Muestra información relacionada con el video en la aplicación.
Physics	Brinda información sobre la física de la aplicación que ha procesado el motor de física.
Physics (2D)	Similar al módulo Physics Profiler, este módulo muestra información sobre dóndel motor de física ha procesado la física 2D en la aplicación.
Network Messages (deprecated)	Da información sobre paquetes y mensajes de nivel inferior enviados o recibidos por la API de alto nivel multijugador.
Network Operations (deprecated)	Muestra información detallada sobre qué tipos de operaciones se encuentran en los mensajes enviados y recibidos por la API de alto nivel multijugador, como cuántas SyncVars o Comandos se han transferido.
UI	Muestra información sobre cómo Unity maneja el procesamiento por lotes de la interfaz de usuario para la

Profiler Module	Función
	aplicación, incluido por qué y cómo Unity procesa los elementos por lotes.
UI Details	De manera similar al módulo de la interfaz de usuario, el gráfico de este módulo agrega datos sobre el recuento de lotes y vértices, así como marcadores que incluyen información sobre los eventos de entrada del usuario que activan los cambios de la interfaz de usuario.
Global Illumination	Evidencia la información sobre la cantidad de recursos de CPU que Unity gasta en el subsistema de iluminación de iluminación global de la aplicación.
Virtual Texturing	Muestra estadísticas sobre la transmisión de texturas virtuales en la aplicación.

Cada módulo tiene su propio gráfico. Cuando selecciona un módulo, aparece un panel Detalles del módulo en la sección inferior de la ventana que se puede usar para inspeccionar los datos detallados que recopila el módulo.

d) Método de perfilación de una aplicación

Cuando se usa Unity Profiler para perfilar la aplicación, hay dos formas principales de registrar datos:

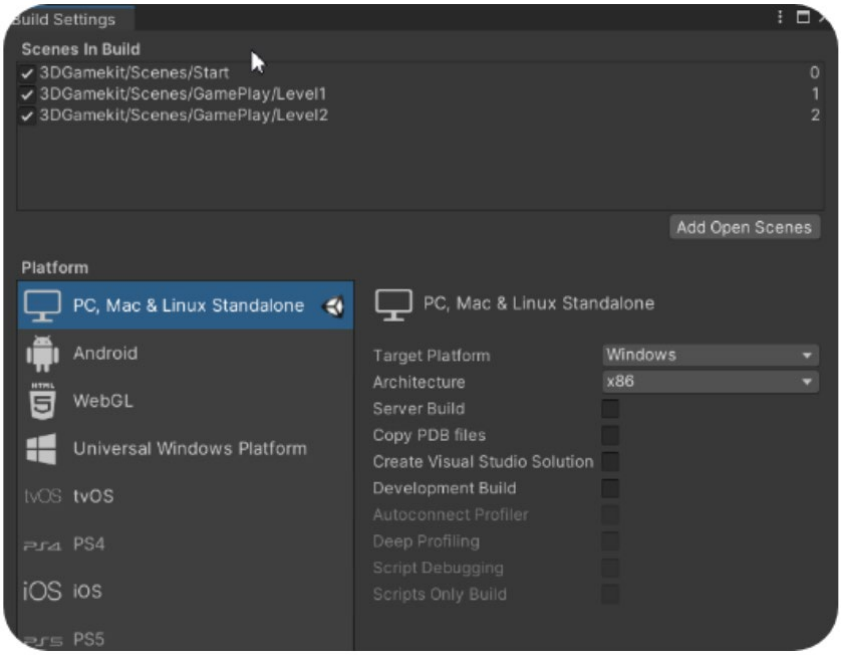
Para perfilar su aplicación en su plataforma de lanzamiento de destino, conecte el dispositivo de destino a su red o directamente a su computadora mediante un cable. También puede conectarse a un dispositivo a través de la dirección IP, solo puede crear un perfil de su aplicación como una compilación de desarrollo.

Para configurar esto, vaya a Configuración de compilación (menú: File > Build Settings) y seleccione la plataforma de destino de su aplicación. Habilite la configuración de

Development Build. Cuando habilita esta configuración, dos configuraciones relacionadas con Profiler están disponibles: Autoconnect Profiler y Deep Profiling Support.

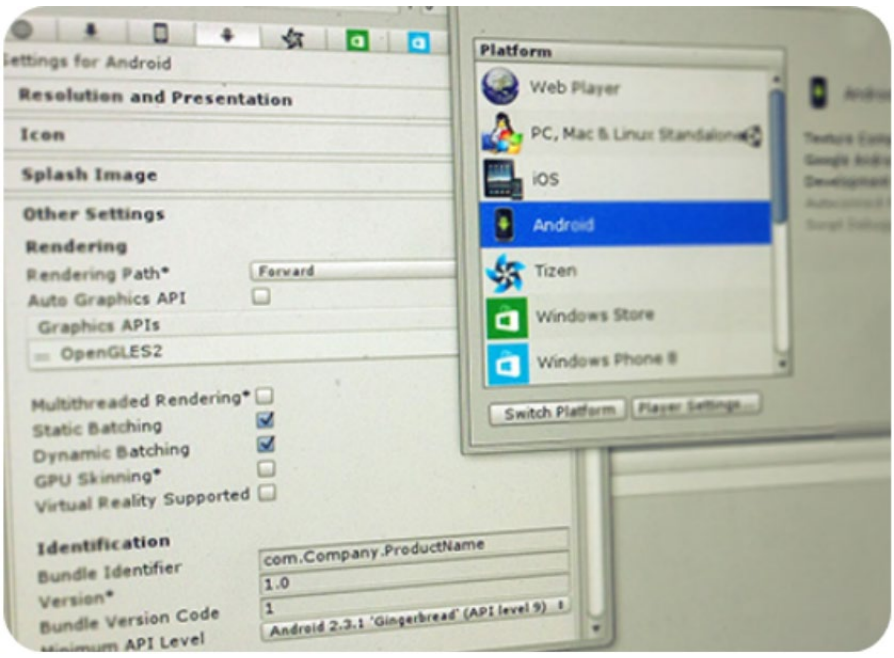
Cuando cambia el objetivo de Profiler a Editor, todas las muestras que estaban previamente ocultas bajo el marcador EditorLoop contribuyen a sus respectivas categorías. Esto significa que la información en el panel de detalles del módulo CPU Profiler y sus gráficos cambia significativamente. Para perfilar los tiempos de inicio del Editor, inicie el Editor con la opción de línea de Comando > Profiler > Enable.

Figura 3. Ventana de dialogo Build Settings



Para reducir el impacto que tiene la ventana del generador de perfiles en el rendimiento del editor, puede utilizar el generador de perfiles independiente, que abre la ventana del generador de perfiles en su propio proceso. Esto es especialmente útil si selecciona el Editor como el objetivo de creación de perfiles, o si está creando perfiles profundos en su aplicación, porque la ventana del generador de perfiles, normalmente, utiliza recursos que pueden sesgar los datos de rendimiento.

Figura 4. Ventana Editor Features



La mejor manera de obtener tiempos precisos sobre la aplicación es perfilarla en la plataforma final en la que se desea publicar, así se sabrá lo que puede afectar el rendimiento de la aplicación.

Sin embargo, puede llevar mucho tiempo crear la aplicación cada vez que se deseen mejorar elementos del rendimiento, por lo tanto, para evaluar rápidamente el rendimiento de la aplicación, se puede perfilar directamente en el modo Reproducir en el Editor. La creación de perfiles en el modo de reproducción no brinda un reflejo preciso de cómo se ve el rendimiento de la aplicación en un dispositivo real, pero sí es útil para verificar rápidamente si los cambios que realiza mejoran el rendimiento de la aplicación después de la creación de perfiles inicialmente en la plataforma final.

Unity Editor puede afectar el rendimiento de la aplicación, ya que usa los mismos recursos que la aplicación cuando se ejecuta en modo Play, por lo que también puede crear un perfil del Editor por separado para determinar qué recursos usa. Esto es particularmente útil si la aplicación solo ha sido diseñada para funcionar en el modo Reproducir, como para hacer películas.

e) Mejores prácticas para perfilar la aplicación

Cuando se crea un perfil de la aplicación, hay algunas cosas que puede hacer para garantizar la coherencia entre las sesiones de creación de perfiles y para asegurarse de que los procesos que utiliza Unity no afecten los datos de creación de perfiles:

Configuración de los módulos de perfilado

Agregue los módulos de Unity Profiler relacionados con el área que desea investigar a la ventana de la herramienta. Para agregar y quitar módulos al generador de perfiles, seleccione el menú desplegable en la parte superior izquierda de la ventana del generador de perfiles Profiler Modules. Se desplegará un cuadro de opciones donde podrá habilitar o deshabilitar los módulos que se deseen.

Habilitación de la configuración de pilas de llamadas

Evite el uso de perfiles profundos, ya que crea una gran sobrecarga cuando lo usa. Si desea ver más detalles sobre muestras con marcadores como GC.Alloc o JobFence.Complete, navegue hasta la barra de herramientas de la ventana Profiler y habilite la configuración de Pilas de llamadas. Esto proporciona la pila de llamadas completa de la muestra que le brinda la información que necesita sin incurrir en la sobrecarga de la creación de perfiles profundos.

Habilitar o deshabilitar Profiler

Utilice el acceso directo F9 para habilitar o deshabilitar Profiler. Puede utilizar este acceso directo para capturar datos de rendimiento sin necesidad de abrir la ventana. Si tiene abierto el generador de perfiles independiente y al usar este acceso directo se inicia la grabación en esta ventana.

Se puede conocer más acerca de la configuración del Profiler en el siguiente video:

Video 3. Configuración del Profiler

Configuración del Profiler

Síntesis del video: Configuración del Profiler
Es un video tutorial, que muestra el paso a paso para la creación de Configuración del Profiler.

1.2. Optimización de archivos de salida

La optimización comprende tanto técnicas como procedimientos, que deben hacerse desde el momento mismo de diseño y construcción de los elementos del videojuego. Como es sabido, los elementos del videojuego comprenden elementos gráficos y visuales en 2D y 3D, elementos de sonido, como voces de diálogos, efectos de sonido, música y sonido

ambiente, elementos de código, que son los conjuntos de archivos de código escritos por el desarrollador y los comprendidos en los paquetes de librerías importadas o integradas previamente al proyecto de videojuego.

Los archivos de salida son esencialmente los archivos finales que, en definitiva, formarán para del videojuego, y que, en el entorno de desarrollo de Unity, están almacenados al interior de la carpeta de proyecto, organizados en subcarpetas según su categoría o función.

Para este apartado, trataremos los archivos de salida, de los elementos característicamente más influyentes en el rendimiento.

1) Optimización del rendimiento gráfico

La optimización del rendimiento gráfico comprende el reto más importante del trabajo de optimización, siendo el aspecto gráfico el que puede tener el mayor nivel de exigencia en procesamiento, consumo de memoria y cálculo matemático. Los elementos gráficos tienen gran impacto de rendimiento en dos sistemas fundamentales: el GPU y el CPU. El primer principio de optimización de gráficos es la detección del problema de rendimiento en estas dos dimensiones, teniendo en cuenta que el proceso de optimización puede ser diferente.

El sistema de GPU

Está limitado por el Fillrate o tasa de relleno que determina, en esencia, la cantidad de píxeles que la tarjeta gráfica es capaz de dibujar o renderizar, procesar y almacenar, por cada unidad de tiempo.

El sistema de CPU

Está limitado por el número de Batches o lotes de información gráfica que necesita ser renderizada. En términos generales, se remite al cálculo por el procesador de la máquina y el almacenamiento de información en memoria RAM.

2) Optimización del CPU

El problema de rendimiento, en el CPU, se da por el número o cantidad de elementos presentes en la cola de renderizado (Batches Rendering). El proceso de optimización consiste en el control de la complejidad de los siguientes elementos gráficos:

Resolución de mapa de bits: el tamaño en resolución de las imágenes puede optimizarse de acuerdo con las necesidades gráficas. Se debe controlar el tamaño de resolución hasta un nivel que compense la calidad de apariencia y su peso en procesamiento.

Dibujo de mallas poligonales: los objetos de malla poligonal deben su complejidad por el número de triángulos que poseen y deben ser calculados, procesados y dibujados. Es necesario, entonces, reducir el número de polígonos / triángulos de una malla, al mismo tiempo que mantener el menor número de mallas poligonales en escena. Por otro lado, es necesario agrupar o unir mallas poligonales con pocos triángulos, dado que en el procesamiento por CPU resulta mayor esfuerzo el cálculo de múltiples mallas con un solo triángulo, que múltiples triángulos presentes en pocas mallas.

Dibujo de materiales y texturas: es importante reducir el número de materiales, en objetos o elementos similares en apariencia, tratando de generar atlas de imágenes de textura que puedan ser utilizados en un mismo material por diferentes objetos.

Efectos de sombreado, reflexión y luces: el cálculo de sombras, luces y efectos de reflexión, supone el cálculo matemático de cada uno de estos, en tiempo real, cada vez por segundo. En este tipo de optimización se utiliza el quemado de texturas (Baking Texture), que permite generar imágenes con el dibujado de píxeles de sombras, luces, y reflexiones en la imagen de textura, evitando el cálculo en tiempo real de estos efectos.

3) Optimización del GPU

En el GPU el problema de rendimiento se da por el ancho de banda que posee el sistema para procesar, analizar y dibujar o renderizar en pantalla (Fillrate). El proceso de optimización consiste en el control de la complejidad de los siguientes elementos gráficos:

Rendimiento de iluminación estática

De una forma similar con la técnica de quemado de texturas, puede optimizarse desde el editor de Unity 3D los efectos de iluminación. En este proceso se busca generar iluminación sin la necesidad de cálculos matemáticos, a partir del uso de la herramienta de lightmapping que permite hacer el quemado de texturas, al interior del motor, calculando sus propiedades una sola vez en elementos estáticos, haciendo más rápido el procesamiento general y alivianando la cantidad de información en el procesador y memoria.

Compresión de texturas y *mipmaps*

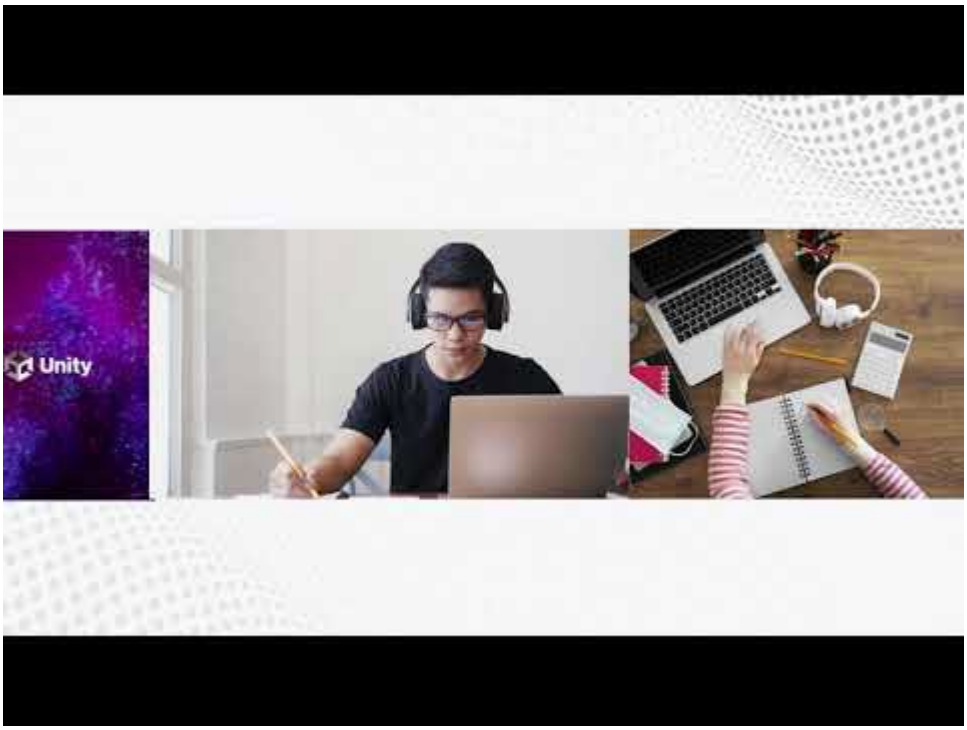
En términos generales la carga del procesamiento de imágenes lo determina las resoluciones de estas. El proceso de optimización puede involucrar dos técnicas principales: en primer lugar, la configuración de la propiedad de Compresión, desde el editor Unity 3D y, en segundo lugar, la activación de Generación de mipmaps para realizar un decrecimiento de la resolución de aquellas imágenes que se renderizarán en triángulos pequeños.

Complejidad de la geometría de un modelo 3D

La complejidad de un modelo tridimensional radica en la cantidad de triángulos que posea. En este sentido es necesario mantener un número reducido de triángulos, procurando no afectar su apariencia. Por otro lado, los vértices presentes en un modelo difieren en cantidad a los renderizados por el GPU. La razón es que los vértices, reales o geométricos, tienden a duplicarse según se encuentren en las costuras del modelo o posean múltiples normales o coordenadas UV.

Mediante el siguiente video, se informarán algunos elementos clave para el proceso de optimización de assets:

Video 4. Optimización de Assets



Optimización de Assets

Síntesis del video: Optimización de Assets
Es un video tutorial, informarán algunos elementos clave para el proceso de optimización de Assets:

1.3. Optimización de archivos de código

La optimización de código se lleva a cabo en el mismo proceso de escritura del código fuente, buscando obtener el mayor rendimiento en el procesamiento de instrucciones, además del peso de los archivos. Por otro lado, es necesario analizar la necesidad o no de crear varios scripts para cada acción o implementarlas en un solo script.

De cualquier forma, desde el diseño algorítmico se comienza a definir el esquema lógico y las relaciones entre diferentes funciones, clases, eventos y variables.

Optimización de scripts en juegos de Unity.

Cuando el juego se ejecuta, la unidad central de procesamiento (CPU) del dispositivo ejecuta las instrucciones. Cada fotograma del juego requiere que se ejecuten millones de estas instrucciones de CPU y para mantener una velocidad de fotogramas uniforme, la CPU debe llevar a cabo sus instrucciones dentro de un período de tiempo establecido. Cuando la CPU no puede llevar a cabo todas sus instrucciones a tiempo, el juego puede ralentizarse, tartamudear o congelarse.

Muchas cosas pueden hacer que la CPU tenga demasiado trabajo por hacer. Los ejemplos podrían incluir código de renderizado exigente, simulaciones físicas muy complejas o demasiadas devoluciones de llamada de animación. Este apartado se centra solo en una de estas razones: problemas de rendimiento de la CPU causados por el código que se escribe en los scripts.

Se aprenderá cómo los scripts se convierten en instrucciones de la CPU, qué puede hacer estos generen una cantidad excesiva de trabajo para la CPU y cómo solucionar los problemas de rendimiento causados por el código en los scripts.

Diagnóstico de problemas de código

Los problemas de rendimiento causados por demandas excesivas en la CPU pueden manifestarse como velocidades de cuadro bajas, rendimiento desigual o congelamientos intermitentes. Sin embargo, otros problemas pueden causar síntomas similares. Si el juego tiene problemas de rendimiento como este, lo primero que debemos hacer es usar la ventana Profiler de Unity para establecer si los problemas de rendimiento se deben a que la CPU no puede completar sus tareas a tiempo. Una vez establecido esto, debemos determinar si los scripts de usuario son la causa del problema, o si el problema es causado por alguna otra parte del juego, física compleja o animaciones, por ejemplo.

Modo de compilación en Unity 3D

Para comprender por qué el código puede no estar funcionando bien, primero se debe entender qué sucede cuando Unity compila el juego. Saber lo que sucede detrás de escena ayudará a tomar decisiones informadas sobre cómo se puede mejorar el rendimiento del juego, a saber:

a) El proceso de compilación

Cuando se construye el juego, Unity empaqueta todo lo necesario para ejecutarlo en un programa que puede ser ejecutado por el dispositivo objetivo. Las CPU solo pueden ejecutar código escrito en lenguajes muy simples conocidos como código máquina o código nativo; no pueden ejecutar código escrito en lenguajes más complejos como C #. Esto significa que Unity debe traducir el código a otros idiomas, a este proceso se le llama compilación.

Unity primero compila los scripts en un lenguaje llamado Common Intermediate Language (CIL) que es fácil de compilar en una amplia gama de diferentes lenguajes de código nativo. Luego, el CIL se compila en código nativo para el dispositivo de destino específico. Este segundo paso ocurre cuando se construye el juego (conocido como compilación anticipada o compilación AOT), o en el dispositivo de destino, justo antes de que se ejecute el código (conocido como compilación justo a tiempo o compilación JIT). Si el juego usa compilación AOT o JIT generalmente depende del hardware de destino.

b) Relación entre el código fuente y el código compilado

El código que aún no ha sido compilado se conoce como código fuente, este determina la estructura y el contenido del código compilado.

En su mayor parte, un código fuente bien estructurado y eficiente dará como resultado un código compilado bien estructurado y eficiente. Sin embargo, es útil saber un poco sobre

el código nativo para entender mejor por qué algunos códigos fuente se compilan en un código nativo más eficiente.

En primer lugar, algunas instrucciones de la CPU tardan más en ejecutarse que otras. Un ejemplo de esto es calcular una raíz cuadrada. Este cálculo requiere más tiempo para que la CPU se ejecute que, por ejemplo, multiplicar dos números. La diferencia entre una sola instrucción de CPU rápida y una sola instrucción de CPU lenta es realmente muy pequeña, pero es útil para nosotros entender que, fundamentalmente, algunas instrucciones son simplemente más rápidas que otras.

Lo siguiente que se debe entender es que algunas operaciones que parecen muy simples en el código fuente pueden ser sorprendentemente complejas cuando se compilan en código. Un ejemplo de esto es insertar un elemento en una lista, pues se necesitan muchas más instrucciones para realizar esta operación que, por ejemplo, acceder a un elemento de una matriz por índice. Nuevamente, cuando se considera un ejemplo individual, se habla de una pequeña cantidad de tiempo, pero es importante entender que algunas operaciones dan como resultado más instrucciones que otras.

Comprender estas ideas ayudará a entender por qué algunos códigos funcionan mejor que otros, incluso cuando ambos ejemplos hacen cosas bastante similares. Un conocimiento previo limitado de cómo funcionan las cosas en un nivel bajo puede ayudar a escribir juegos que funcionen bien.

Comunicación en tiempo de ejecución entre el código de Unity Engine y el código de secuencia de comandos

Es útil comprender que las secuencias de comandos escritas en C # se ejecutan de una manera ligeramente diferente al código que forma gran parte de Unity Engine. La mayor parte de la funcionalidad principal de Unity Engine está escrita en C ++ y ya se ha compilado en código nativo. Este código de motor compilado es parte de la instalación de Unity.

El código compilado en CIL, como el código fuente, se conoce como código administrado. Cuando el código administrado se compila en código nativo, se integra con algo llamado tiempo de ejecución administrado. El tiempo de ejecución administrado se

encarga de cosas como la administración automática de la memoria y las verificaciones de seguridad para garantizar que un error en el código resulte en una excepción en lugar de que el dispositivo se bloquee.

Cuando la CPU cambia entre el código del motor en ejecución y el código administrado, se debe trabajar para configurar estas comprobaciones de seguridad. Cuando se devuelven datos del código administrado al código del motor, es posible que la CPU deba realizar un trabajo para convertir los datos del formato utilizado por el tiempo de ejecución administrado al formato que necesita el código del motor.

Esta conversión se conoce como clasificación y aquí una vez más, la sobrecarga de una sola llamada entre el código administrado y el del motor no es particularmente costosa, pero es importante entender que este costo existe.

Causas del código de bajo rendimiento

Ahora que se comprende lo que le sucede a el código cuando Unity compila y ejecuta el juego, se puede entender que cuando el código funciona mal es porque crea demasiado trabajo para la CPU en tiempo de ejecución. A continuación, se consideran las diferentes razones de esto:

a) La primera posibilidad

Que el código sea un desperdicio o esté mal estructurado. Un ejemplo de esto podría ser el código que realiza la misma llamada de función repetidamente cuando solo podría realizar la llamada una vez con lo que se cubrirán varios ejemplos comunes de estructura deficiente y mostrará soluciones de ejemplo.

b) La segunda posibilidad

Que el código parece estar bien estructurado, pero hace llamadas innecesariamente costosas en rendimiento a otro código. Un ejemplo de esto podría ser el código que da como resultado llamadas no necesarias entre el código administrado y el del motor.

c) La tercera posibilidad

Que el código sea eficiente, pero que se llame cuando no es necesario. Un ejemplo de esto podría ser el código que simula la línea de visión de un enemigo. El código en sí puede funcionar bien, pero es un desperdicio ejecutar este código cuando el jugador está muy lejos del enemigo.

d) La cuarta posibilidad

Que el código sea simplemente demasiado exigente. Un ejemplo de esto podría ser una simulación muy detallada en la que una gran cantidad de agentes utilizan IA compleja. Si se han agotado otras posibilidades y optimizado este código tanto como se puede, es posible que simplemente se necesite rediseñar el juego para hacerlo menos exigente: por ejemplo, falsificar elementos de la simulación en lugar de calcularlos.

Mejorar el rendimiento del código

Una vez que se ha establecido que los problemas de rendimiento en el juego se deben al código, se debe pensar detenidamente cómo resolver estos problemas. La optimización de una función exigente puede parecer un buen punto de partida, pero es posible que la función en cuestión ya sea óptima como puede ser y simplemente sea cara por naturaleza. En lugar de cambiar esa función, puede haber un pequeño ahorro de eficiencia que podemos hacer en un script que usan cientos de GameObjects y que da un aumento de rendimiento mucho más útil. Además, mejorar el rendimiento de la CPU del código puede tener un costo: los cambios pueden aumentar el uso de la memoria o descargar el trabajo a la GPU.

Por estos motivos, se define una serie de sugerencias para mejorar el rendimiento del código, con ejemplos de situaciones en las que se pueden aplicar. Al igual que con toda la optimización del rendimiento, no existen reglas estrictas y rápidas. Lo más importante que se debe hacer es perfilar el juego, comprender la naturaleza del problema, experimentar con diferentes soluciones y medir los resultados de los cambios.

Escribir código eficiente

Escribir y estructurar sabiamente el código eficiente puede conducir a mejoras en el rendimiento del juego. Si bien los ejemplos que se muestran están en el contexto de un juego de Unity, estas sugerencias de mejores prácticas generales no son específicas de los proyectos de Unity o las llamadas a la API de Unity.

Código fuera de bucles

Los bucles son un lugar común para que se produzcan ineficiencias, especialmente cuando están anidados. Las ineficiencias pueden sumarse si están en un bucle que se ejecuta con mucha frecuencia, especialmente si este código se encuentra en muchos GameObjects del juego. En el siguiente ejemplo simple, el código itera a través del ciclo cada vez que se llama a Update(), independientemente de si se cumple o no la condición.

```
void Update()
{
    for (int i = 0; i < myArray.Length; i++)
    {
        if (exampleBool)
        {
            ExampleFunction(myArray[i]);
        }
    }

}
```

Con un simple cambio, el código itera a través del ciclo solo si se cumple la condición.

```
void Update()
{
   if (exampleBool)
   {
      for (int i = 0; i < myArray.Length; i++)
      {
         ExampleFunction(myArray[i]);
      }
   }
}
```

Este es un ejemplo simplificado, pero ilustra un ahorro real que se puede hacer. Se debería examinar el código en busca de lugares en los que se han estructurado mal nuestros bucles.

Considerar si el código debe ejecutar cada cuadro

Update() es una función que Unity ejecuta una vez por cuadro. Update() es un lugar conveniente para colocar el código que debe llamarse con frecuencia o el código que debe responder a cambios frecuentes. Sin embargo, no es necesario que todo este código ejecute todos los fotogramas. Sacar el código de Update() para que se ejecute solo cuando sea necesario puede ser una buena forma de mejorar el rendimiento.

Ejecute código solo cuando las cosas cambien

Ahora se verá un ejemplo muy simple de optimización del código para que solo se ejecute cuando las cosas cambien. En el siguiente código, DisplayScore () se llama en Update(). Sin embargo, es posible que el valor de la puntuación no cambie con cada fotograma. Esto significa que estamos llamando innecesariamente a DisplayScore ().

```
private int score;

public void IncrementScore(int incrementBy)
{
    score += incrementBy;
}

void Update()
{
    DisplayScore(score);
}
```

Con un simple cambio, ahora se asegura que DisplayScore () se llame solo cuando el valor de la puntuación haya cambiado.

```
private int score;

public void IncrementScore(int incrementBy)
{
    score += incrementBy;
    DisplayScore(score);
}
```

De nuevo, el ejemplo anterior está deliberadamente simplificado, pero el principio es claro: si se aplica este enfoque en todo el código, es posible ahorrar recursos de CPU.

Ejecutar código cada [x] fotogramas

Si el código necesita ejecutarse con frecuencia y no puede ser activado por un evento, eso no significa que deba ejecutar cada fotograma. En estos casos, podemos optar por ejecutar código cada [x] fotogramas.

En este código de ejemplo, una función costosa se ejecuta una vez por cuadro.

```
void Update()
{
    ExampleExpensiveFunction();
}
```

De hecho, sería suficiente para las necesidades ejecutar este código una vez cada 3 fotogramas. En el siguiente código, se usa el operador de módulo para asegurarse que la función costosa se ejecute solo en cada tercer fotograma.

intervalo int privado = 3;

```
void Update()
{
    if (Time.frameCount % interval == 0)
    {
        ExampleExpensiveFunction();
    }
```

Un beneficio adicional de esta técnica es que es muy fácil distribuir un código costoso en marcos separados, evitando picos. En el siguiente ejemplo, cada una de las funciones se llama una vez cada 3 fotogramas y nunca en el mismo fotograma.

intervalo int privado = 3;

```
void Update()
{
    if (Time.frameCount % interval == 0)
    {
        ExampleExpensiveFunction();
    }
    else if (Time.frameCount % interval == 1)
    {
        AnotherExampleExpensiveFunction();
    }
```

```
    }
```

De hecho, sería suficiente para las necesidades ejecutar este código una vez cada 3 fotogramas. En el siguiente código, se usa el operador de módulo para asegurarse que la función costosa se ejecute solo en cada tercer fotograma.

```
    private int interval = 3;

    void Update()
    {
        if (Time.frameCount % interval == 0)
        {
            ExampleExpensiveFunction();
        }
    }
```

Un beneficio adicional de esta técnica es que es muy fácil distribuir el código costoso en marcos separados, evitando picos. En el siguiente ejemplo, cada una de las funciones se llama una vez cada 3 fotogramas y nunca en el mismo fotograma.

```
    private int interval = 3;

    void Update()
    {
        if (Time.frameCount % interval == 0)
        {
            ExampleExpensiveFunction();
        }
        else if (Time.frameCount % interval == 1)
        {
            AnotherExampleExpensiveFunction();
        }
}
```

Utilizar el almacenamiento en caché

Si el código llama repetidamente a funciones costosas que devuelven un resultado y luego descarta esos resultados, esta puede ser una oportunidad para la optimización. El almacenamiento y la reutilización de referencias a estos resultados pueden resultar más eficaz. Esta técnica se conoce como almacenamiento en caché.

En Unity es común llamar a GetComponent () para acceder a los componentes. En el siguiente ejemplo, se llama a GetComponent () en Update() para acceder a un componente de Renderer antes de pasarlo a otra función. Este código funciona, pero es ineficaz debido a la llamada repetida a GetComponent ().

```
    void Update()
    {
        Renderer myRenderer = GetComponent<Renderer>();
        ExampleFunction(myRenderer);
}
```

El siguiente código llama a GetComponent () solo una vez, ya que el resultado de la función se almacena en caché. El resultado almacenado en caché se puede reutilizar en Update() sin más llamadas a GetComponent ().

```
    private Renderer myRenderer;

    void Start()
    {
        myRenderer = GetComponent<Renderer>();
    }

    void Update()
    {
        ExampleFunction(myRenderer);
    }
```

Se debería examinar el código en busca de casos en los que se hacen llamadas frecuentes a funciones que devuelven un resultado. Es posible, entonces, reducir el costo de estas llamadas utilizando el almacenamiento en caché.

Usar agrupación de objetos

Generalmente es más costoso crear instancias y destruir un objeto que desactivarlo y reactivarlo. Esto es especialmente cierto si el objeto contiene código de inicio, como llamadas a GetComponent () en una función Awake () o Start (). Si se necesita generar y deshacerse de muchas copias del mismo objeto, como balas en un juego de disparos, entonces se puede obtener beneficios de la agrupación de objetos.

La agrupación de objetos es una técnica en la que, en lugar de crear y destruir instancias de un objeto, los objetos se desactivan temporalmente y luego se reciclan y reactivan según sea necesario. Aunque es una técnica muy conocida para administrar el uso de la memoria, la agrupación de objetos también puede ser útil como técnica para reducir el uso excesivo de la CPU.

Ejecutar código solo cuando es necesario ejecutarlo

Hay un dicho en programación: "el código más rápido es el código que no se ejecuta". A menudo, la forma más eficaz de resolver un problema de rendimiento es no utilizar una técnica avanzada, es simplemente eliminar el código que no necesita estar allí en primer lugar.

Eliminación (Culling): Unity contiene un código que comprueba si los objetos están dentro del cono de vista de una cámara. Si no están dentro del cono de vista de una cámara, el código relacionado con la representación de estos objetos no se ejecuta; el término para esto es sacrificio frustum.

Se puede adoptar un enfoque similar al código en los scripts. Si se tiene un código que se relaciona con el estado visual de un objeto, es posible que no se necesite ejecutar este código cuando el jugador no pueda ver el objeto. En una escena compleja con muchos objetos, esto puede resultar en ahorros de rendimiento considerables.

En el siguiente código de ejemplo simplificado, se tiene un ejemplo de un enemigo patrullando. Cada vez que se llama a Update(), el script que controla a este enemigo llama a dos funciones de ejemplo: una relacionada con el movimiento del enemigo y otra relacionada con su estado visual.

```
void Update()
{
    UpdateTransformPosition();
    UpdateAnimations();
}
```

En el siguiente código, ahora se verificará si el renderizador del enemigo está dentro del frustum de cualquier cámara. El código relacionado con el estado visual del enemigo se ejecuta solo si el enemigo es visible.

```
private Renderer myRenderer;

void Start()
{
```

```
    myRenderer = GetComponent<Renderer>();
}

void Update()
{
    UpdateTransformPosition();

    if (myRenderer.isVisible)
    {
        UpateAnimations();
    }
}
```

Deshabilitar el código de objetos no visibles: si se sabe que determinados objetos de la escena no son visibles en un punto concreto del juego, se pueden desactivar manualmente. Cuando se tenga menos certeza y se necesite calcular la visibilidad, se podría usar un cálculo aproximado (por ejemplo, verificar si el objeto detrás del reproductor), funciones como OnBecameInvisible () y OnBecameVisible (), o un raycast más detallado. La mejor implementación depende en gran medida del juego, y la experimentación y la creación de perfiles son esenciales.

Nivel de detalle

También conocido como LOD, es otra técnica común de optimización de renderizado. Los objetos más cercanos al jugador se renderizan con total fidelidad utilizando mallas y texturas detalladas; los objetos distantes las utilizan menos detalladas.

Se puede usar un enfoque similar con el código. Por ejemplo, al tener un enemigo con un script de IA que determina su comportamiento. Parte de este comportamiento puede involucrar operaciones costosas para determinar qué puede ver y escuchar, y cómo debería reaccionar a esta entrada. Se podría usar un sistema de nivel de detalle para habilitar y deshabilitar estas costosas operaciones en función de la distancia del enemigo al jugador.

En una escena con muchos de estos enemigos, se haría un ahorro de rendimiento considerable si solo los enemigos más cercanos realizan las operaciones más costosas.

La API CullingGroup de Unity permite conectarse al sistema LOD de Unity para optimizar el código. La página del manual de la API de CullingGroup contiene varios ejemplos de cómo se puede utilizar en el juego. Como siempre, se debe probar, perfilar y encontrar la solución adecuada para el juego.

2. Documentación del proceso de integración de assets

En este apartado se observará la información pertinente que, a manera de manual de integración, podrá servir para documentar el proceso de creación y configuración de un proyecto de videojuego, y el consecuente proceso de integración de assets y recursos del videojuego. En primera instancia se observarán los detalles de creación del proyecto y, posteriormente, los detalles de integración de los sistemas de assets y posteriores procesos de optimización y ajuste preliminar.

Es importante que, en esta fase de desarrollo, se tengan detalles de los requerimientos del proyecto, es decir, una base documental que, en esencia, es el documento de diseño del videojuego o Game Design Document.

Los detalles de requerimientos podrán ofrecer una orientación sobre las características esenciales que deberá tener la carpeta de proyecto de Unity 3D y, por consiguiente, de la manera como se deberá configurar y optimizar cada asset a integrar o integrado en la plataforma.

Por otro lado, brindará información pertinente a cada miembro del equipo de trabajo, sobre los detalles y paso a paso de los procesos de configuración. En términos generales, esta documentación se podrá definir y pensar como una bitácora de viaje donde se registran los detalles de cada evento y servirá, posteriormente, como referente para el análisis de errores y fallas, y por consiguiente de cambios y ajustes.

2.1. Proceso de creación y configuración del proyecto

Al crear un proyecto de aplicación dentro del entorno de desarrollo de Unity, es necesario realizar una serie de pasos ordenados y consecuentes que podrán determinar las características esenciales para el buen funcionamiento y configuración del proyecto.

Sin embargo, se hace necesario tener una serie de documentos claramente diseñados y parametrizados que permitan delimitar los detalles de configuración del proyecto, que involucra variables como plataformas de publicación, ubicación y organización de archivos, configuración de parámetros de compilación e incluso de perfiles de optimización.

Para mayor ilustración sobre la creación de la carpeta de proyecto, se invita a ver el siguiente video.

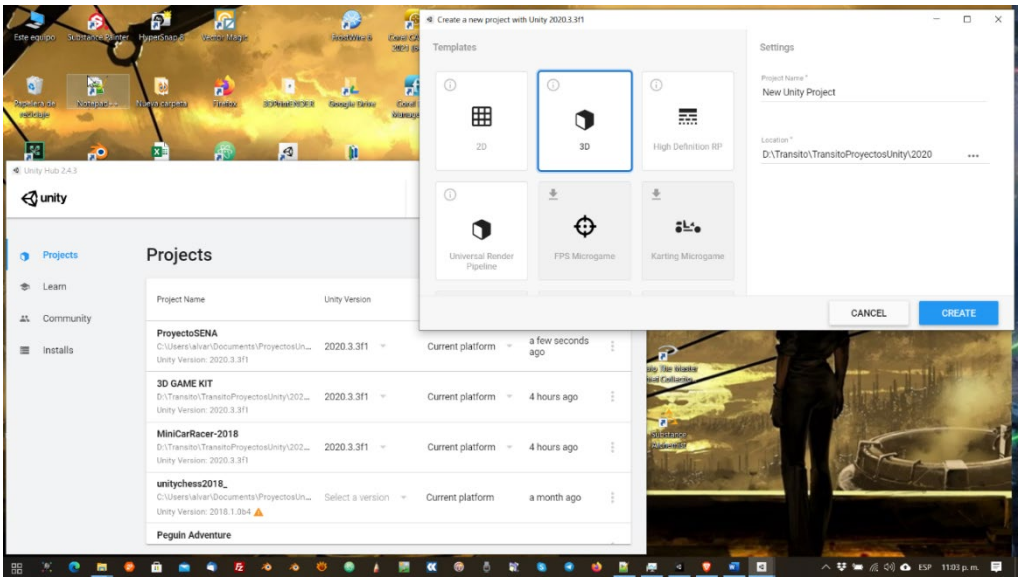
Video 5. Creación carpeta de proyecto

Creación carpeta de proyecto

Síntesis del video: Creación carpeta de proyecto
Es un video tutorial, informarán algunos elementos clave para la creación de la carpeta del proyecto.

Configuración predeterminada / personalizada de un proyecto en Unity: cuando se crea un proyecto nuevo en Unity, se abre una ventana que funciona como asistente de creación de proyecto, solicitando una primera información de base, como el nombre del proyecto, su ubicación en el sistema de carpetas y su plantilla de base, que define si será 2D o 3D.

Figura 5. Ventana del asistente de creación de proyecto



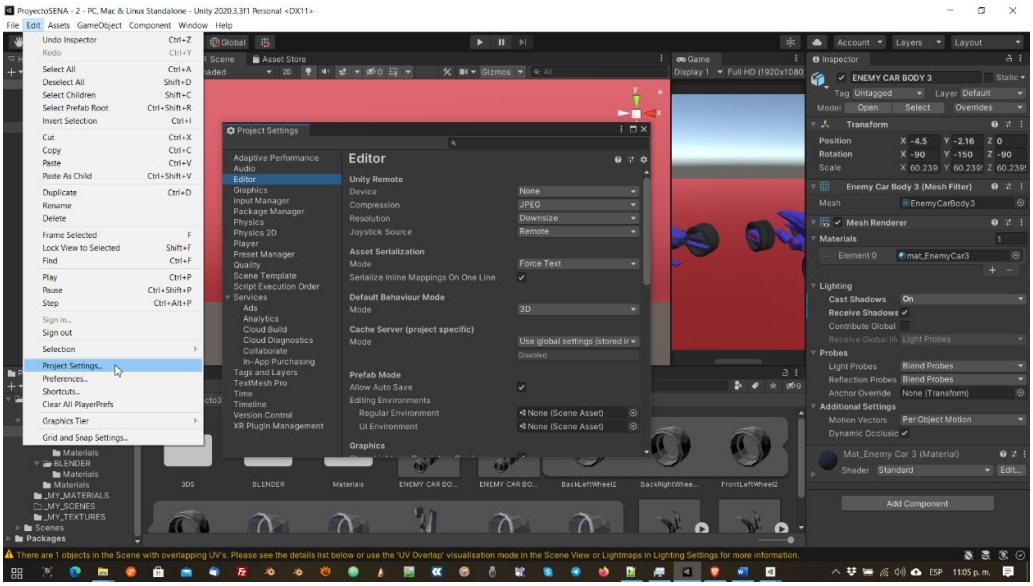
Crear proyecto 2D vs. 3D

Una vez se eligen las opciones, principalmente la plantilla 2D o 3D, se comienza la definición de cada parámetro. Es importante tener claridad de las dimensiones del videojuego, es decir 2D o 3D, porque una vez se crea con este parámetro, se cargarán configuraciones predeterminadas para una u otra dimensión de juego.

Ahora, se invita a ver el video para la configuración compilación.

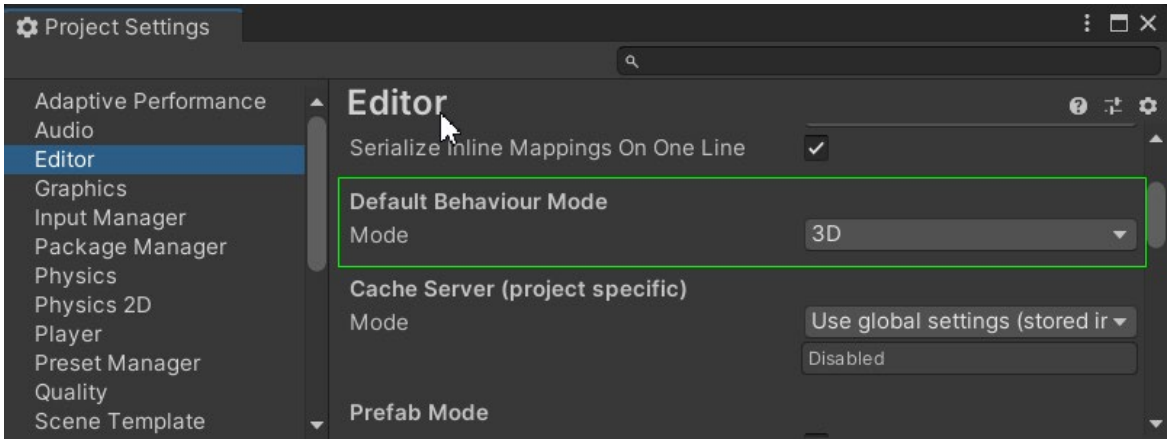
Sin embargo, puede darse la circunstancia que deba cambiarse esta configuración y realizar este cambio sin mayores dificultades, teniendo en cuenta que cambiarán aspectos internos de la carpeta de proyecto. Puede cambiarse de 2D a 3D o viceversa. Para realizar este cambio se recurre a la siguiente opción de ruta en menú Edit (menú: Edit > Project Settings > Editor), tal cual como se muestra en la siguiente figura.

Figura 6. Ventana de diálogo para la configuración del modo de plantilla



En el Inspector, aparecerá la opción Default Behavior Mode y podrá elegir el modo específico de comportamiento.

Figura 7. Detalle de la ventana de diálogo para la configuración del modo de plantilla



Configuración predeterminada de un proyecto 2D y 3D en Unity

Como se anotó anteriormente, la configuración de cada plantilla es diferente y realiza ajustes preestablecidos de manera específica para cada plantilla. A continuación, se podrán observar los aspectos básicos de configuración según el modo escogido.

Para la configuración predeterminada de Unity en modo 2D

Las imágenes se integran e importan como Sprites.

La vista en el Panel o Visor de Edición aparece como 2D.

Se habilita la herramienta Sprite Packer.

Los game objects, no presentan luz discrecional.

La posición de la cámara aparece en modo de vista ortográfica y en una posición de coordenadas X = 0, Y = 0, Z = -10.

En la ventana de iluminación

La opción de Skybox esta deshabilitado para nuevas escenas.

La opción de Ambient Source es un color sólido gris oscuro (RGB: 54, 58, 66).

La opción de Precomputed Realtime GI está desactivada.

La opción de Backed GI está desactivada.

La opción de Auto Building está desactivada.

Para la configuración predeterminada de Unity en modo 3D

Al importar las imágenes del videojuego, Unity las integrará como texturas.

La vista en el Panel o Visor de Edición aparece como 3D.

Se deshabilita la herramienta Sprite Packer.

Los game objects presentan luz discrecional en tiempo real que permite visualizar volumetría.

La posición de la cámara aparece en modo de vista perspectivas y en una posición de coordenadas X = 0, Y = 1, Z = -10.

En la ventana de iluminación

La opción de Skybox usa el default Skybox Material.

La opción de Ambient Source está preconfigurada en Skybox.

La opción de Precomputed realtime GI está activa.

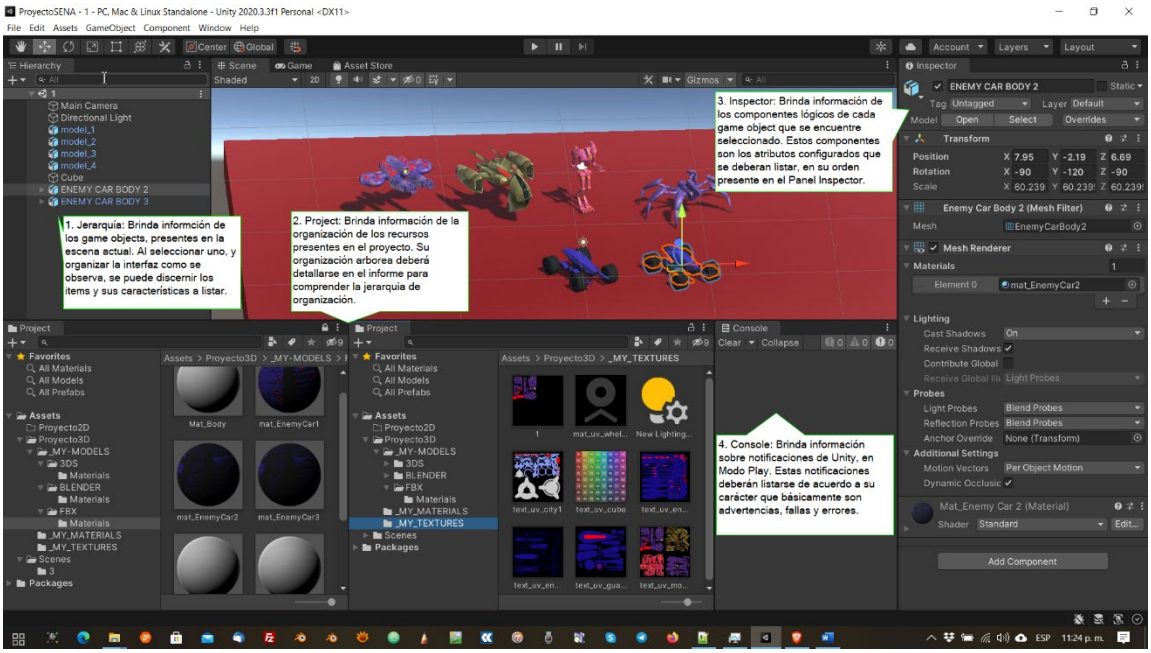
La opción de Backed GI está activa.

La opción de Auto-Building está activa.

Es importante tener claridad que si bien el proyecto puede ser de modo 2D en la plantilla, puede contener elementos de características 3D como modelos tridimensionales poligonales. O puede configurarse en modo 3D en la plantilla y contener elementos bidimensionales como Sprites.

Ahora, se verá cómo funciona la integración y caracterización de assets mediante el siguiente video. Integracion y caracterizacion de assets

Figura 8. Paneles de configuración de atributos, integración de componentes y organización de assets



2.2. Detalles de configuración de la plataforma de publicación y compilación

Cuando se configura el proyecto en sus detalles iniciales, los primeros aspectos a resolver son la plataforma de compilación y publicación. Sin embargo, Unity permite cambiar la plataforma definida, haciéndolo a través de la opción Switch Platform.

A continuación, se expone el video tutorial de cómo se realiza el cambio de plataforma.

Una vez se determina la plataforma definitiva e incluso las posibilidades de conversión, se hace necesario documentar los detalles de configuración, asegurándose de concluir cada aspecto inherente a la compilación y plataforma de publicación.

Detalles de compilación Standalone: la configuración de compilación independiente para PC, Mac y Linux contiene opciones que se pueden utilizar para configurar y comenzar el proceso de compilación de la aplicación en estas plataformas. Esta contiene configuraciones para crear compilaciones de desarrollo, así como publicar su compilación final. Para acceder a la ventana Configuración de compilación, se debe ir a File > Build Settings. Una vez que se especifique la configuración de compilación, se selecciona Build para crear la compilación, o seleccionar Build and Run para crear y ejecutar la compilación en la plataforma que especifique.

Tabla 3. Configuraciones para compilación

Configuración	Descripción
Plataforma destino	
Windows	Compilación para Windows.
macOS X (Not available on Linux)	Compilación para macOS.
Linux (not available on Mac)	Compilación para Linux.
Arquitectura/ Not available on macOS	
x86	32-bit CPU.
x86_64	64-bit CPU.
x86 + x86_64 (Universal)	Todos los dispositivos CPU para Linux.
Server Build	Habilitar esta configuración para crear el reproductor para uso del servidor y sin elementos

Configuración	Descripción
	visuales (sin cabeza), sin la necesidad de ninguna opción de línea de comando. Cuando se habilita esta configuración, Unity crea scripts administrados con la definición de UNITY_SERVER, lo que significa que se puede escribir código específico del servidor para la aplicación. También se puede compilar la versión de Windows como una aplicación de consola para acceder a stdin y stdout. Los registros de Unity van a la salida estándar de forma predeterminada.
Copy PDB files(Windows only)	Habilitar esta configuración para incluir archivos de base de datos de programas de Microsoft (.pdb) en el reproductor independiente integrado. Los archivos .pdb contienen información de depuración de la aplicación que puede utilizar para depurar la aplicación. La copia de archivos .pdb puede aumentar el tamaño de su reproductor, por lo que debe deshabilitar esta configuración para las compilaciones destinadas a la publicación. Esta configuración está deshabilitada de forma predeterminada.
Create Visual Studio Solution(Windows only)	Habilitar esta configuración para generar archivos de la solución de Visual Studio para su proyecto, de modo que pueda crear su ejecutable final en Visual Studio.

Configuración	Descripción
Create Xcode Project (Mac Only)	Habilitar esta configuración para generar un proyecto de Xcode para que pueda construir su paquete de aplicación final en Xcode. Xcode tiene soporte integrado para la firma de código y la carga de la aplicación en la Mac App Store.
Development Build	Habilitar esta configuración para incluir símbolos de depuración de secuencias de comandos e incluir el generador de perfiles en su compilación. Cuando se habilita esta configuración, se establece la definición de scripting DEVELOPMENT_BUILD. Debe utilizar esta opción cuando desee probar la aplicación.
Autoconnect profiler	Requiere que se habilite la opción Development Build. Cuando habilita esta configuración, Unity Profiler se conecta automáticamente a su compilación.
Deep Profiling Support	Requiere que se habilite la opción Development Build. Deep Profiling Support permite que Unity Profiler registre datos más detallados instrumentando cada llamada de función. Se debe habilitar Deep Profiling might slow down script execution.
Script debugging	Requiere que se habilite la opción Development Build. Cuando habilita esta configuración, Unity agrega símbolos de depuración a su código de secuencia de comandos.
Scripts Only Build	Requiere que se habilite la opción Development Build. Cuando habilita esta configuración, puede reconstruir solo los scripts para la aplicación

Configuración	Descripción
	mientras deja intactos los archivos de datos de una compilación que ha ejecutado previamente. La compilación de scripts solo mejora significativamente los tiempos de iteración si solo está cambiando el código en la aplicación. Se debe compilar todo el proyecto una vez antes de poder utilizar esta configuración.
x86	32-bit CPU.

2.3. Lista de chequeo de ítems y configuraciones

Hasta el momento se han definido los conceptos relativos a los aspectos esenciales de optimización de recursos del videojuego y de configuración de la compilación y plataforma de publicación. Es necesario, a partir de este punto, realizar una serie de listas de chequeo que permitan determinar el cumplimiento de los requisitos del videojuego y los detalles fundamentales de configuración.

Básicamente se trata de enumerar en primer lugar los ítems, determinar si se encuentran presentes o no en la carpeta de proyecto y verificar si su integración se logró con éxito en un primer intento, y de no ser así, describir en los detalles el proceso que se llevó a cabo para su optimización y/o correcta integración. Como componente de bitácora, la lista tendrá, además, la ruta de acceso al recurso, y de ser posible, capturas de los paneles de Inspector y Jerarquía para visualizar gráficamente los detalles de su configuración y/o integración de componentes de juego como archivos de código o efectos.

Por otro lado, se listan las características de compilación, mostrando incluso de forma gráfica mediante captura, que en efecto se realizó la configuración de acuerdo con los requerimientos, ayudando a mismo tiempo al equipo de trabajo, a entender la ruta de acceso e interacción con las herramientas de compilación (Build Settings).

A continuación, se presenta un ejemplo de lista de chequeo de integración y configuración de assets, donde puede observarse el orden de caracterización del ítem, de lo general (categoría) a lo particular (criterio / atributo). Y en un espacio de la lista, lo correspondiente al listado de errores y ajustes, de ser necesario.

Figura 9. Ejemplo lista de chequeo de integración y configuración de assets

Lista de chequeo de integración y configuración de *assets*				
CATEGORÍA/ITEM		Modelos 3DS/BuggyEnemy.3ds		
REFERENCIA				
N.°	CRITERIO/ATRIBUTO	**CUMPLIDO**		**OBSERVACIONES**
		SÍ	NO	
1	Poligonalidad LowPoly	X		El modelo presenta configuración LOWPOLY: 2.832 tris
2	Materiales	X		1 material: mat_buggy_enemy
3	Texturas	X		1 textura: text_uv_buggy_enemy.png (512px X 512px)
4	Mapeado UV	X		1 UVMap
N°	ASPECTO/ERROR	**CORREGIDO**		**DETALLES**
		SÍ	NO	
1	Normales/Invertidas	X		El modelo presentaba las normales invertidas de la carrocería, y no era posible verla en el motor. Se corrigió desde la aplicación de origen.

2.4. Lista de errores y métricas

En apartados anteriores se describió la metodología básica para recurrir a herramientas propias de Unity para perfilar y obtener datos de rendimiento en tiempo de ejecución clasificados por módulos funcionales. Desde esta herramienta de análisis e información y otras que se describirán a continuación, se pueden establecer listas de errores y métricas, que representan en orden cronológico, toda la serie de datos pertinentes para determinar su nivel de error falla o influencia en rendimiento. Esta información se genera de manera automática y su diferencia entre versiones de datos de un mismo indicador analizado, radicará en la forma como se haya logrado optimizar.

Por otro lado, además de los datos de error y falla, se describen los correspondientes a los de ajuste y optimización que se hayan podido caracterizar, pero que obedecerán a procesos manuales y elaborados por determinado miembro del equipo de desarrollo.

Lista de eventos desde Log.Files: puede haber momentos durante el desarrollo en los que necesite obtener información de los registros del reproductor independiente que ha creado, el dispositivo de destino, el Administrador de paquetes o el Editor. Los archivos de registro son útiles cuando se ha experimentado un problema, para averiguar exactamente dónde ocurrió el problema.

Todos los mensajes, advertencias y errores escritos en la ventana de la consola en el Editor también se escriben en estos archivos de registro. Se pueden escribir los propios mensajes en la consola y los archivos de registro utilizando la clase Debug.

En macOS, se puede acceder a los registros del reproductor, el administrador de paquetes y el Editor a través de la utilidad Console.app.

En Windows, los registros del Administrador de paquetes y del Editor se ponen en carpetas que no se muestran en el Explorador de Windows de forma predeterminada.

Administrador de empaquetación: para ver el registro de Package Manager, se debe navegar hasta la carpeta de registro de Unity:

Tabla 4. Registro de Package Manager

Sistema operativo	Log Files
Linux	~/.config/unity3d/upm.log
Macos	~/library/logs/unity/upm.log también puede utilizar la utilidad console.app desde su la carpeta /applications/utilities para buscar el archivo log.

Sistema operativo	Log Files
Windows (non-system user account)	%localappdata%\unity\editor\upm.log
Windows (system user account)	%allusersprofile%\unity\editor\upm.log

Editor: Para ver el registro del editor, se debe abrir una ventana de la consola (menu: Window > General > Console) y seleccionar Abrir registro del editor en el menú de la ventana de la consola. También se puede navegar a la siguiente carpeta:

Tabla 5. Registro del editor

Sistema operativo	Log Files
Linux	~/.config/unity3d/Editor.log
macOS	~/Library/Logs/Unity/Editor.log También puede utilizar la utilidad Console.app desde su la carpeta /Applications/Utilities para buscar el archivo log.
Windows	%LOCALAPPDATA%\Unity\Editor\Editor.log

Jugador: para ver el registro del jugador, se debe abrir una ventana de la consola (menú: Window > General > Console) y seleccionar Abrir registro del jugador en el menú de la ventana de la consola. También se puede navegar a la siguiente carpeta:

Tabla 6. Registro del jugador

Sistema operativo	Log Files
Linux	~/.config/unity3d/CompanyName/ProductName/Player.log
macOS	~/Library/Logs/Company Name/Product Name/Player.log También puede utilizar la utilidad Console.app desde su la carpeta /Applications/Utilities para buscar el archivo log.
Windows	%USERPROFILE%\AppData\LocalLow\CompanyName\ProductName\Player.log

Síntesis

A continuación, la síntesis del componente formativo:

Glosario

Audio clip: contenedor para datos de audio en Unity. Unity admite activos de audio mono, estéreo y multicanal (hasta ocho canales). Unity puede importar formatos de archivo de audio .aif, .wav, .mp3 y .ogg, y formatos de módulo de seguimiento .xm, .mod, .it y .s3m

Asset: cualquier medio o dato que pueda usarse en su juego o proyecto. Un activo puede provenir de un archivo creado fuera de Unity, como un modelo 3D, un archivo de audio o una imagen. También puede crear algunos tipos de activos en Unity, como un Animator Controller, un Audio Mixer o una Render Texture.

Asset package: colección de archivos y datos de proyectos de Unity, o elementos de proyectos, que se comprimen y almacenan en un archivo, similar a los archivos Zip, con la extensión .unitypackage. Los paquetes de activos son una forma práctica de compartir y reutilizar proyectos y colecciones de activos de Unity.

Asset server: sistema de control de activos y versiones con una interfaz gráfica de usuario integrada en Unity. Permite a los miembros del equipo trabajar juntos en un proyecto en diferentes computadoras.

Asset store: biblioteca en crecimiento de activos comerciales y gratuitos creada por Unity y miembros de la comunidad. Ofrece una amplia variedad de activos, desde texturas, modelos y animaciones hasta ejemplos de proyectos completos, tutoriales y extensiones de editor.

Asset Store package: una categoría Profiler identifica los datos de la carga de trabajo para un subsistema de Unity (por ejemplo, las categorías Rendering, Scripting y Animation). Unity aplica códigos de colores a las categorías para ayudar a distinguir visualmente los tipos de datos en la ventana Profiler.

Category: método de almacenamiento de datos que reduce la cantidad de espacio de almacenamiento que se requiere. Consulte compresión de textura, compresión de animación, compresión de audio, compresión de compilación.

Compression: conjunto de medidas preventivas y reactivas de las organizaciones y sistemas tecnológicos que permiten resguardar y proteger la información buscando mantener la confidencialidad, la disponibilidad e integridad de datos.

Frustum: región del espacio 3D que una cámara en perspectiva puede ver y renderizar. En la vista de Escena, el tronco está representado por una pirámide rectangular truncada con su parte superior en el plano de recorte cercano de la cámara. Un plano que limita qué tan cerca puede ver una cámara desde su posición actual. El plano es perpendicular a la dirección de avance (Z) de la cámara.

GameObject: objeto fundamental en las escenas de Unity, que puede representar personajes, accesorios, escenarios, cámaras, waypoints y más. La funcionalidad de un GameObject está definida por los Componentes adjuntos a él.

Level of Detail: técnica de nivel de detalle (LOD) es una optimización que reduce la cantidad de triángulos que Unity tiene que representar para un GameObject cuando aumenta su distancia de la cámara.

Model: representación de modelo 3D de un objeto, como un personaje, un edificio o un mueble.

Model file: archivo que contiene datos 3D, que pueden incluir definiciones de mallas, huesos, animación, materiales y texturas.

Package: contenedor que almacena varios tipos de características y activos para Unity, incluidas las herramientas y bibliotecas de Editor o Runtime, colecciones de activos y plantillas de proyectos. Los paquetes son unidades independientes que Unity Package Manager puede compartir entre los proyectos de Unity. La mayoría de las veces se denominan paquetes, pero ocasionalmente se denominan paquetes Unity Package Manager (UPM).

Prefab: tipo de activo que le permite almacenar un GameObject completo con componentes y propiedades. La casa prefabricada actúa como una plantilla a partir de la cual puede crear nuevas instancias de objetos en la escena.

Profiler: ventana que ayuda a optimizar el juego. Muestra cuánto tiempo se pasa en las distintas áreas del juego. Por ejemplo, puede informar el porcentaje de tiempo dedicado a renderizar, animar o en la lógica de su juego.

Project: en Unity se usa un proyecto para diseñar y desarrollar un juego. Un proyecto almacena todos los archivos relacionados con un juego, como el activo y los archivos de escena.

Project Settings: amplia colección de configuraciones que le permiten configurar cómo se comportan física, audio, redes, gráficos, entrada y muchas otras áreas de su proyecto.

Sprite Atlas: textura que se compone de varias texturas más pequeñas. También se conoce como atlas de texturas, sprite de imagen, hoja de sprite o textura empaquetada.

State Machine: conjunto de estados en un Animator Controller en el que puede estar un personaje o GameObject animado, junto con un conjunto de transiciones entre esos estados y una variable para recordar el estado actual. Los estados disponibles dependerán del tipo de juego, pero los estados típicos incluyen cosas como inactivo, caminar, correr y saltar.

State Machine Behaviour: secuencia de comandos que se adjunta a un estado dentro de una máquina de estado para controlar lo que sucede cuando la máquina de estado entra, sale o permanece dentro de un estado, como reproducir sonidos cuando se ingresan los estados.

Referencias bibliográficas

Bond, G. J. (2014). *Introduction to Game Design, Prototyping, and Development: From Concept to Playable Game with Unity and C# (Illustrated ed.).* Addison-Wesley Professional.

Mernard, M. (2011). *Game Development with Unity.* Cengage Learning PTR.

Okita, A. (2014). *Learning C# Programming with Unity 3D.* Taylor & Francis.

Créditos

Nombre	Cargo	Regional y Centro de Formación
Claudia Patricía Aristizábal Gutiérrez	Responsable del equipo	Dirección General
Liliana Victoria Morales Gualdrón	Responsable de línea de producción	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Rafael Neftalí Lizcano Reyes	Responsable del equipo de diseño instruccional	Centro Industrial del Diseño y la Manufactura - Regional Santander
Fabián Andrés Gómez Pico	Experto temático 3D	Centro de Servicios y Gestión Empresarial - Regional Antioquia
Johnier Felipe Perafán Ledezma	Experto temático 3D	Centro de Servicios y Gestión Empresarial - Regional Antioquia
Luz Aida Quintero Velásquez	Diseñadora y evaluadora instruccional	Centro de Gestión Industrial – Regional Distrito Capital
Gustavo Santis Mancipe	Diseñador instruccional	Centro de Diseño y Metrología - Regional Distrito Capital
Oscar Absalón Guevara	Evaluador instruccional	Centro de Gestión Industrial – Regional Distrito Capital
Julia Isabel Roberto	Diseñadora y evaluadora instruccional	Centro para la Industria de la Comunicación Gráfica – Regional Distrito Capital
Gloria Amparo López Escudero	Adecuadora instruccional - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Andrés Felipe Velandia Espitia	Metodólogo para la formación virtual - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Francisco José Lizcano Reyes	Responsable del equipo	Centro Industrial del Diseño y la Manufactura - Regional Santander

Yuly Andrea Rey Quiñonez	Diseñador de contenidos digitales - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Laura Gisselle Murcia Pardo	Animador y productor multimedia - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Edison Eduardo Mantilla Cuadros	Diseño web	Centro Industrial del Diseño y la Manufactura - Regional Santander
Jorge Enrique Haylock Calderín	Desarrollo front-end	Centro Industrial del Diseño y la Manufactura - Regional Santander
Jhon Jairo Urueta Alvarez	Desarrollador full-stack - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Carolina Coca Salazar	Evaluadora para contenidos inclusivos y accesibles - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Lina Marcela Perez Manchego	Validadora de recursos digitales - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Leyson Fabián Castaño Pérez	Validador de recursos digitales - 2023	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital