MASTERING COMPUTER SCIENCE

Mastering Unity

A Beginner's Guide

EDITED BY

Sufyan bin Uzayr

CRC Press
Taylor & Francis Group

Dominando la unidad

Dominar la informática

Editor de la serie: Sufyan bin Uzayr

Dominar Unity: una guía para principiantes
Divya Sachdeva y Aruqqa Khateib

Dominar Unreal Engine: una guía para principiantes
Divya Sachdeva y Aruqqa Khateib

Dominar maquetas y marcos de interfaz de usuario:
Una guía para principiantes
Mohamed Mustafa MC y Kapil Kishnani

Dominar Ruby on Rails: una guía para principiantes
Mathew Rooney y Madina Karybzhanova

Mastering Sketch: una guía para principiantes
Mathew Rooney y MD Javed Khan

Dominar Java: una guía para principiantes
Divya Sachdeva y Natalya Ustukpayeva

Para obtener más información sobre esta serie, visite: https://www.routledge.com/Mastering-Computer-Science/serie de libros/MCS

La serie de libros "Mastering Computer Science" está escrita por los miembros del equipo de la Academia Zeba, dirigidos por Sufyan bin Uzayr.

Zeba Academy es una empresa de EdTech que desarrolla cursos y contenido para estudiantes principalmente en campos STEM y ofrece consultoría educativa a universidades e instituciones de todo el mundo. Para obtener más información, visite https://zeba.academy

Dominando la unidad

Una guía para principiantes

Editado por Sufyan bin Uzayr

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

Primera edición publicada 2022
por Prensa CRC

6000 Broken Sound Parkway NW, Suite 300, Boca Ratón, FL 33487-2742

y por CRC Press
2 Park Square, Milton Park, Abingdon, Oxon, OX14 4RN

CRC Press es un sello de Taylor & Francis Group, LLC

ISBN: 9781032103198 (hbk)
ISBN: 9781032103174 (pbk)
ISBN: 9781003214755 (ebk)

DOI: 10.1201/9781003214755

Composición tipográfica en Minion
por KnowledgeWorks Global Ltd.

Contenido

Sobre el editor

Sufyan bin Uzayr es escritor, programador y emprendedor con más de una década de experiencia en la industria. Es autor de varios libros en el pasado, pertenecientes a una amplia gama de temas, que van desde Historia hasta Computadoras/TI.

Sufyan es el Director de Parakozm, una empresa multinacional de TI especializada en soluciones EdTech. También dirige Zeba Academy, una vertical de aprendizaje y enseñanza en línea con un enfoque en los campos STEM.

Sufyan se especializa en una amplia variedad de tecnologías, como JavaScript, Dart, WordPress, Drupal, Linux y Python. Tiene varios títulos, incluidos los de Administración, TI, Literatura y Ciencias Políticas.

Sufyan es un nómada digital que divide su tiempo entre cuatro países. Ha vivido y enseñado en universidades e instituciones educativas de todo el mundo. Sufyan muestra un gran interés por la tecnología, la política, la literatura, la historia y los deportes, y en su tiempo libre disfruta enseñando programación e inglés a jóvenes estudiantes.

Obtenga más información en sufyanism.com.

Introducción a la unidad

Unity es un sofisticado entorno de desarrollo integrado (IDE) multiplataforma para desarrolladores, así como un motor de juego tridimensional y bidimensional (3D/2D). Echemos un vistazo más de cerca a lo que esto implica.

Al igual que un motor de juego, Unity puede proporcionar muchos de los elementos integrados más cruciales que hacen que un juego funcione. Esto incluye física, representación 3D y detección de colisiones. Desde el punto de vista de un desarrollador, esto indica que no hay necesidad de reinventar la rueda. En lugar de comenzar un nuevo proyecto desarrollando un nuevo motor de física desde cero, calculando cada movimiento individual de cada sustancia o la forma en que la luz debe rebotar en varias superficies.

Lo que hace que Unity sea aún más potente es la presencia de una "Tienda de activos" en auge. Este es simplemente un lugar donde

DOI: 10.1201/9781003214755-1

los desarrolladores pueden publicar sus inventos y compartirlos con la comunidad.

¿Quieres un efecto de fuego impresionante y no tienes tiempo para crear uno desde cero? Deberíamos poder encontrar algo en la tienda de activos. ¿Quiere agregar controles de inclinación a nuestro juego sin el lento proceso de ajustar la sensibilidad? Lo más probable es que también haya un activo para eso.

Todo esto significa que el desarrollador del juego puede concentrarse en lo que es importante: crear una experiencia única y agradable mientras codifica solo los elementos específicos para eso.
visión.

¿QUÉ ES EXACTAMENTE EL IDE DE UNITY?

Unity es un IDE además de un motor de juego. El término “entorno de desarrollo integrado” se refiere a una interfaz que proporciona acceso a todas las herramientas de desarrollo en un solo lugar. El programa Unity tiene un editor visual que permite a los desarrolladores arrastrar y soltar elementos en escenas y ajustar sus características.

El software Unity también incluye una gran cantidad de otras funciones y herramientas esenciales, como navegar entre carpetas de proyectos y crear animaciones usando una herramienta de línea de tiempo.

Cuando se trata de codificación, Unity utilizará un editor alternativo de su elección. La opción más frecuente es Visual Studio de Microsoft, que integra la mayoría de los
tiempo.

¿Cuál es el lenguaje utilizado por Unity?

Unreal utiliza C# para administrar el código y la lógica, con una gran cantidad de clases e interfaz de programación de aplicaciones (API)

que tendrás que entender. La buena noticia es que puedes hacer mucho en Unity sin tener que lidiar con mucho código.
Sin embargo, saber programar abre muchas posibilidades nuevas de lo que puedes hacer, y Unity te permite personalizar prácticamente cualquier cosa.

Afortunadamente, C# también es uno de los lenguajes de programación más accesibles para principiantes. También vale la pena conocerlo porque se usa ampliamente en el negocio y tiene mucho en común con otros lenguajes prominentes como C y Java. En otras palabras, estudiar Unity con C# es una excelente manera de comenzar con la codificación.

¿Qué es Unity 3D y cómo se usa?

En pocas palabras, Unity es el motor de juegos más popular del mundo. Tiene muchas funciones y es lo suficientemente versátil como para crear prácticamente cualquier juego que se te ocurra.

Unity es popular tanto entre los desarrolladores aficionados como entre los estudios AAA debido a su inigualable funcionalidad multiplataforma. Se ha utilizado para crear juegos como Pokémon GO, Hearthstone, RimWorld, Cuphead y muchos más.

Si bien el nombre implica 3D, Unity 3D también incluye capacidades para la producción de juegos en 2D.

Debido a la API de secuencias de comandos de C# y la integración integrada de Visual Studio, a los programadores les encanta. Para aquellos que buscan una alternativa a Visual Studio, Unity proporciona JavaScript como lenguaje de secuencias de comandos y MonoDevelop como lenguaje. VA.

Por otro lado, los diseñadores lo adoran ya que tiene herramientas de animación robustas que simplifican la creación de nuestras secuencias 3D o el desarrollo de animaciones 2D desde cero. En Unity, se puede animar casi cualquier cosa.

Además, Unity 3D tiene una versión gratuita que permite a los creadores lanzar juegos producidos con Unity Personal sin pagar por el programa, siempre y cuando generen menos de $100 000 con los juegos.

Para aquellos que estén dispuestos a pagar, Unity ofrece capacidades adicionales específicas y un plan de licencia personalizable a través de un enfoque de suscripción por niveles. Los clientes Premium también obtendrán acceso al código fuente y desarrollo de Unity.
asistencia.

Debido a que Unity existe desde 2005, ha acumulado una base de usuarios significativa y una increíble biblioteca de materiales. Unity no solo ofrece una excelente documentación, sino que los videos y tutoriales están disponibles en línea en abundancia.

Solo por esta razón, los principiantes deberían comenzar a usar Unity. Unity actúa como un centro de conocimiento y recursos entre muchos motores de videojuegos basado simplemente en su gran comunidad.

Otros motores de juego frente a Unity

Otros motores de juegos grandes están disponibles para el desarrollo. Unreal Engine y Cryengine se encuentran entre los motores de juegos que compiten con Unity. Entonces, ¿qué hace que Unity sea tan atractivo?

Debido a que está en un sitio web de Android, probablemente esté interesado en el desarrollo móvil. Aquí es donde Unity brilla como herramienta de programación. Si bien alguna vez se conoció como "Unity 3D", el programa ha evolucionado para ser tan poderoso como una herramienta de creación en 2D. No solo eso, sino que la forma en que se manejan las imágenes hizo que la transferencia de experiencias a hardware de gama baja fuera bastante simple.

Unity impulsa la gran mayoría de los productos en Google Play Store por estos motivos.

Sin embargo, debido a que Unity es multiplataforma, crear juegos en iOS, PC o incluso consolas de juegos es igual de simple. Unity también ofrece una excelente compatibilidad con la realidad virtual (VR) para los desarrolladores que desean crear aplicaciones para Oculus Rift o HTC Vive.

Desarrollo de juegos 3D de Unity

Los gráficos, las redes, el aumento tecnológico y el rendimiento en tiempo real han evolucionado enormemente en el desarrollo de juegos. Múltiples participantes del mercado actualmente buscan proporcionar creación y desarrollo de juegos a más personas y en más plataformas.

Un motor de juego se destaca entre la multitud: Unity 3D.
Es un motor de juego sofisticado que funciona en varias plataformas y es muy fácil de usar tanto para profesionales como para principiantes. Unity 3D es el motor de juego que debe elegir si desea un motor de juego sofisticado que pueda generar imágenes del mundo real sin consumir mucha potencia informática.

¡Considera estas estadísticas! Unity es responsable del 34 % de los juegos móviles gratuitos disponibles en Google Play Store y Apple App Store. ¿No es fascinante?
Las plataformas de juegos de Unity 3D han ayudado a desarrollar juegos que han llegado a más de 500 millones de jugadores en todo el mundo, una cifra que crece año tras año.
Sin duda, es uno de los motores de juego más populares del mercado.

Soluciones de juegos 3D de Unity: las soluciones de juegos 3D de Unity se encuentran entre las mejores de la industria. La multiplataforma

El motor ayuda a los creadores de juegos a crear juegos que se pueden utilizar en una variedad de contextos, que incluyen:

- **Juegos de consola y PC:** Esto proporciona gráficos enriquecidos y herramientas y juegos de herramientas fáciles de usar para desarrolladores, lo que no solo aumenta su capacidad para realizar modificaciones, sino que también garantiza que sus juegos funcionen de manera óptima.
- **Juegos instantáneos:** Project Tiny, un producto insignia de Unity Technologies, crea juegos livianos, rápidos y compactos con nuevo tiempo de ejecución y estabilidad en todos los segmentos.
- **Juegos móviles:** una de las plataformas más populares para usar las soluciones de juegos 3D de Unity. Esta plataforma es un maestro en juegos móviles, con contenido listo para dispositivos, optimización y potencial de ingresos.
- **Juegos AR/VR: Los juegos** de realidad aumentada/virtual (AR/VR) han crecido en popularidad durante la última década.
  Unity 3D hace realidad tales ambiciones al proporcionar herramientas inmediatas en tiempo real que permiten una amplia gama de posibilidades creativas en los motores AR/VR.

En términos de imágenes, los creadores de juegos con frecuencia quieren que su juego sea lo más similar posible a la realidad. Las imágenes deben dar al jugador la sensación de “estar en el juego”.
La tienda de activos para Unity3D proporciona modelos 3D sobresalientes para animaciones, sombreadores y texturas, componentes que hacen que la experiencia sea más realista. Si adquirimos efectos de sonido y activos de juego, Unity 3D nos lo permite ya que sobresale en todo si quieres adquirir efectos de sonido.

Considere esto: algunos individuos son brillantes desarrolladores, otros son excelentes en gráficos o animación, y algunos

son fantásticos con diversidad musical. Unity 3D es una excelente herramienta de activos que ayuda a unir varias características en una sola plataforma y crear algo original e indígena.

Todas las soluciones de juego bajo un mismo techo

Si buscamos la creación de juegos de extremo a extremo con excelentes módulos de desarrollo y soporte y mantenimiento adicional, Juego Studios es el lugar indicado.

En Juego Studios ofrecemos una de las mejores soluciones, si no la más completa, para ayudarlo a diseñar, crear y promocionar su producto. Somos profesionales en el desarrollo y diseño de conceptos y características de juegos rentables, confiables y expansivos.

Somos líderes en el mercado de diseño y producción de juegos y brindamos soluciones integrales. Nuestro equipo de más de 150 diseñadores, desarrolladores, ilustradores, animadores, diseñadores gráficos y otros nos ayudarán a llevar nuestro juego del concepto a la realidad.

Creamos y entregamos juegos para varias plataformas, incluidas PC, web, dispositivos móviles y consolas. Poco a poco estamos desarrollando o dejando nuestra huella en la creación de juegos AR y VR. Nuestros diversos juegos han obtenido múltiples honores empleando la tecnología y la conectividad de red más novedosas. Desde la plataforma 2D hasta el motor 3D y el entorno AR/VR, creamos juegos.

Juego Studios se complace en crear juegos emocionantes que amplían los límites de la industria utilizando las tecnologías y los marcos más potentes disponibles. Trabajamos con una variedad de motores de juegos, pero el que se muestra aquí es Unity 3D.

NUEVE BENEFICIOS SIGNIFICATIVOS DE DESARROLLO DE JUEGOS UNITY 3D

- **Múltiples plataformas:** una de las razones más importantes por las que los creadores de juegos de todo el mundo valoran Unity 3D es la capacidad de construir, administrar y entregar juegos multiplataforma. Esto implica que los creadores de juegos no están restringidos a una sola plataforma y, por lo tanto, pueden reproducirse en más de 25 grandes plataformas, incluidas dispositivos móviles, PC, consolas, televisión y, más recientemente, AR y VR; este tipo de facilidad y flexibilidad es lo que hace que la creación de juegos sea tan emocionante y refrescante. vocación.

- **Eficaz y confiable:** según una investigación de 2018, Unity Technologies y su motor de juegos insignia Unity 3D tenían más del 60 % de la participación de mercado en el campo de AR y VR, con más del 40 % de las plataformas de juegos móviles que utilizan la plataforma para producir juegos. Es eficiente, confiable y ampliamente utilizado por jugadores de todo el mundo.

- **Editor y desarrollador:** el modo de reproducción, las herramientas de historia de la línea de tiempo, la iluminación global en tiempo real y la generación de perfiles de memoria completa con animadores redireccionables son solo algunas de las funciones que hacen de Unity 3D un editor poderoso y avanzado pero fácil de usar.

- **Representación múltiple:** Unity 3D, que ha recibido varios honores por su sistema de creación de juegos, es quizás uno de los tres mejores sistemas del mundo para la representación e implementación de juegos. Si bien es extremadamente

rápido con modelos de renderizado 2D, es igualmente excelente con renderizado 3D.

- **Modo de reproducción:** una de las herramientas más excelentes para la edición iterativa rápida es el modo de reproducción. La función de modo de juego en Unity 3D es una de las características más populares del motor de juego. Permite al desarrollador ver y jugar rápidamente dentro del juego y probarlo y evaluarlo. Facilita la conveniencia de probar cómo pueden funcionar las cosas sin mucha dificultad. Si se encuentra con un problema técnico o cree que el juego no se está ejecutando correctamente, se puede pausar y ajustar a satisfacción del desarrollador del juego, actualizando rápidamente los resultados. La referencia de fotograma a fotograma también es posible en el modo Play o Play Plus.

- **Sistemas multijugador:** el uso de la plataforma Unity 3D es una de las formas más sencillas de desarrollar un sistema de juego en red y en tiempo real. La excelente experiencia multijugador de este motor de juegos no solo es flexible, sino que también se implementa y expande rápidamente. Unity 3D le permite crear sistemas multijugador integrados que aprovechan los emparejadores y los servidores de retransmisión como plataforma.

- **Gran experiencia visual:** Unity 3D es una plataforma visual fantástica y una excelente plataforma para desarrollar juegos de experiencia visual. La aplicación es fantástica y es menos complicada y fácil en comparación con muchas otras tecnologías.

- **Análisis:** Unity 3D tiene análisis a los que cualquier desarrollador de juegos o cliente puede acceder a través del editor.

Con Unity Analytics, podemos obtener, descubrir y usar información relacionada con el juego. Puede proporcionarle información valiosa para establecer una plataforma más robusta y realizar pequeños ajustes para ofrecer una experiencia fantástica a los jugadores.

- **Comunidad de desarrolladores:** La comunidad de desarrolladores de Unity es un foro para que todos los desarrolladores vengan y discutan sus problemas y recomendaciones para mejorar el sistema y familiarizarse de inmediato con el motor.

En general, Unity 3D Game Development es un producto multiplataforma, todo en uno. Los creadores de juegos pueden importar juegos creados en otras plataformas, como iOS, PC, Play Store y consolas de juegos. Los desarrolladores pueden optar por realizar cambios menores en sus juegos para aprovechar al máximo las funciones de Unity 3D.

Opciones de licencia

Unity tiene tres niveles de precios: personal, plus y profesional.

Si bien la versión gratuita lo ayudará a comenzar, querremos actualizar a plus o pro si realmente queremos producir juegos comerciales.

Quienes no paguen la suscripción mensual deberán acreditar ganancias inferiores a $100.000 por los juegos que creen con Unity.

Los aficionados pueden suscribirse al plan Plus para obtener acceso a funciones adicionales y materiales de capacitación, que les ayudarán a monetizar sus juegos y mejorar su

habilidades. Esto es maravilloso para los desarrolladores independientes que recién comienzan con el desarrollo de Unity.

El nivel Pro está diseñado para estudios de juegos y equipos profesionales que requieren asistencia interna y aquellos que ganan más de $200,000 de los proyectos de Unity.

Vistas generales

Hemos estado usando Unity 3D durante muchos años para poder compartir comentarios y errores de primera mano.

En primer lugar, creemos que Unity es un motor excelente. No es la mejor, pero es una herramienta fantástica y completa que es ideal para principiantes.

En segundo lugar, no existe el motor de juego más refinado. Sólo hay herramientas disponibles para el trabajo. Algunos son superiores a otros, y todo depende de los requisitos del proyecto único. Si bien necesitaremos habilidades de codificación para utilizar Unity, las herramientas pueden ayudarnos a aprender cualquier cosa que elijamos.

Una breve búsqueda en Google produce ejemplos de código y materiales para todo, desde juegos de disparos en primera persona hasta juegos de combinación estilo Candy Crush y todo lo demás.
Sin embargo, existen razones convincentes para dominar la codificación además de los trabajos de 3D/gráficos por computadora (CG). Unity es, en esencia, un motor de juego completo que simplifica la producción de juegos.
Si bien puede haber mejores motores para usar según los requisitos de nuestro proyecto, comprender Unity solo puede ayudarnos a mejorar como desarrolladores de juegos.

Unity es el camino a seguir si somos completamente novatos porque la comunidad ayudará a aprender y las herramientas durarán mucho tiempo. Si tuviéramos más experiencia, preferiríamos

12 ÿ Dominar la unidad

Unity debido a sus posibilidades de implementación multiplataforma y su proceso de desarrollo más rápido. Tenga en cuenta que muchos desarrolladores de juegos AAA usan Unity, lo que lo convierte en una opción sólida en el vasto campo de los videojuegos.

Requisitos del sistema para el editor de Unity

- **Requisitos del sistema**
  - **Versión del sistema operativo:** Windows 7 (SP1+), Windows 10 y Windows 11, solo versiones de 64 bits.
  - **CPU:** arquitectura X64 con conjunto de instrucciones SSE2 apoyo.
  - **API de gráficos:** compatible con DX10, DX11 y DX12 GPU.
  - **Requisitos adicionales:** Controladores admitidos oficialmente por el proveedor de hardware.

Requisitos del sistema para Unity Player

- **Móvil**
  - **Sistema operativo**
  - **Versión:** 4.4 (API 19)+
  - **CPU:** ARMv7 con compatibilidad con Neon (32 bits) o ARM64
  - **API de gráficos:** OpenGL ES 2.0+, OpenGL ES 3.0+ y Vulkan
  - **Requisitos adicionales:** 1GB+ RAM

- **Escritorio**
  - **Sistema operativo**
  - **Versión:** Windows 7 (SP1+), Windows 10 y Windows 11.
  - **CPU:** arquitectura x86, x64 con instrucción SSE2 establecer soporte.
  - **API de gráficos:** compatible con DX10, DX11, DX12.
  - **Requisitos adicionales:** los controladores fueron admitidos oficialmente por el proveedor de hardware.

El backend de secuencias de comandos de Intermediate Language To C++ (IL2CPP) requiere Visual Studio 2015 con C++ Componente de herramientas o posterior y el SDK de Windows 10 para desarrollo.

ARQUITECTURA DE LA UNIDAD

El motor de Unity está escrito en C/C++ nativo, pero interactuamos con él a través de una capa de C#. Como resultado, debemos estar familiarizados con algunas de las nociones fundamentales de las secuencias de comandos de C#. Esta parte del Manual del usuario describe cómo Unity implementa .NET y C#, así como cualquier error que pueda ver al codificar.

Descripción general de .NET en Unity

Unity aprovecha la plataforma .NET de código abierto para garantizar que las aplicaciones creadas con Unity puedan funcionar en una amplia variedad de configuraciones de hardware. .NET brinda soporte para varios lenguajes y bibliotecas de API.

Secuencias de comandos de back-end

Unity presenta dos backends de secuencias de comandos: Mono e IL2CPP (Intermediate Language To C++), cada uno con su técnica de compilación:

- Mono emplea compilación justo a tiempo (JIT) y genera código según sea necesario durante el tiempo de ejecución.
- IL2CPP emplea compilación anticipada (AOT), que construye nuestro programa completo antes de la ejecución.

La ventaja de emplear un backend de secuencias de comandos basado en JIT es que compila mucho más rápido que AOT y es independiente de la plataforma.

El editor de Unity está basado en JIT, con Mono como backend de secuencias de comandos. Podemos seleccionar qué backend de programación utilizar al crear un reproductor para su aplicación. Para lograr esto en el Editor, vaya a Edición > Configuración del proyecto > Reproductor, abra el panel Otras configuraciones, haga clic en la opción Scripting Backend y elija el backend deseado.

Eliminación de código dirigida

Cuando desarrolla su programa, Unity busca en los ensamblados integrados (.DLL) código innecesario y lo elimina.
Este método minimiza el tamaño binario final de nuestra compilación y aumenta el tiempo de compilación.

Cuando se usa Mono, la eliminación de código está deshabilitada de forma predeterminada; sin embargo, la extracción de código no se puede desactivar para IL2CPP. Se puede ajustar la agresividad de Unity mientras se elimina el código. Abra el panel Otras configuraciones, haga clic en el menú desplegable Nivel de eliminación administrado y elija el nivel de código que niega ese deseo.

Recolección de Basura

Tanto para el backend Mono como para IL2CPP, Unity emplea el recolector de elementos no utilizados de Boehm. De forma predeterminada, Unity emplea el modo incremental. Aunque Unity sugiere usar el modo Incremental, puede desactivarlo para emplear la recolección de basura "detener el mundo".

Para cambiar entre el modo incremental y "detener el mundo", navegue por Editar > Configuración del proyecto > Reproductor, abra el panel Otras configuraciones y marque la casilla de verificación Usar GC incremental. En el modo incremental, el recolector de elementos no utilizados de Unity se ejecuta durante un tiempo limitado y es posible que no siempre recopile todos los objetos en una sola pasada. Esto distribuye el tiempo que se tarda en adquirir cosas en muchos fotogramas, lo que reduce la intermitencia y los picos de CPU.

Utilice Unity Profiler para examinar la cantidad de asignaciones y posibles picos de CPU en su aplicación. También podemos usar la API GarbageCollector para desactivar la recolección de basura por completo en los reproductores. Cuando el colector está apagado, debemos tener cuidado de no asignar demasiada memoria.

Bibliotecas del sistema para .NET

Unity es compatible con una amplia gama de sistemas y puede emplear una variedad de backend de secuencias de comandos según la plataforma. En varias circunstancias, las bibliotecas del sistema .NET requieren implementaciones específicas de la plataforma para funcionar correctamente. Si bien Unity hace todo lo posible para admitir la mayor cantidad posible del ecosistema .NET, existen varias excepciones a las secciones de las bibliotecas del sistema .NET que Unity expresamente no admite.

Unity no ofrece garantías sobre la velocidad o la asignación de las bibliotecas del sistema .NET en todas las versiones de Unity. Unidad,

como regla general, no corrige ninguna regresión de rendimiento en las bibliotecas del sistema .NET.

Unity no es compatible con el sistema. no esta garantizado que la biblioteca de dibujos funcionará en todos los sistemas.

Un back-end de secuencias de comandos JIT nos permite generar código dinámico C#/.NET Intermediate Language (IL) durante el tiempo de ejecución de nuestra aplicación, mientras que un back-end de secuencias de comandos AOT no lo hace. Es fundamental tener esto en cuenta al usar bibliotecas de terceros, ya que pueden utilizar rutas de código distintas para JIT y AOT o emplear rutas de código que se basan en código producido dinámicamente.

Aunque Unity admite varios perfiles de API de .NET, para todas las aplicaciones nuevas, debemos utilizar el nivel de compatibilidad de API de .NET Standard 2.0 por los siguientes motivos:

- Debido a que .NET Standard 2.0 tiene una superficie API más baja, también tiene una implementación más pequeña. El tamaño de nuestro archivo ejecutable final se reduce como resultado de esto.

- Debido a que .NET Standard 2.0 ha mejorado la compatibilidad entre plataformas, es más probable que nuestro código se ejecute en todas las plataformas.

- Debido a que .NET Standard 2.0 es compatible con todos los tiempos de ejecución de .NET, nuestro código funcionará en una gama más amplia de escenarios de VM/tiempo de ejecución (p. ej., .NET Framework, .NET Core, Xamarin, Unity).

- Más errores se trasladan al tiempo de compilación en .NET Standard. Varias API en .NET 4.7.1 están disponibles en el momento de la compilación; sin embargo, las implementaciones de algunas plataformas arrojan un error durante el tiempo de ejecución.

Otros perfiles pueden ser útiles si necesitamos admitir una aplicación existente más antigua, por ejemplo. Cambie el perfil de .NET en la configuración del reproductor si desea un nivel de compatibilidad de API diferente. Para hacerlo, vaya a Editar > Configuración del proyecto > Reproductor > Otras configuraciones y elija el nivel deseado de la selección Nivel de compatibilidad API.

Uso de bibliotecas .NET de terceros

Las bibliotecas .NET de terceros solo se deben usar si se prueban minuciosamente en una amplia gama de configuraciones y sistemas de Unity.

Debemos perfilar el uso de las bibliotecas de su sistema .NET en todas las plataformas de destino, ya que sus características de rendimiento pueden diferir según el backend de secuencias de comandos, las versiones de .NET y los perfiles que empleamos.

Considere los siguientes aspectos al revisar una biblioteca de terceros:

- **Compatibilidad:** es posible que algunas plataformas de Unity y el backend de secuencias de comandos no sean compatibles con bibliotecas de terceros.
- **Rendimiento:** las bibliotecas de terceros pueden funcionar de manera significativamente diferente en Unity que en otros .NET tiempos de ejecución
- **Tamaño binario de AOT:** debido a la cantidad de dependencias utilizadas por la biblioteca, las bibliotecas de terceros pueden aumentar drásticamente el tamaño binario de AOT.

Reflexión aérea en C#

Mono e IL2CPP almacenan en caché internamente todos los objetos de reflexión de C# (System.Reflection), y Unity no los desecha.

recogerlos por diseño. Como resultado de este comportamiento, el recolector de elementos no utilizados busca constantemente los objetos de reflexión de C# almacenados en caché durante la vida útil de nuestro programa, lo que provoca costos innecesarios y potencialmente considerables en el recolector de elementos no utilizados.

Evite enfoques como la reducción de los gastos generales del recolector de basura. Assembly.GetTypes y Type.GetMethods() en nuestra aplicación generan una gran cantidad de objetos de reflexión de C# en tiempo de ejecución. En su lugar, escanee los ensamblajes en el Editor para obtener los datos necesarios y serialice y codifique para el uso del tiempo de ejecución.

Comportamiento único de UnityEngine.Object

UnityEngine: en Unity, un objeto es un objeto C# específico, ya que está conectado a un objeto equivalente nativo de C++.
Por ejemplo, cuando utilizamos un componente Camera, Unity no mantiene el estado del objeto en el objeto C# sino en su equivalente nativo de C++.

Actualmente, Unity no admite el uso de la clase C# WeakReference con UnityEngine.Objects.
Como resultado, nunca debemos usar una referencia débil para referirnos a un elemento cargado.

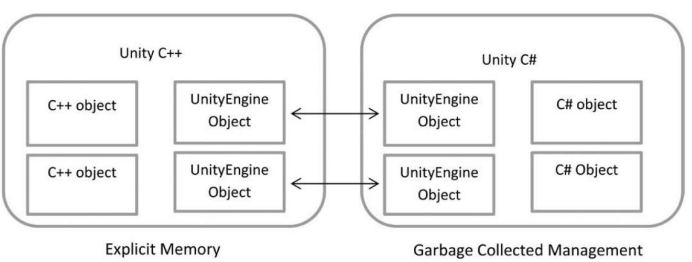


FIGURA 1.1 Objetos en Unity.

Los objetos UnityEngine son compartidos por Unity C# y Unity C++

Cuando usamos un método como Object.Destroy u Object.DestroyImmediate para destruir un UnityEngine. Unity elimina (descarga) el objeto de contador nativo cuando se deriva de un objeto. Debido a que el recolector de elementos no utilizados mantiene la memoria, no podemos eliminar explícitamente el objeto de C#. El recolector de basura recoge y destruye el objeto manipulado una vez que no hay más referencias a él.

Si se vuelve a acceder a un UnityEngine.Object destruido, Unity recrea el objeto homólogo nativo para la mayoría de los tipos. Dos excepciones a este comportamiento recreativo son MonoBehaviour y ScriptableObject: Unity nunca las vuelve a cargar una vez destruidas.

MonoBehaviour y ScriptableObject anulan los operadores de igualdad (==) y desigualdad (!=). Cuando un MonoBehaviour o ScriptableObject destruido se compara con un valor nulo, los operadores devuelven verdadero si el objeto administrado todavía está presente y aún no se ha recolectado basura.

Porque el ?? y ?. los operadores no se pueden sobrecargar, son incompatibles con los objetos derivados de UnityEngine. Objeto. Cuando se usa en un MonoBehaviour o ScriptableObject destruido mientras el objeto administrado todavía está activo, los operadores no producen los mismos resultados que los operadores de igualdad y desigualdad.

Evite el uso de Async y Await

Debido a que la API de Unity no es segura para subprocesos, debemos evitar usar tareas asíncronas y en espera. Cuando se ejecutan, los trabajos asincrónicos asignan objetos con frecuencia, lo que puede causar dificultades de rendimiento si se usa en exceso. Además, Unity no

no detener instantáneamente los procesos asíncronos que se ejecutan en subprocesos administrados cuando salimos del modo de reproducción.

Tanto en el modo de edición como en el de reproducción, Unity reemplaza el SynchronizationContext predeterminado con un UnitySynchronizationContext personalizado y ejecuta todas las tareas en el subproceso principal. Para usar tareas asíncronas, debemos generar y administrar manualmente nuestros hilos usando TaskFactory, y debemos usar el SynchronizationContext predeterminado en lugar de la versión de Unity.

Para detener manualmente los trabajos, use EditorApplication. playModeStateChanged para escuchar los eventos de entrada y salida del modo de reproducción. Sin embargo, debido a que no utilizamos UnitySynchronizationContext, la mayoría de las API de secuencias de comandos de Unity no están disponibles.

Recargar código en el editor de Unity

Las recargas de información del dominio y cómo afectan el rendimiento de la aplicación. También incluye información sobre la ejecución del código cuando se inicia el Editor y cómo ingresar y salir rápidamente del modo de reproducción usando el modo de ingreso configurable.

Serialización de Scripts

La serialización es el acto de convertir automáticamente estructuras de datos o estados de objetos en un formato que Unity puede almacenar y reconstruir más tarde. Esta sección incluye información sobre cómo utilizar la serialización de manera efectiva en nuestro Proyecto.

Compilación de guiones

Cómo y en qué secuencia Unity construye sus programas.
Esta sección también incluye información sobre Definiciones de ensamblado y mejores prácticas para utilizarlas.

¿CUÁL DE LOS SIETE LENGUAJES DE DESARROLLO DE JUEGOS DE UNITY DEBEMOS APRENDER?

Nunca ha sido tan fácil crear juegos. Las plataformas de desarrollo de juegos de Unity permiten la creación de todo, desde sencillos juegos de plataformas en 2D hasta juegos de disparos en primera persona en 3D altamente sofisticados. Unity está disponible de forma gratuita para los pequeños desarrolladores, y hay varias instrucciones sobre cómo utilizar el editor para crear prototipos de sus ideas.

Aprender a utilizar el software de Unity solo puede llevarlo hasta cierto punto. El código que dicta el comportamiento de tu juego será el verdadero corazón del mismo. Elegir un idioma para aprender para la producción de juegos puede ser difícil, pero el caso de Unity es sencillo.

C# es la mejor opción

C# es el mejor lenguaje para aprender sobre Unity para cualquier persona nueva en la plataforma o que tenga experiencia previa con la programación orientada a objetos. De hecho, por una buena razón, C# es el único lenguaje que vale la pena conocer para la plataforma.

Mono, una adaptación multiplataforma del marco .NET de Microsoft, es utilizado por Unity. C# es el lenguaje de programación principal de .NET y todas las bibliotecas de Unity están escritas en C#. No sería exagerado afirmar que C# es el lenguaje de Unity. Unity ha dicho inequívocamente que C# sería el único lenguaje compatible con el motor en el futuro.

Esta es una excelente noticia porque C# es un lenguaje robusto que también es fácil de aprender. Unity es solo una de las muchas razones convincentes para aprender C# y, como novatos, es posible que incluso lo encontremos más accesible. La creación de juegos ofrece una estructura de aprendizaje y los objetivos basados en proyectos conducen a una mejor comprensión de los nuevos temas.

Unity está ampliando los límites de lo que C# puede lograr al presentar el sistema de trabajo de C# y ECS, y el nuevo compilador Burst lo hace más rápido que nunca.

JavaScript es la alternativa actual

También se admite UnityScript, que se basa en JavaScript. Desde su primer lanzamiento, JavaScript ha coexistido con C# como un lenguaje de programación de Unity completamente completo. La documentación de secuencias de comandos de Unity incluía código de muestra en C# y JavaScript para la mayoría de las partes de la biblioteca.

Esto fue útil para los desarrolladores con experiencia en JavaScript, ya que podían utilizar una sintaxis similar, aunque se organizaron variaciones en el código. Sin embargo, había un problema.

Si bien UnityScript parece ser similar a JavaScript, no lo es. UnityScript tiene clases, mientras que JavaScript no. UnityScript carece de funciones de JavaScript, como la declaración de múltiples variables y los puntos y comas opcionales.

Quizás lo más importante es que ha sido difícil buscar experiencia en JavaScript en proyectos de Unity, ya que la mayoría de la gente se refería a él como JavaScript en lugar de UnityScript. El diseño del sitio y los resultados del desarrollo del juego fueron confusos, y la distinción entre los idiomas fue una fuente de disputa entre los desarrolladores de JavaScript puro.

Como era de esperar, Unity declaró que suspendería el soporte para UnityScript, y se ha establecido un plazo para su desmantelamiento.

La tercera opción tradicional: Boo

Boo, un lenguaje similar a Python, estaba disponible en los primeros días de Unity. Esto es algo inesperado dado que el diseñador de Boo, Rodrigo B. De Oliveira, trabajó anteriormente con

Unidad. El lenguaje es compatible con .NET y Mono, y estaría completamente integrado con el motor del juego.

¿Dónde salieron mal las cosas?

Pocas personas lo usaron, muy probablemente porque asumieron que solo era un intento de emular a Python. Con el tiempo, Unity eliminó la compatibilidad con Boo, y las próximas actualizaciones de UnityScript dejarán obsoletos todos los scripts de Boo anteriores en Unity. Algunos pueden considerar que esta es una oportunidad desperdiciada, ya que Boo fue un intento fantástico de sintaxis similar a Python para la programación .NET.

IronPython es una opción inusual

Python generalmente no es el lenguaje para nosotros si queremos crear juegos, aunque es factible. Charlie Calvert explica cómo ejecutar Python desde C# en su blog de la comunidad de desarrolladores de Microsoft, pero no es para los débiles de corazón.

IronPython todavía está en desarrollo activo más de diez años después.

Para resumir, debemos obtener las bibliotecas de IronPython de GitHub e incluirlas en su proyecto de C#. Esto nos permite llamar a los scripts de Python desde los scripts de C# como lo haríamos con cualquier otra biblioteca.

IronPython también permite que Python llame a bibliotecas .NET. Tan útil como parece, debido a que Unity se basa en C#, es ineficaz.

IronPython y su proyecto hermano, IronRuby, que conecta C# con el lenguaje de programación Ruby, son proyectos hermosos, pero no son viables para su uso con Unity.

Lua es una opción intrigante

MoonSharp, un intérprete de Lua, es una de las implementaciones más aceptables de un lenguaje externo para Unity.

Este proyecto no pretende reemplazar a C# como lenguaje de programación, sino servir como puente. El caso de uso ideal para MoonSharp sería proporcionar un medio para que los usuarios de nuestro juego desarrollen modificaciones del juego en el lenguaje de programación Lua.

También podemos usarlo para definir elementos y niveles de diseño de forma individual a partir de nuestro código de juego principal.

Vale la pena considerar MoonSharp si ya estamos trabajando en C# y buscando un método emocionante para conectarnos con nuestro código. Puede importarlo directamente a nuestros proyectos porque se puede acceder a él en la tienda de recursos de Unity.

C/C++ es el mejor lenguaje para complementos

A pesar de la rica biblioteca de Unity y todas las herramientas proporcionadas por C#, es posible que deseemos crear nuestros complementos de vez en cuando. Las principales razones por las que las personas eligen complementos son la velocidad y el acceso a un código base escrito en otro idioma. Construir estas secuencias de comandos en complementos de biblioteca de vínculos dinámicos (DLL) ahorra tiempo y, en determinadas circunstancias, mejora la velocidad.

C++ se usará para crear complementos en la mayoría de las situaciones, aunque C funcionaría igual de bien. El código se puede almacenar en la carpeta de complementos de Unity y se puede hacer referencia en el código siempre que se construya en una DLL. Sin embargo, si ya estamos familiarizados con la escritura en C/C++, aprender C# debería ser un esfuerzo razonablemente sencillo.

Rust es un nuevo lenguaje de programación para complementos

Rust es un lenguaje que tiene mucha publicidad al respecto. Los programadores experimentados lo adoran porque proporciona un enorme grado de control y evita los problemas de codificación en lenguajes menos seguros como C++. Mozilla diseñó el óxido

en 2009 como una herramienta para que los desarrolladores construyan rápidamente aplicaciones de alto rendimiento.

Si bien no es factible escribir Rust directamente en Unity, podemos usar funciones y métodos escritos en Rust desde nuestro código de Unity. En su pieza Medium, Jim Fleming explica cómo lograrlo en detalle. Si esto le resulta familiar, es porque es otro método para crear complementos nativos. Podemos usar la función DllImport de Unity para acceder a las funciones de Rust directamente desde el código C# usando la capacidad de Rust para interactuar con otros lenguajes. Naturalmente, hay diferentes etapas intermedias, y se recomienda leer el artículo de seguimiento de Jim y aprender sobre las FFI (interfaces de funciones extranjeras).

- **Una opción simple:** la postura de Unity hacia cualquier lenguaje que no sea C# es inconfundible, y los continuos avances hacia Unity se basan en este enfoque inquebrantable. Cuando combinamos esto con las mejoras continuas de Microsoft a C# como lenguaje, dominar C# para la producción de juegos de Unity es una obviedad. Además, para obtener un enfoque más directo sobre el aprendizaje de la creación de juegos, consulta Unity Learn.

Sin embargo, Unity es solo un motor entre muchos, y hay varias alternativas de software de producción de juegos para elegir.

Diez beneficios del lenguaje de programación C# para desarrolladores de Unity

En programación, C# es uno de los lenguajes de programación más aceptados, estructurados y populares. C# es ampliamente considerado como uno de los lenguajes de programación más destacados y potentes. Uno de los lenguajes compatibles es C#. Completa el trabajo rápidamente y funciona sin problemas. En este ensayo,

veremos los beneficios de C# sobre otros lenguajes de programación:

- **Lenguaje orientado a objetos:** debido a que C# es un lenguaje orientado a objetos, puede construir programas modulares que se pueden mantener y código reutilizable. Este es uno de los beneficios más significativos de C# frente a C++.
- **Recolección automática de basura:** C# ofrece una técnica muy eficaz para borrar y eliminar toda la basura del sistema. C# no causa un desorden en el sistema y no hace que el sistema se cuelgue durante
  ejecución.
- **No hay problema si la memoria se pierde:** C# ofrece una ventaja significativa en la copia de seguridad de la memoria. Las fugas de memoria y otros problemas similares no existirían en C#, como sucede en C++. En este escenario, C# supera a todos los demás lenguajes.
- **Fácil de desarrollar:** las extensas bibliotecas de clases simplifican el desarrollo de numerosas funcionalidades. C# ha influido en la mayoría de los programadores del mundo y tiene una larga historia en el mundo de la programación.
- **Multiplataforma:** Nuestro programa solo se ejecutará correctamente si el sistema tiene instalado NET Framework. Este es el requisito previo más crítico de C#. Esta también podría ser una excelente oportunidad para que los programadores novatos se familiaricen con el marco .NET.
- **Mejor integración:** las aplicaciones .NET tendrán una mejor integración e interoperabilidad con otras tecnologías NET. C# se basa en un lenguaje común

runtime (CLR), lo que simplifica la interfaz con componentes escritos en otros lenguajes (específicamente, lenguajes compatibles con CLR).

- **Codificación más comprensible:** la idea de los métodos get-set se ha formalizado, lo que hace que los códigos sean más legibles. Tampoco tenemos que preocuparnos por los archivos de encabezado en C#. Codificar en C# valdría la pena.
- **Escasez de opciones:** en la pila de Microsoft, hay una herramienta para todo. Entonces, simplemente adaptamos sus necesidades a la herramienta y la usamos. Por eso proponemos C# como un lenguaje beneficioso, especialmente para los novatos.
- **Soporte de programación:** en C# (.NET framework), puede comprar soporte de Microsoft, a diferencia de Java, donde la comunidad es nuestro soporte. Por lo tanto, si algo sale mal, podemos comunicarnos con Microsoft para obtener ayuda.
- **Compatibilidad con versiones anteriores:** las aplicaciones .NET solo son compatibles con los sistemas Windows y Microsoft dejará de brindar soporte para las plataformas Windows más antiguas. Si actualizamos a una nueva versión de Windows, siempre deberá actualizar nuestro marco .NET. Esto podría ser tanto un beneficio como un inconveniente. Un deseo de mejorar continuamente nos motiva a trabajar duro y prosperar en nuestra área. Esto, en mi opinión, es algo positivo.

Capítulo 2

Configuración de la unidad

En este capítulo, comenzaremos con la configuración de Unity en nuestras máquinas.

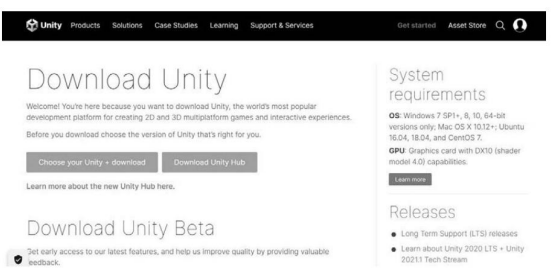
INSTALACION Y CONFIGURACION

La necesidad fundamental para crear contenido con Unity es descargar el motor y el entorno de desarrollo de Unity.

Es posible que descarguemos módulos adicionales para entregar a varias plataformas, así como herramientas para integrar secuencias de comandos de Unity en Visual Studio, además del motor principal.

Para obtener Unity, vaya a https://unity3d.com/get-unity/descargar.

Una vez allí selecciona: Elige tu Unity + Descargar.



DOI: 10.1201/9781003214755-2

En la siguiente pantalla, en Personal, haga clic en la opción Probar ahora. Esta es la alternativa gratuita de Unity, que incluye todas las funciones clave. Al comenzar este curso, es preferible comprender cómo utilizar el motor antes de actualizar a Plus o Pro.

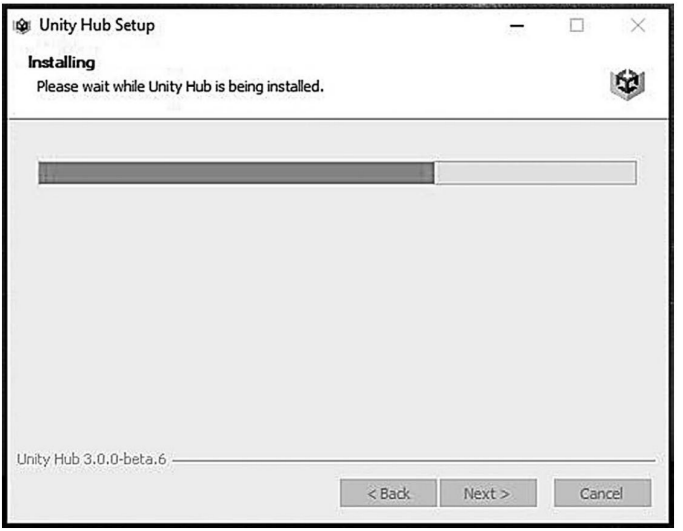
Requisitos del sistema para Unity Hub

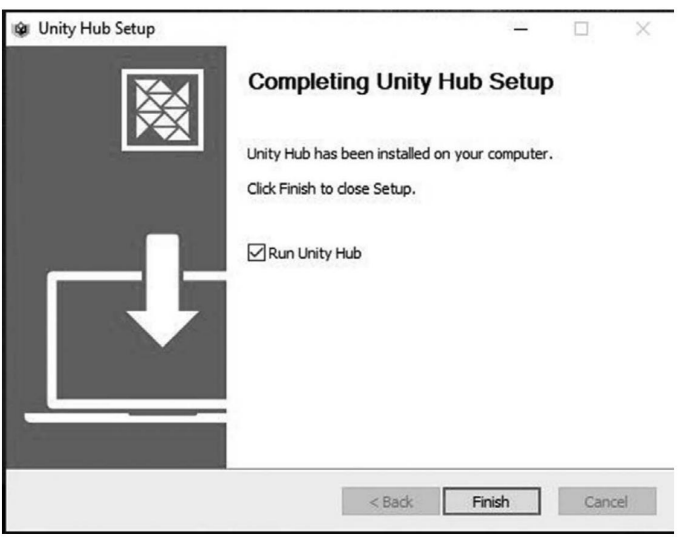
Sistemas operativos: Windows 7 SP1+, 8, 10, solo de 64 bits; Mac OS X 10.12+; Ubuntu 16.04, 18.04 y CentOS 7.

GPU: una tarjeta gráfica compatible con DX10 (shader model 4.0).

- Ejecutar la instalación que descargamos.
- Acepte la licencia y los términos, y luego presione Siguiente botón.
- Seleccione los componentes que queremos instalar con Unity y luego haga clic en "Siguiente". Tenga en cuenta que siempre podemos volver a ejecutar la instalación si deseamos modificar los componentes.
- Podemos modificar la ubicación donde se instalará Unity o dejar la ubicación predeterminada y hacer clic en "Siguiente".
- Es posible que veamos preguntas adicionales antes de la instalación, dependiendo de los componentes que elijamos. Siga las instrucciones en pantalla y luego haga clic en "Instalar".

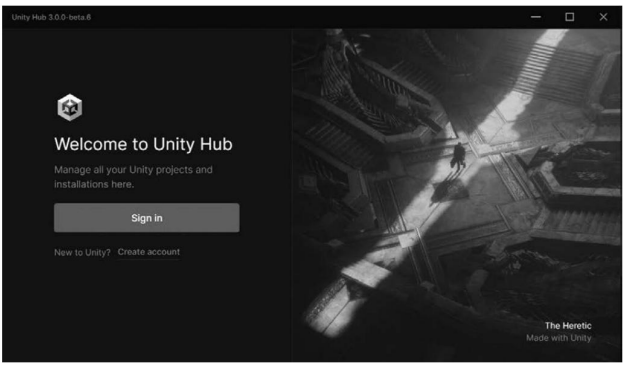
  Puede llevar algún tiempo instalar Unity. Unity se instalará en nuestra computadora después de que se complete la instalación.



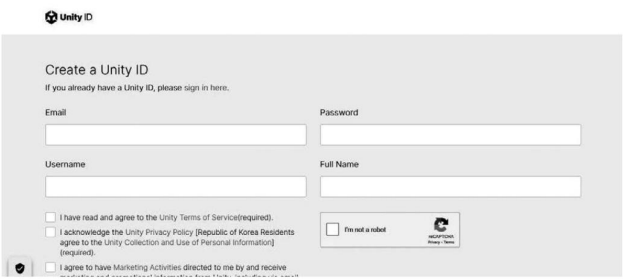


CREAR UNA CUENTA DE UNIDAD

- Para usar Unity, primero debe crear una cuenta. Comience iniciando Unity, lo que se puede hacer a través de los accesos directos del escritorio o del menú de inicio.



- Si ya tenemos una cuenta de Unity, inicie sesión aquí y omita el resto de esta instrucción. Si aún no tenemos una cuenta de Unity, haga clic en el botón "crear una".



- Para crear una cuenta de Unity, complete los espacios en blanco. Luego, selecciona "Crear un ID de Unity". Alternativamente, podemos unirnos usando nuestra cuenta de Google o Facebook.

- La cuenta de correo electrónico que usamos para registrarnos para obtener un ID de Unity recibirá un correo electrónico de confirmación. Para confirmar nuestra dirección de correo electrónico, haga clic en el botón "Enlace para confirmar correo electrónico".

- Después de verificar nuestro correo electrónico, regrese a la aplicación Unity y seleccione "Continuar".

- Haga clic en "Siguiente" después de seleccionar "Unity Personal".

DESARROLLANDO TU PRIMER PROYECTO

- Unity funciona bien para juegos bidimensionales (2D) y tridimensionales (3D). Desde la pantalla de inicio, todos los juegos de Unity comienzan como Proyectos.

- Inicie la unidad recién instalada; proyectos existentes aparecerá en el área nebulosa.

- Como se ilustra arriba, el icono Nuevo se encuentra en la esquina superior derecha de la ventana. Cuando hacemos clic en el botón, se nos enviará a la pantalla de configuración del proyecto.

- Podemos darle un nombre a nuestro proyecto, elegir dónde se guardará, el tipo de proyecto y agregar los existentes.
activos.

- Por el momento, llamemos a nuestro primer proyecto “Hola ¡Mundo!" y configúrelo en modo 2D.

- Haga clic en Crear proyecto para permitir que Unity cree los archivos principales de su proyecto. Esto puede tomar algún tiempo dependiendo del rendimiento de nuestra computadora, los activos agregados previamente y el tipo de Proyecto.

- Una vez creado nuestro nuevo proyecto y lanzado Unity.

- Echemos un vistazo rápido a lo que se ve en esta ventana. Por el momento, estamos enfocados en cuatro áreas principales:

  - Esta es la ventana donde crearemos nuestras Escenas. Las escenas son los escenarios en los que transcurre nuestro juego. Si hacemos clic en la pequeña pestaña Juego, obtendremos una ventana de vista previa que muestra cómo se verá el juego para el jugador. Por el momento, debería tener un fondo azul claro.

  - El Inspector vive en este lugar. Por el momento, está vacío porque no hay elementos en nuestro escenario. Veremos cómo se usa el Inspector en el futuro.

  - La jerarquía de escenas se muestra en esta ventana. Muestra una lista de todos los objetos en su escena actualmente activa, junto con su jerarquía padre-hijo. Estaremos agregando cosas a esta lista en breve.

  - Finalmente, la ventana Project Assets se encuentra en esta sección. Todos los recursos de nuestro proyecto actual se guardan y conservan en esta carpeta. Los activos importados externamente, incluidas texturas, fuentes y archivos de sonido, se guardan aquí antes de utilizarse
    en una escena

¿CÓMO FUNCIONA LA UNIDAD?

Todos los juegos en Unity tienen lugar en escenas. Las escenas son niveles en los que ocurren todos los componentes de nuestro juego, incluidos los niveles de juego, la pantalla de título, los menús y las escenas cinemáticas.

Una nueva escena en Unity contendrá un objeto de cámara denominado Cámara principal de forma predeterminada. Es posible agregar numerosas cámaras a la escena, pero por el momento nos ocuparemos de la cámara principal.

La cámara principal representa lo que observa o “captura” en un espacio conocido como ventana gráfica. Todo lo que entra en esta zona es visible para el jugador.

Al colocar el mouse dentro de la vista de escena y desplazarnos hacia abajo para alejar la vista de escena, podemos ver este puerto de vista como un rectángulo gris. (También podemos lograr esto manteniendo presionada la tecla Alt y arrastrando el botón derecho).

Una escena se compone de elementos conocidos como GameObjects. Los GameObjects pueden variar desde el modelo del jugador hasta la interfaz gráfica de usuario (GUI) de la pantalla, desde botones y adversarios hasta "administradores" invisibles, como el sonido.
fuentes.

Los GameObjects están asociados con un conjunto de componentes que explican cómo se comportan en la escena y cómo se relacionan con otros en la escena.

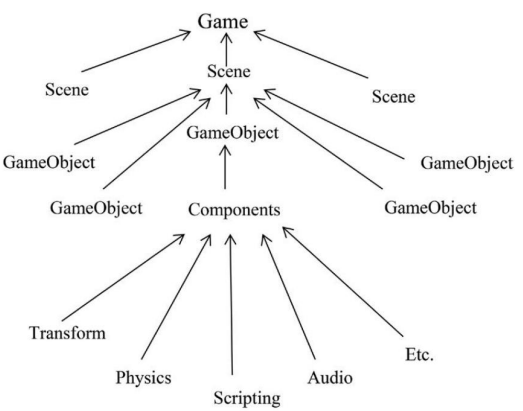
Podemos investigarlo ahora mismo. Mire al Inspector haciendo clic en la cámara principal en la jerarquía de escenas.
Ya no estará vacío; en cambio, incluirá varios "módulos".

El componente Transform es el componente más crítico de cualquier GameObject. Cualquier elemento de una escena tendrá una transformación que describa su posición, rotación y escala en el mundo del juego o, si corresponde, su padre.

Al hacer clic en Agregar componente y elegir el componente requerido, podemos adjuntar más elementos a un objeto.
En los siguientes tutoriales, también conectaremos Scripts a GameObjects para darles un comportamiento programado.

Considere los siguientes ejemplos de componentes:

- **Procesador:** La persona o programa a cargo de representar y hacer visibles los elementos.
- **Colisionador:** especifica los límites de colisión física para objetos.
- **Rigidbody:** Proporciona a un objeto atributos físicos en tiempo real como peso y gravedad.
- **Fuente de audio:** proporciona características del objeto para reproducir y almacenar sonido.
- **Oyente de sonidos:** este componente "escucha" el audio y lo envía a los altavoces del reproductor. Uno está presente por defecto en la cámara principal.
- **Animador:** esto permite que un elemento interactúe con el sistema de animacion
- **Luz:** hace que el elemento funcione como una fuente de luz con una variedad de efectos.



En este diagrama, podemos ver cómo Unity se compone a sí mismo en escenas usando GameObjects.

CREANDO SPRITES EN UNIDAD

- Los sprites son objetos 2D básicos con gráficos gráficos (llamados texturas) en ellos.
- Cuando el motor está en modo 2D, los sprites se usan de forma predeterminada. Debido a que los sprites no tienen ancho Z, parecen delgados como el papel cuando se ven en el espacio 3D. A menos que se conviertan en un espacio 3D, los sprites siempre miran a la cámara en un ángulo perpendicular.
- Cuando Unity crea un nuevo sprite, emplea la utilidad zación de una textura.
- Esta textura luego se transfiere a un nuevo GameObject, que luego se adjunta a un componente Sprite Renderer. Esto hace que nuestro gameObject sea aparente con nuestra textura y le da propiedades que controlan cómo se muestra en la pantalla.
- Para hacer un sprite en Unity, primero debemos darle una textura al motor.
- Comencemos por hacer nuestra textura. Seleccione un archivo de imagen típico, como PNG o JPG, guárdelo y arrastre la imagen a la sección Activos de Unity.
- Arrastre la imagen de la carpeta Activos a la Jerarquía de escena. Veremos que en cuanto soltamos el botón del ratón aparece en la lista un nuevo GameObject con el nombre de nuestra textura.

- Al crear un sprite, mantenga los siguientes puntos en mente:
  - Estamos agregando un activo a Unity arrastrándolo desde una fuente externa.
  - Debido a que este recurso es una imagen, se convierte a una textura.
  - Al deslizar esta textura en la jerarquía de la escena, creamos un nuevo GameObject y un Sprite Renderer con el mismo nombre que nuestra textura.
  - Este renderizador de sprites dibuja la imagen en el juego usando esa textura. En nuestro escenario, ahora hemos incluido un sprite.

CAMBIO DE SPRITES EN UNITY

El sprite que acabamos de importar también puede modificarse de varias maneras para modificar su apariencia.

Se puede ver una barra de herramientas en la esquina superior izquierda de la interfaz del motor.

Repasemos ahora las funcionalidades de los botones:

- La herramienta Mano se utiliza para navegar por la escena sin interferir con ninguno de los elementos.
- Luego está la herramienta Mover. Esto se usa para mover elementos en el entorno del juego.
- Tenemos la herramienta Rotar en el medio, que nos permite rotar cosas junto con el eje Z del mundo del juego (u objeto principal).

- La herramienta Escalar se coloca a la derecha. Esta herramienta le permite cambiar el tamaño (escala) de los objetos a lo largo de ejes específicos.
- Finalmente, está la herramienta Rect. Esta herramienta funciona de manera similar a una combinación de las herramientas Mover y Escalar; sin embargo, es propenso a perder precisión. Es más beneficioso para organizar los componentes de la interfaz de usuario (UI).

A medida que crece la complejidad del proyecto, estas herramientas se vuelven más valiosas.

TRANSFORMA Y OBJETO CRIANZA EN UNIDAD

Cuando comenzamos, hablamos sobre cómo la transformación de un gameObject es probablemente su componente más importante.
En este capítulo, analizaremos el elemento en profundidad.

También se nos presentará la noción de Object Parenting.

Las transformaciones tienen tres propiedades discernibles: posición, rotación y escala. Cada uno de ellos tiene tres valores posibles para cada uno de los tres ejes. Cuando se trata de ubicación, los juegos 2D a menudo no enfatizan el eje Z. El eje Z se usa más comúnmente en juegos 2D para crear paralaje.

Los atributos de rotación proporcionan la cantidad de rotación (en grados) que tiene un objeto sobre ese eje sobre el entorno del juego o el objeto principal.

Cuando se relaciona con su tamaño original o nativo, la escala de un objeto determina qué tan grande es.

Como ejemplo, considere un cuadrado con dimensiones de 2 × 2. Si escalamos este cuadrado por 3 contra el eje X y 2 contra el eje Y, obtenemos un cuadrado de 6 × 4.

¿QUÉ ES EXACTAMENTE LA CRIANZA OBJETIVA?

Los objetos en Unity siguen una arquitectura jerárquica.
Los GameObjects pueden convertirse en "padres" de otros GameObjects utilizando este mecanismo.

Cuando un GameObject tiene un padre, todas las modificaciones de transformación se realizan sobre otro GameObject en lugar del mundo del juego.

Por ejemplo, un objeto sin padre colocado en (10, 0 y 0) estará a 10 unidades de distancia del centro del globo del juego.

Sin embargo, un gameObject con un padre en (10, 0, 0) considerará que la ubicación actual del padre es el centro.

Es tan básico como arrastrar y soltar GameObjects en el padre designado para criarlos. Un elemento "secundario" se representa en la lista de objetos mediante una pequeña sangría y una flecha junto al objeto principal.

Parenting GameObjects tiene una variedad de aplicaciones.
Por ejemplo, todas las numerosas secciones de tanques pueden tener un solo GameObject llamado "tanque".

Como resultado, cuando este GameObject padre "tanque" se mueve, todas las piezas se mueven con él, ya que su ubicación se actualiza continuamente de acuerdo con su padre.

ACTIVOS INTERNOS DE LA UNIDAD

Además de importar recursos externos desde otras aplicaciones, como archivos de audio, fotos, modelos 3D, etc., Unity le permite crear recursos internos. Estos activos se desarrollan dentro de Unity y no requieren una aplicación adicional para crear.

Los siguientes son algunos ejemplos notables de activos:

- **Escenas:** Sirven como "niveles".
- **Animaciones:** Incluyen información sobre el ani maciones de un gameObject.
- **Materiales:** Estos dictan cómo la iluminación influye en el aspecto de un objeto.
- **Scripts:** Este es el código que se escribirá para los gameObjects.
- **Prefabs:** Estos sirven como “modelos” para GameObjects, lo que les permite ser producidos durante el tiempo de ejecución.

Los marcadores de posición, los sprites y los modelos también son activos esenciales. Se utilizan cuando necesitamos gráficos y modelos rápidos de marcador de posición para actualizarlos más tarde con gráficos y modelos reales.

Haga clic con el botón derecho en la carpeta Activos y seleccione Crear para crear un activo interno. Haremos un Triángulo y un Cuadrado en este ejemplo. Desplácese hacia abajo hasta la sección Sprites y elija Triángulo.

Repita para Square y debería tener dos gráficos nuevos activos.

GUARDAR Y CARGAR ESCENAS EN UNIDAD

Cuando hayamos completado una cantidad sustancial de trabajo, querremos guardar nuestro progreso. En Unity, usar Ctrl + S no guardará nuestro proyecto.

Todo en Unity tiene lugar en escenas. También se requiere guardar y cargar; debemos guardar nuestro trabajo actual como una escena (extensión .unity) en nuestros recursos.

Pongámoslo a prueba. Si presionamos Ctrl + S y nombramos nuestra escena, veremos un nuevo activo en nuestra área de Activos. El archivo de la escena se puede encontrar aquí.

Tratemos de hacer una nueva escena ahora. Para hacerlo, vaya al menú Activos y seleccione Crear -> Escena. Dale un nombre a nuestra nueva escena y presiona la tecla Enter.

Las escenas se pueden cargar en el editor haciendo doble clic en ellas en el modo Editor (mientras el juego no se está ejecutando).
Cuando cargamos una escena con modificaciones no guardadas en nuestra escena actual, se nos pedirá que guardemos o descartemos nuestros cambios.

NUESTRO PRIMER GUIÓN

Importar fotos y que permanezcan inmóviles en nuestro juego no nos llevará muy lejos. Podría ser un gran marco para fotos, pero no es un juego.

Se requiere secuencias de comandos para crear juegos en Unity.
La creación de scripts es el proceso de creación de bloques de código que se adjuntan a GameObjects en la escena como componentes.
Las secuencias de comandos se encuentran entre las herramientas más efectivas a nuestra disposición, con la capacidad de hacer o deshacer un buen juego.

La creación de scripts en Unity se realiza utilizando C# o la implementación de JavaScript de Unity, conocida como UnityScript (aunque, con el ciclo de 2018, UnityScript actualmente está entrando en su fase de obsolescencia; por lo tanto, no se recomienda utilizarlo). Usaremos C# para el resto de esta serie.

Para crear un nuevo script, haga clic con el botón derecho en la carpeta Activos y seleccione Crear -> Script C#. También podemos usar la pestaña Activos en la barra superior del motor.

Debería aparecer un nuevo activo cuando creamos un nuevo script.
Por el momento, mantenga el nombre como está y haga doble clic en él.

Junto con la ejecución del script, debería iniciarse nuestro entorno de desarrollo integrado (IDE) predeterminado.

Echemos un vistazo más de cerca a lo que es:

```
utilizando System.Collections;
usando System.Collections.Generic;
utilizando UnityEngine;
clase pública NewBehaviourScript: MonoBehaviour
{

// Esto debería usarse para el inicio
vacío Inicio () {

}
// La función de actualización se llama una vez cada marco

actualización nula ()
{
}
}
```

Nuestro nombre de script aparecerá como una clase derivada de MonoBehaviour. ¿Qué es exactamente MonoBehaviour? Es un extensa colección de clases y métodos. Ayuda en el desarrollo de todos los scripts en Unity de una forma u otra. Cuantos más programas se creen en Unity, más descubriremos lo útil que es MonoBehaviour.

A medida que continuamos, tenemos dos scripts privados sin tipos de retorno, en particular los procedimientos de inicio y actualización. La función Start se llama una vez para el primer cuadro en el que el gameObject en el que se llama está activo en la escena.

Después de la función de inicio, el método de actualización ejecuta cada fotograma del juego. Por lo general, los juegos de Unity se ejecutan en

60 FPS (fotogramas por segundo), lo que significa que la función Actualizar se llama 60 veces por segundo mientras el elemento está activo.

Puede utilizar secuencias de comandos de Unity para acceder a la clase MonoBehaviour completa, así como a la funcionalidad clave de C#, como colecciones genéricas, expresiones lambda y análisis XML, por mencionar algunas.

ESCRITURA BÁSICA DEL MOVIMIENTO EN UNIDAD

En función de la entrada del usuario, desarrollaremos un código que haga que un gameObject se mueva hacia arriba, hacia abajo, hacia la izquierda y hacia la derecha. Debería facilitarnos la comprensión del flujo de trabajo de secuencias de comandos de Unity.

Tenga en cuenta que cada GameObject tiene al menos un componente: Transform. Lo que es único acerca de Transform de gameObject es que aparece como una variable en el lado de creación de secuencias de comandos de Unity, lo que nos permite manipularlo mediante código. Esto no se limita a la Transformación; todos los componentes de Unity tienen atributos a los que se puede acceder a través de variables en secuencias de comandos.

Comencemos con el guión de movimiento. Cree un nuevo guión y llámelo "Movimiento".

Ahora, inicie el script y debería ver lo mismo información como en la lección anterior.

Hagamos una variable flotante pública de velocidad. Crear una variable pública en Unity tiene varios beneficios:

- La variable aparece como un campo editable dentro del editor, eliminando la necesidad de modificar los valores en el código manualmente.

```
Movimiento de clase pública: MonoBehaviour {

velocidad de flotación pública;
}
```

Este script debería compilarse en Unity si lo guardamos sin tocar las otras funciones.

Luego, desde los Activos, arrastre y suelte el script en el GameObject. Si lo hacemos bien, deberíamos ver las propiedades del GameObject.

Podemos usar la función actualizar () en lugar de iniciar () ya que el valor de la velocidad es configurable y no necesita modificarse en el código todo el tiempo ().

Considere los siguientes objetivos para el método Update:

- Examinar la entrada del usuario.
- Lea las instrucciones de entrada si hay alguna.
- Cambiar la transformación de los valores de posición del elemento en función de su velocidad y dirección. Usaremos el siguiente código:

```
actualización nula ()
{
flotante hola = Entrada.
GetAxisRaw("Horizontal");
float vi = Entrada.GetAxisRaw("Vertical");
gameObject.transform.position = new Vector2
(transform.position.a + (hola *
velocidad), transform.position.b + (vi * velocidad));
```

Veamos ahora rápidamente el código.

Primero, creamos una variable de punto flotante llamada hola (por horizontal), cuyo valor está determinado por la Entrada.

El método GetAxisRaw. Dependiendo de la pulsación de tecla del jugador en las flechas arriba/abajo/izquierda/derecha, esta función devuelve ÿ1, 0 o 1.

La clase Input está a cargo de recibir la entrada del usuario en forma de pulsaciones de teclas, entrada del mouse, entrada del controlador, etc. La técnica GetAxisRaw es un poco más difícil de comprender, por lo que volveremos a ella más adelante.

La ubicación de nuestro gameObject luego se actualiza a un nuevo punto determinado mediante la construcción de un nuevo Vector2. El Vector2 requiere dos argumentos, que son los valores de x e y. Suministramos el total de la ubicación actual del objeto y la velocidad para el valor de x, agregando así una cierta cantidad a su posición cada cuadro que se presiona la tecla.

Regrese a Unity y guarde este script. Unity actualizará automáticamente todas las secuencias de comandos después de una compilación exitosa, por lo que no tendremos que volver a adjuntar la secuencia de comandos repetidamente.

Después de eso, modifica el valor de la velocidad en los atributos de GameObject a 0.8. Esto es importante ya que un número mayor hará que el jugador se mueva demasiado rápido.

Ahora, presiona el botón Play para ver nuestro primer minijuego. en acción.

Intenta moverte usando las teclas de flecha. Simplemente presione el botón Reproducir nuevamente para finalizar el juego. También podemos cambiar la velocidad en tiempo real, por lo que no tenemos que detener y encender la máquina todo el tiempo.

ENTENDER LAS COLISIONES EN LA UNIDAD

Las colisiones en Unity se separan del Sprite real, se conectan como componentes distintos y se calculan de forma independiente. Investiguemos ahora la razón detrás de esto.

Cada GameObject en tu juego es un GameObject. Incluso los mosaicos individuales que componen su nivel son GameObjects por derecho propio.

Cuando consideramos que cada componente es un GameObject, vemos que puede haber miles de GameObjects en una escena, todos interactuando de alguna manera.

Si Unity introdujera colisiones en cada uno de los GameObject, sería impracticable que el motor calculara las colisiones para cada uno de ellos.

Seguiremos adelante y construiremos un "muro" rudimentario contra el cual nuestro personaje jugador puede chocar. Cree otro sprite y amplíelo con la herramienta Rect para hacer esto.
Además, lo teñiremos de rojo usando la propiedad Color del componente Sprite Renderer.

Ahora, en el Inspector, seleccione Agregar componente e ingrese "Box Collider 2D". Al hacer clic en el primer componente, debería surgir uno nuevo.

Aparecerá una línea verde brillante alrededor de la circunferencia de nuestro GameObject. Este es el punto en el que dos objetos chocan. Determina la forma principal de elementos colisionables.

Repita el proceso con nuestro GameObject móvil.

Por supuesto, las colisiones en Unity no se limitan solo a las cajas. Pueden tomar muchas formas y tamaños diferentes, y no son necesariamente réplicas precisas de los atributos del objeto. También pueden tener forma poligonal.

Es relativamente poco común que los desarrolladores y diseñadores empleen formas aproximadas en los bordes de colisión para simplificar los colisionadores y eliminar los cálculos adicionales del motor.
Pronto aprenderemos a usar nuestros colisionadores para hacer diferentes formas y tamaños.

Ahora que hemos establecido nuestros límites de colisión, presione reproducir para ver cómo funciona. Nuestro elemento móvil no está funcionando con normalidad, como veremos.

CUERPOS RÍGIDOS Y FÍSICA EN UNIDAD

Ahora cambiaremos los valores de la posición del GameObject directamente. Simplemente agregamos un valor a la posición cuando el jugador presiona una tecla. Necesitamos un método para que el jugador se mueva para que responda adecuadamente a los bordes y otros GameObjects.

Para hacerlo, primero debemos definir rigidbodies. Los Rigidbodies son componentes de GameObject que les permiten responder a la física en tiempo real. Esto incluye las respuestas a las fuerzas y la gravedad, así como la masa, la resistencia y la velocidad.

Simplemente haga clic en Agregar componente y escriba Rigidbody2D en el área de búsqueda para agregar un Rigidbody a su GameObject.

Al seleccionar Rigidbody2D, podemos vincular el componente a nuestro GameObject. Notaremos que se han abierto varios campos más ahora que está conectado.

El GameObject caerá verticalmente hacia abajo debido a la gravedad si se utilizan los parámetros predeterminados. Establezca la escala de gravedad en 0 para evitar esto.

Debido a que GameObject aún no tiene nada que ver con su componente de física, no habrá una diferencia observable cuando juguemos.

Reabrimos nuestro código y reescribámoslo para solucionar nuestro problema:

```
Movimiento de clase pública: MonoBehaviour {

velocidad de flotación pública;
cuerpo público Rigidbody2D;
// La función de actualización se llama una vez cada
 cuadro
actualización nula ()
 {
```

```
flotar hola = Input.GetAxisRaw("Horizontal");
float vi = Entrada.GetAxisRaw("Vertical");
cuerpo.velocidad = new Vector2(hi * velocidad, vi * velocidad);

}
}
```

Podemos ver que en las definiciones, hacemos referencia a un Rigidbody2D, y nuestro método de actualización usa esa referencia en lugar de la transformación del Objeto. Esto significa que al Rigidbody ahora se le ha asignado la tarea de moverse.

Debido a que no hemos establecido nada en la referencia del cuerpo, podemos anticipar generar una NullReferenceException. Si compilamos y ejecutamos el juego tal como está, obtendremos el siguiente error en la esquina inferior izquierda del editor.

Veamos el componente generado por el script para ver cómo podemos remediarlo. Recuerde que los lazos de propiedad pública en la Unidad producen sus campos.

Juega después de aumentar el ritmo a aproximadamente 5.

Nuestras colisiones ahora funcionarán correctamente.

LÍMITES DE COLISIÓN PERSONALIZADOS EN UNITY

Comencemos con el Box Collider. El Box Collider (2D) es un rectángulo con cuatro lados móviles. Haga clic en el cuadro en el componente del Colisionador.

El colisionador mostrará cuatro "asas". Podemos cambiar el tamaño de estos controladores arrastrándolos.

Unity determina la mejor coincidencia factible para la forma del colisionador para las formas básicas, suponiendo que seleccionemos la correcta. Elegir el colisionador de círculos en un sprite circular, por ejemplo, lo emparejará con su radio.

Unity se esforzará por construir la forma de colisión más simple pero más sofisticada para formas cada vez más complejas. Debemos usar el Polygon Collider 2D para esto.

Intente hacer clic en el botón Editar colisionador y experimente con los colisionadores.

ENTENDIENDO PREFABRICADOS Y INSTANTACIÓN EN UNIDAD

Durante el juego, es fundamental poder crear instancias y eliminar objetos. Instanciar simplemente implica traer algo a la existencia. Los elementos "generan" en el juego, los adversarios mueren, los componentes de la GUI desaparecen y los escenarios se cargan en el juego en todo momento. Saber cómo deshacerse de manera efectiva de las cosas innecesarias y traer las que hacemos se vuelve aún más importante.

Primero definamos la prefabricación. Se cree que los prefabricados son esenciales para comprender cómo funciona la creación de instancias en Unity.

Los prefabricados son planos para GameObjects. Los prefabricados son, de alguna manera, un duplicado de un GameObject que se puede formar y colocar en una escena incluso si el GameObject no existía en el momento en que se construyó la escena; en otras palabras, los prefabricados se pueden usar para construir GameObjects dinámicamente.

Arrastra el GameObject relevante desde nuestra jerarquía de escenas a los Activos del proyecto para crear un prefabricado. Ahora, en nuestro script, llamamos a la función Instanciar() para crear un GameObject.
Este método MonoBehaviour acepta un GameObject como parámetro para determinar qué GameObject crear/
duplicar. También proporciona varias anulaciones para modificar la transformación del objeto recién creado y la crianza.

Veamos si podemos hacer un nuevo hexágono cada vez que presionemos la tecla Espacio. Cree un nuevo script llamado Instanciador y

lanzarlo Ingrese el siguiente código en el método de actualización.
En este caso, estamos utilizando la función GetKeyDown de la clase Input para ver si el jugador presionó un botón en particular durante el cuadro anterior. Como queremos que siga comprobándose, lo ponemos en Actualizar, que se ejecuta 60 veces por segundo.
Si la tecla proporcionada por la enumeración KeyCode (que especifica todas las teclas posibles en un teclado convencional) se presiona en ese cuadro, la función GetKeyDown devuelve verdadero.

```
Instanciador de clase pública: MonoBehaviour {

público GameObject Hexágono1;
// La función de actualización se llama una vez cada
cuadro
Actualización nula ()
{
if (Input.GetKeyDown(KeyCode.Space)) {

Instanciar(Hexágono1);
}
}
}
```

La declaración pública de GameObject en la parte superior genera un espacio idéntico a nuestros cursos anteriores para Rigidbody2D.
Esta ranura, sin embargo, solo acepta prefabricados (en tiempo de edición) y gameObjects (en tiempo de ejecución).

Guarde el script y espere a que se compile. Después de eso, vaya al menú contextual de nuestra jerarquía de objetos y elija Crear vacío para crear un GameObject nuevo y vacío.

Asigne a este Objeto un nombre fácil de recordar, como Objeto Instatiador, conéctelo con nuestro script recién creado. Arrastra el prefabricado que hicimos a la ranura que aparece para el GameObject.

Cuando ejecutemos el juego ahora, presionar la barra espaciadora generará un nuevo objeto hexagonal similar al que usamos para construir la casa prefabricada. En la jerarquía de objetos, podemos ver cómo se genera cada hexágono. No podemos verlos en el juego porque todos se están haciendo uno tras otro en este momento.

DESTRUCCIÓN DE GAMEOBJECTS EN UNITY

Es tan crucial destruir GameObjects como crearlos. Esta sesión nos enseñará a destruir GameObjects.

Afortunadamente, destruir GameObjects es tan simple como crearlos. Todo lo que necesita es una referencia al objeto a destruir y luego llamar a la función Destroy() con esa referencia como argumento.

Intentemos ahora crear cinco hexágonos que se autodestruirán cuando se presione una tecla específica.

Abra Visual Studio y cree un nuevo script llamado HexagonDestroyer. Para comenzar, crearemos una variable KeyCode pública. Un KeyCode se usa para indicar una tecla en un teclado estándar y la clase Input lo utiliza en sus métodos. Podemos hacer que esta variable sea accesible a través del editor haciéndola pública como hicimos anteriormente con Rigidbody y Prefabs. Al hacer pública la variable, no necesitamos codificar valores como "KeyCode.A" en el código. Podemos hacer que el código sea tan versátil como queramos incluyendo tantos objetos como deseemos.

```
clase pública HexagonDestroyer: MonoBehaviour {

KeyCode público keyToDestroy;
// La función de actualización se llama una vez cada
cuadro
```

```
Actualización nula () {

if (Input.GetKeyDown(keyToDestroy)) {

Destruir (objeto de juego);
}
}
}
```

Observe cómo utilizamos la variable "gameObject" (g minúscula, O mayúscula) en el procedimiento.

Esta nueva variable gameObject (de tipo GameObject) hace referencia al gameObject con el que está asociado este script. Si conectamos este script a muchos objetos, todos reaccionarán de la misma manera siempre que esta variable esté presente.

Sin embargo, no los mezcles.

- La clase GameObject con G mayúscula y O cubre todos los GameObjects y ofrece métodos básicos como Instanciar, Destruir y formas de obtener componentes.
- La instancia particular de GameObject, marcada con g minúscula y O mayúscula, se refiere al gameObject con el que ahora se adjunta este script.

Ahora construyamos nuestro código y volvamos a Unity.

Ahora crearemos un nuevo sprite hexagonal y le vincularemos nuestro script. Luego, en la jerarquía, haga clic con el botón derecho en gameObject y seleccione Duplicar. En la jerarquía, se produce un nuevo sprite; utilice la herramienta Mover para cambiar su posición. Repita los procedimientos para hacer más hexágonos.

Examine los componentes del guión de cada hexágono haciendo clic en él. Ahora podemos programar un GameObject para

destruirse a sí mismo cuando se pulsa una tecla en particular. Por ejemplo, hagamos cinco hexágonos y programémoslos para que exploten cuando se presionen las teclas A, S, D, F y G.

Podemos colocar la misma tecla en numerosos hexágonos, y todos ellos se destruirán al mismo tiempo cuando se presione la tecla; este es un ejemplo de la referencia gameObject, que podemos usar para referirnos a objetos particulares usando el script sin tener que configurarlos por separado.

Se puede poner la misma llave en numerosos hexágonos, y todos se destruirán al mismo tiempo cuando se presione la llave; este es un ejemplo de cómo utilizar la referencia gameObject, que podemos usar para referirnos a objetos particulares en el script sin tener que configurarlos por separado.

Es fundamental darse cuenta de que la destrucción de un GameObject no hace que el objeto se rompa o explote. En términos del juego (y su programación), la destrucción de un elemento simplemente (e instantáneamente) termina con su existencia. Los enlaces a este objeto y sus referencias simplemente ya no funcionan, y tratar de acceder o usar generalmente dará como resultado errores y bloqueos.

CORUTINAS EN UNIDAD

Al crear juegos con Unity, las herramientas más valiosas son las rutinas. Considere la siguiente pieza de código para comprender mejor de qué se tratan las corrutinas.

```
IEnumerator MyCoroutineMethod() {

//
rendimiento del código devuelve nulo;
}
```

En general, cuando llamamos a una función en Unity (o C#, para el caso), la función se ejecutará de principio a fin. En términos de su código, esto es lo que consideraría un comportamiento "típico". Sin embargo, hay ocasiones en las que deseamos ralentizar deliberadamente una función o hacerla esperar durante un período más prolongado que la fracción de segundo durante la que se ejecuta.

Una corrutina puede hacer precisamente eso: una corrutina es una función que puede esperar y cronometrar su actividad y pausarla por completo.

Mira algunos ejemplos de cómo funciona una rutina.

Supongamos que queremos crear un cuadrado que alterna entre rojo y azul en intervalos de un segundo.

Para empezar, haremos un sprite. Luego, crea un nuevo script.

y llámelo ColorChanger.

Recibimos una referencia al Sprite Renderer del sprite en este script. Sin embargo, obtendremos el componente de manera diferente. En lugar de arrastrar y soltar el componente en una ranura, le pediremos al script que detecte el elemento en sí.

Esto se logra mediante la función GetComponent, que devuelve el primer componente coincidente encontrado.

Podemos usar esta función para identificar automáticamente y obtener una referencia a nuestro renderizador porque solo utilizamos un Sprite Renderer por objeto.

Recuerda que el renderizador es el encargado de hacer visible el sprite en pantalla.

El renderizador contiene una propiedad de color que afecta el color global del sprite; este valor debe ser cambiado. Hacer públicos los valores de Color nos permite seleccionarlos usando el editor en el selector de color predeterminado de nuestro sistema operativo.

```
SpriteRenderer privado srr;
Color público color1;
Color público color2;
```

```
vacío Inicio () {

srr = GetComponent<SpriteRenderer>();
StartCoroutine(ChangeColor());
}
IEnumerator ChangeColor() {

mientras (verdadero)
{
if (srr.color == color1)
sr.color = color2;
más
srr.color = color1;
yield return new WaitForSeconds(3);
}
}
```

Ahora usaremos un bucle while para capturar nuestra función de rutina.

En C#, simplemente construimos un método que devuelve un IEnumerator para establecer una rutina. También se requiere una declaración de rendimiento. La línea de retorno de rendimiento es única en el sentido de que informa a Unity que pause el script y lo reanude en el siguiente cuadro.

Existen varios métodos para obtener una devolución, uno de los cuales es crear una instancia de la clase WaitForSeconds. Esto hace que la rutina se detenga durante una cierta cantidad de segundos reales antes de continuar.

Construyamos nuestro código y volvamos a Unity. Seleccione nuestros colores alternos y presione el botón de reproducción. Nuestro objeto ahora debería alternar entre los dos colores a los tres segundos.

intervalos También podemos hacer del intervalo una variable pública y variar la frecuencia de los cambios de color.

Las corrutinas se usan comúnmente para procedimientos cronometrados como el que acabamos de realizar. Cada uno de los métodos WaitForX tiene su propio conjunto de aplicaciones. Las corrutinas también se pueden usar para ejecutar programas "al margen" que se ejecutan de forma independiente mientras se ejecuta el juego. Esto es útil para cargar elementos fuera de la pantalla de un gran nivel mientras el jugador comienza en una ubicación específica, por ejemplo.

LA CONSOLA EN UNIDAD

Los resultados del Desarrollador se leerán en la Consola. Estas salidas se pueden usar para probar rápidamente bits de código que no requieren más funciones de prueba.

En la consola predeterminada, hay tres categorías de mensajes. La mayoría de los estándares del compilador se pueden asociar con estos mensajes:

- **Errores:** Los errores son problemas o excepciones que pro impedir que el código funcione.
- **Señales de advertencia:** las advertencias son defectos que no impedirán la ejecución de su código, pero pueden causar problemas durante la ejecución.
- **Mensajes:** Los mensajes son salidas que comunican información al usuario; rara vez revelan preocupaciones.

Incluso podemos indicarle a la Consola que muestre nuestros mensajes, precauciones y fallas. Haremos uso de la clase Debug.

La clase Debug es un componente de MonoBehaviour que proporciona métodos para escribir mensajes en la consola, muy parecidos a cómo produciríamos mensajes de salida típicos en nuestras aplicaciones iniciales.

La Consola se puede encontrar en la pestaña sobre la región de Activos. Los resultados de la consola son más valiosos para el programador que para el usuario final o el jugador.

Enviemos un mensaje importante a la Consola. Esto nos alertará si se presiona la tecla Espacio. Usaremos el método Log para esto, que acepta un objeto como argumento, que llenaremos con una cadena.

Podemos comenzar desde cero o editar un guión existente.

```
actualización nula ()
{
si (Entrada.GetKeyDown(KeyCode.Space))
Debug.Log("¡Tecla espaciadora presionada!");
}
```

Después de guardar, compilar y ejecutar este código (conectándolo a un GameObject, por supuesto), intente presionar la barra espaciadora. Nuestro mensaje se escribirá si hacemos clic en la pestaña Consola.

De manera similar, los métodos LogWarning y LogError se pueden usar para generar advertencias y errores, respectivamente. Estos serán útiles para probar pequeños fragmentos de código sin tener que implementarlos.

INTRODUCCIÓN AL AUDIO EN UNIDAD

Hay una razón por la que el audio es tan importante en los juegos; es esencial para agregar valor estético al juego. Desde el primer Pong, se pueden escuchar pitidos y pitidos cuando la pelota

golpea alternativamente las paletas. En su momento, era una muestra de onda cuadrada corta muy rudimentaria, pero ¿qué más se le puede pedir al abuelo de todos los videojuegos?

Muchos factores influyen en cómo escuchamos el sonido en la vida real, incluida la velocidad del elemento, el tipo de contexto en el que se encuentra y la dirección de la que proviene. Una variedad de variables podría ejercer una presión indebida sobre nuestro motor. En su lugar, intentamos imaginar cómo nuestro sonido podría funcionar en nuestro juego y luego diseñar en torno a eso.

Esto es especialmente notable en los juegos 3D porque hay tres ejes con los que lidiar.

Tenemos componentes especializados de percepción y reproducción de audio en Unity. Estos elementos trabajan juntos para producir un sistema de sonido efectivo que se siente natural en el juego.

Unity ofrece una gran cantidad de herramientas y efectos esenciales, como la reverberación, el efecto Doppler, la mezcla y los efectos en tiempo real, etc.

componentes de audio

Aprenderemos sobre los tres componentes de audio esenciales en Unity.

- **AudioSource:** el componente AudioSource es el componente principal adjunto a un GameObject para que reproduzca sonido. Reproducirá un AudioClip cuando lo active el mezclador, el código o cuando se active de forma predeterminada.

  Un AudioClip no es más que un archivo de sonido que se ha insertado en un AudioSource. Puede ser cualquier formato de archivo de audio estándar, como .mp3, .wav u otros. Un AudioClip también es un componente en sí mismo.

- **AudioListener:** un AudioListener es un componente que escucha todo el audio de la escena y lo envía a los altavoces de la computadora. Sirve como los oídos del juego. Todo el audio que escuchamos es desde la perspectiva de este AudioListener. Para que un escenario funcione correctamente, solo debe estar presente un AudioListener. El Oyente está vinculado a la cámara principal de forma predeterminada. El oyente no tiene atributos expuestos que interesen al diseñador.

- **Filtros de** audio: los filtros de audio se pueden usar para modificar la salida de un AudioSource o la entrada de un AudioListener. Estos son componentes individuales que pueden alterar la reverberación, el coro, el filtrado, etc.
  Cada filtro viene como su componente con configuraciones accesibles para ajustar cómo suena.

Haciendo un ruido

Intentemos construir un botón que emita un sonido cuando se presiona. Para comenzar, crearemos un sprite Circle y lo colorearemos de rojo.

Ahora agreguemos una fuente de audio a este sprite. Debemos proporcionar un sonido para que el elemento lo reproduzca. Hagamos un buen uso de este efecto de sonido:

https://orangefreesounds.com/ding-sfx/.

Arrastre el efecto de sonido a la carpeta Activos.

Cuando Unity importa este activo como un archivo de sonido, lo convierte en un AudioClip automáticamente. Como resultado, podemos arrastrar este clip de sonido directamente desde los Activos a la ranura de Clip de audio en la Fuente de audio de nuestro sprite.

Recuerde anular la selección de "Reproducir al despertar" en el Audio Atributos de origen después de arrastrar el clip de sonido desde el Activos directamente en la ranura Audio Clip en nuestro sprite

Fuente de audio; de lo contrario, el sonido se reproducirá en el instante en que comience el juego.

Comencemos con nuestra codificación. Cree un nuevo script llamado "BellSound" y ejecútelo.

Debido a que nuestra fuente de audio está controlada por código, primero debemos obtener una referencia a ella.

Utilizaremos la función GetComponent nuevamente.

```
clase pública BellSound: MonoBehaviour {

AudioSource miFuente;
// para la inicialización Usa esto
vacío Inicio () {

miFuente = GetComponent<Fuente de Audio>();
}
```

Presentemos ahora la técnica para detectar el objeto en el que se ha hecho clic. MonoBehaviour nos proporciona el método que necesitamos, OnMouseDown. Cuando el mouse está dentro del rango de un colisionador de ese gameObject, se invoca la función.

Conectemos un colisionador a nuestro botón ahora porque no lo he hecho todavía.

No necesitaremos un Rigidbody para este ni necesitaremos acceder a él a través del código. Solo necesita estar presente para que el procedimiento funcione.

Pongamos el enfoque a prueba y veamos si funciona.
Agregue el siguiente código a nuestro script y vincúlelo al botón.

```
void OnMouseDown()
{
Depurar.Registrar("Hacer clic");
}
```

Juega después de guardar y adjuntar el guión. Cuando hacemos clic en el botón, debería aparecer un mensaje en la Consola.

Ahora estamos un paso más cerca de escuchar el sonido. Todo lo que queda es invocar el método Reproducir en el objeto Fuente de audio.

```
void OnMouseDown()
{
miFuente.Play();
}
```

Guarda tu script y ejecútalo en el juego. Cuando presionamos el botón, deberíamos escuchar el sonido.

COMENZANDO CON LA IU EN UNITY

Esta sesión enseñará sobre el proceso de diseño de los componentes de la interfaz de usuario en Unity. Esto proporciona la configuración básica, así como una descripción general de las partes comunes de Unity.

El enfoque para diseñar la interfaz de usuario en Unity difiere un poco del que hemos estado siguiendo hasta ahora. Para empezar, los componentes de la interfaz de usuario no son GameObjects normales y, por lo tanto, no se pueden utilizar como tales. Los elementos de la interfaz de usuario se crean de manera diferente; por ejemplo, un botón de menú que parece correcto en una resolución de 4:3 puede aparecer estirado o deformado en una pantalla de 16:9 si no se configura correctamente.

En Unity, los componentes de la interfaz de usuario no se ponen en escena inmediatamente. Se agregan constantemente como elementos secundarios de un GameObject en particular conocido como Canvas. El lienzo sirve como una "hoja de dibujo" para la interfaz de usuario de la escena, donde se representan todos los componentes de la interfaz de usuario. Sin un lienzo existente, la creación de un elemento de interfaz de usuario desde el menú contextual Crear producirá uno para nosotros.

Echemos un vistazo al Canvas GameObject ahora para aprender sobre los nuevos componentes:

Rect Transform en la parte superior parece contener una serie de características adicionales de las que carece un Transform de GameObject convencional.

Mientras que Transform de un GameObject estándar representa un punto imaginario en el espacio 3D, RectTransform especifica un rectángulo imaginario. Esto implica que necesitaremos más atributos para determinar dónde está el rectángulo, qué ancho tiene y cómo está orientado.

Podemos ver varios atributos de rectángulo convencionales como Alto y ancho y dos nuevas propiedades de anclaje.

Los anclajes son puntos en el lienzo en los que otras cosas pueden "fijarse". Esto implica que si un elemento de la interfaz de usuario (por ejemplo, un botón) se adjunta al lienzo a la derecha, cambiar el tamaño del lienzo mantendrá el botón a la derecha relativa del lienzo en todo momento.

Por defecto, no podremos cambiar la forma del área del lienzo, que será un rectángulo colosal alrededor de nuestro escena.

El componente Canvas viene a continuación. Este es el componente maestro, que tiene algunos parámetros globales sobre cómo se representa la interfaz de usuario.

El modo de renderizado es la primera opción que encontramos. Esta propiedad especifica el mecanismo para dibujar el lienzo en la pantalla del juego.

En el menú desplegable, tenemos tres posibilidades. Dejar aprendamos sobre las posibilidades en las partes que siguen.

- **Superposición para espacio de pantalla:** este es el modo más común para menús, HUD, etc. Representa la interfaz de usuario sobre todo lo demás en la escena, exactamente como

está dispuesto y sin excepción. También ajusta la interfaz de usuario con elegancia a medida que cambia el tamaño de la pantalla o la ventana del juego. Este es el modo de renderizado predeterminado de Canvas.

- **Espacio de la pantalla de la cámara:** la cámara produce un plano de proyección imaginario a una distancia predeterminada de la cámara y proyecta todo el contenido de la interfaz de usuario en él. Esto implica que la apariencia de la interfaz de usuario en la escena está muy influenciada por la configuración de la cámara, como la perspectiva, el campo de visión y pronto.

- **Espacio mundial: los** elementos de la interfaz de usuario actúan en el modo Espacio mundial como si fueran GameObjects regulares colocados en el mundo. Debido a que son similares a los sprites, a menudo se emplean como parte del mundo del juego en lugar de para el jugador, como monitores y pantallas en el juego.
  Debido a esto, podemos cambiar la configuración de Canvas RectTransform directamente en este modo.

Canvas Scaler es un conjunto de configuraciones que nos permite modificar la escala y el aspecto de los componentes de la interfaz de usuario; nos permitirá especificar cómo los elementos de la interfaz de usuario cambian de tamaño cuando cambia el tamaño de la pantalla.

Por ejemplo, los componentes de la interfaz de usuario pueden tener el mismo tamaño independientemente y en proporción al tamaño de la pantalla, o pueden escalarse siguiendo una Resolución de referencia.

Graphics Raycaster es el principal responsable de la transmisión de rayos (enlace a la documentación de Unity para Raycasting) los componentes de la interfaz de usuario y de garantizar que las actividades iniciadas por el usuario, como hacer clic y arrastrar, funcionen correctamente.

EL BOTÓN DE LA UNIDAD

Aprenderemos cómo agregar componentes de interfaz de usuario a nuestra escena y cómo operar con ellos.

Comencemos con un botón. Para agregar un botón, haga clic con el botón derecho en cualquier parte de la Jerarquía de escenas y seleccione Crear -> IU -> Botón. Si aún no tenemos un Canvas y un EventSystem, Unity creará uno para nosotros y configurará el botón dentro del Canvas.

Recuerde que el tamaño del lienzo es independiente del tamaño de la cámara en el modo de renderizado Superposición, que es el modo predeterminado. Podemos probarlo yendo a la pestaña Juego.

Cuando juguemos el escenario, notaremos que el botón ya tiene una funcionalidad estándar, como detectar el movimiento del mouse y cambiar de color cuando se presiona.

Para ser útil en la interfaz de usuario, un botón debe tener funcionalidad. Sus atributos se pueden utilizar para implementar esta capacidad.

Hagamos un nuevo script llamado ButtonBehaviour:

```
clase pública ButtonBehaviour: MonoBehaviour {

 en un;
vacío público OnButtonPress()
 {
un++;
Debug.Log("Botón pulsado " veces."); + un + "

 }
 }
```

Creamos una técnica simple que registra cuántas veces presionamos el botón.

Comencemos con un GameObject en blanco y conectemos este script a él. Hacemos esto porque un botón no hace nada por sí mismo; simplemente llama a la función definida en su secuencia de comandos.

Ahora, navegue a las propiedades del botón y busque la propiedad OnClick().

Cuando hacemos clic en el ícono Más en la pestaña inferior, debería aparecer una nueva entrada en la lista.

Este elemento especifica a qué objeto afecta el clic del botón y qué función del script de ese objeto se invoca.
Debido al mecanismo de eventos utilizado en la pulsación del botón, podemos agregar más funciones a la lista para activarlas.

Navegue hasta el menú desplegable Sin función y elige nuestro método OnButtonPress.

(Recuerde que podemos llamarlo como desee; OnButtonPress es solo una convención de nomenclatura predefinida). Debería estar en la sección ButtonBehavior.

Si jugamos el juego ahora, podemos probar el botón, y el consola nos dirá cuántas veces lo presionaste.

ELEMENTO DE TEXTO EN UNIDAD

Incluso si los elementos creados por la comunidad más potentes y eficientes lo superan, la interfaz de usuario de texto inherente de Unity es un hermoso punto de partida para que los principiantes comiencen a desarrollar la interfaz de usuario.

Para nuestros propósitos, el elemento Texto predeterminado es más que adecuado.

El hecho de que el texto sea un elemento único de la interfaz de usuario se debe principalmente a su dinamismo. Imprimir el puntaje actual del jugador en la pantalla, por ejemplo, requiere que el valor numérico del puntaje se convierta en una cadena, a menudo usando la función .toString().

Para agregar un elemento de interfaz de usuario de texto, vaya a la jerarquía de escenas y seleccione Crear -> Interfaz de usuario -> Texto.

En su área de Lienzo, debería aparecer un nuevo elemento de Texto. Cuando observamos sus atributos, podemos ver que tiene varias opciones beneficiosas.

El cuadro de texto, por otro lado, es el más importante. Podemos poner lo que queramos que diga el cuadro de texto en ese campo, pero nos gustaría ir un paso más allá.

Para modificar la fuente del texto, primero, importe el archivo de fuente desde su computadora como un Activo a Unity. No es necesario que un tipo de letra esté vinculado explícitamente a nada en la escena y se puede acceder directamente desde los Activos.

También se puede acceder al elemento de texto a través de secuencias de comandos, lo que enfatiza la importancia de la interfaz de usuario dinámica.

```
utilizando UnityEngine;
utilizando UnityEngine.UI;
clase pública ButtonBehaviour: MonoBehaviour {

en un;
Texto público miTexto;
vacío público OnButtonPress()
{
un++;
myText.text = "Botón presionado " veces.";          + un +     "

}
}
```

La primera modificación que hicimos fue crear una nueva referencia de espacio de nombres. Incluimos el uso de la línea UnityEngine.UI porque esta referencia se utiliza para tratar con los componentes de la interfaz de usuario de Unity.

A continuación, establecemos una variable de texto pública para arrastrar y soltar nuestro elemento de interfaz de usuario de texto.

Finalmente, usamos myText.text para obtener el texto que contiene este elemento de la interfaz de usuario.

Si guardamos nuestro script, notaremos una nueva ranura en nuestro ButtonManager para el elemento Text UI. Arrastre y suelte el gameObject que contiene el elemento Texto en la ranura, luego presione el botón Reproducir.

EL DESLIZADOR EN UNIDAD

Aprenderemos sobre el último elemento de la interfaz de usuario. El control deslizante se usa con frecuencia cuando se debe establecer un valor entre un par de valores máximo y mínimo. Una de las aplicaciones más típicas para esto es controlar el volumen del audio o el brillo de la pantalla.

Para hacer un control deslizante, vaya a Crear -> IU -> Control deslizante. En nuestra escena, debería aparecer un nuevo elemento Slider.

Si vamos a las propiedades del Slider, descubrirás un montón de posibilidades de personalización.

Veamos si podemos construir un control deslizante de volumen a partir de este control deslizante. Para hacerlo, abra el script ButtonBehaviour (cambie el nombre de ButtonManager GameObject ya que ahora hace más que simplemente administrar un botón) e incluya una referencia al Control deslizante. También modificaremos el código un poco más.

```
Comportamiento de botón de clase pública:
MonoComportamiento
{
en un;
Texto público miTexto;
Control deslizante público mySlider;
```

```
actualización nula ()
{
myText.text = "Volumen actual: " mySlider.value;          +


  }
}
```

Comprenda cómo estamos utilizando el método de actualización para mantener actualizado el valor de myText.text.

Marque la casilla "Números enteros" en la configuración del control deslizante y establezca el valor máximo en 100.

Cambiaremos el color del texto usando sus atributos para que se note más.

Repitamos el proceso de arrastrar el Control deslizante GameObject en la nueva ranura y presionando play.

Se recomienda encarecidamente estudiar y experimentar con los otros controles de la interfaz de usuario para descubrir cuáles funcionan con qué método.

MATERIALES Y TONOS EN UNIDAD

En esta sección, estudiaremos un poco de material y shaders. Comenzaremos un nuevo proyecto 3D en lugar del actual 2D para hacer las cosas más transparentes. Esto hará que sea más fácil para nosotros notar los numerosos cambios.

Una vez que hayamos creado el nuevo proyecto, vaya a Jerarquía, haga clic con el botón derecho y seleccione Objeto 3D -> Cubo. Esto colocará un nuevo cubo en el centro de la escena. En la vista de escena, podemos mirar alrededor del cubo haciendo clic con el botón derecho y arrastrando el mouse. También podemos usar la rueda de desplazamiento para acercar y alejar.

Ahora, haga clic en el cubo y examine sus atributos.

El atributo en la parte inferior parece tener una relación de pareja predeterminada rial y un sombreador estándar.

¿Qué es exactamente un material?

Un Material en Unity (y muchos otros elementos de modelado 3D) es un archivo que contiene información sobre la iluminación de un elemento con ese material. Tome nota de cómo una esfera gris representa el material con algo de luz que entra desde la parte superior.

No se deje engañar por el nombre; a El material no tiene nada que ver con la masa, las colisiones o la física en general. Un material define cómo afecta la iluminación a un artículo hecho de ese material.

Intentemos hacer nuestra sustancia. Haga clic con el botón derecho en la sección Activos, seleccione Crear -> Material y llámelo "Mi material".

Estas cualidades son diferentes a todo lo que hemos visto antes.
Esto se debe a que son atributos programados por shaders en lugar de propiedades programadas por materiales.

Los materiales de los que están hechos sus productos son los que los hacen visibles en primer lugar. Incluso en 2D, empleamos un material específico que no requiere tanta iluminación. Por supuesto, Unity lo produce y lo aplica a todo, por lo que ni siquiera somos conscientes de su presencia.

¿Qué es exactamente un sombreador?

Un sombreador es un software que determina cómo se representa cada píxel de la pantalla. Los sombreadores no están escritos en C# ni en ningún otro lenguaje de programación orientado a objetos (OOPS).
Están escritos en GLSL, un lenguaje similar a C que puede enviar instrucciones directas a la GPU para un procesamiento rápido.

EL SISTEMA DE PARTÍCULAS EN UNIDAD

Los sistemas de partículas ayudan en la generación eficiente de una gran cantidad de partículas con una vida útil corta. Estos sistemas tienen su mecanismo de renderizado y pueden generar partículas incluso cuando hay cientos o miles de objetos.

En el Sistema de Partículas, las partículas son una frase ambigua; una partícula es cualquier textura particular, instancia material u objeto formado por el sistema de partículas. Estos no son necesariamente puntos flotando en el espacio (¡aunque podrían serlo!), y pueden emplearse en varios contextos.

Un GameObject mantiene un Sistema de partículas con el componente Sistema de partículas conectado; los sistemas de partículas no requieren ningún Activo para instalarse; sin embargo, pueden ser necesarios varios materiales según el efecto que desee.

Para crear un sistema de partículas, use la opción Agregar componente para agregar el componente Sistema de partículas, vaya a Jerarquía y seleccione Crear -> Efectos -> Sistema de partículas. Esto creará un nuevo GameObject que incluye el sistema de partículas.

Cuando examinemos las características del Sistema de Partículas, notaremos que comprende varios módulos. Solo tres módulos están activos por defecto: Emisión, Forma y Renderizador. Se puede acceder a otros módulos haciendo clic en el pequeño círculo junto a sus nombres.

Es posible que veamos una pequeña flecha negra a la derecha de algunos números. Esto nos da más control sobre los valores de las partículas individuales. Por ejemplo, podemos indicarle al Sistema de partículas que represente partículas aleatorias de diferentes tamaños, como una manguera de agua, configurando el Tamaño de inicio en Aleatorio entre dos constantes.

USO DE LA TIENDA DE ACTIVOS EN UNITY

Asset Store es uno de los activos más potentes de Unity en la industria de motores de juegos; contiene una cantidad significativa de activos, herramientas, scripts e incluso proyectos completos preempaquetados que podemos descargar.

Debemos tener una ID de Unity válida para utilizar la tienda de activos. Si aún no tenemos uno, podemos crear uno en el sitio web de Unity.

Después de crear una ID de Unity, vaya a la pestaña Tienda de activos, que se encuentra en la misma fila que la Vista de escena.

Cuando iniciamos sesión, debería poder ver nuestro nombre de usuario en la esquina superior derecha.

Importaremos el proyecto Survival Shooter Tutorial en este ejemplo. Para hacerlo, lo buscaremos en la pestaña y luego haremos clic en el activo liberado por Unity.

Daremos clic en Descargar y esperaremos a que finalice. El botón Descargar cambiará a Importar; haga clic nuevamente para agregar nuestro nuevo activo al proyecto actualmente activo cuando haya terminado.

Aparecerá una nueva ventana que muestra el contenido del activo recién importado.

Dependiendo de lo que hayamos descargado, puede ser un solo archivo, un grupo de archivos o un árbol completo que contiene jerarquías de carpetas y archivos. Cuando presiona el botón Importar en Unity, automáticamente importará todos los componentes de activos, precisamente lo que queremos. Ahora, dejemos que Unity haga lo suyo haciendo clic en Importar.

Intentar descargar materiales sin pagar por ellos es ilegal y siempre existe el riesgo de virus, problemas o falta de actualizaciones.

Capítulo 3

Trabajando con Escenas y Objetos de juego

Habiendo instalado Unity, comenzaremos con la administración de escenas en este capítulo.

¿QUÉ SON LAS ESCENAS?

En Unity, las escenas son donde trabajas con el material. Son activos que contienen la totalidad o una parte de un juego o programa. Por ejemplo, un juego principal puede construirse en una sola escena, pero un juego más complicado podría requerir una escena por nivel, cada escenario, personajes, obstáculos, decoraciones e interfaz de usuario (UI). El número de escenas que se pueden incluir en un proyecto es ilimitado.

DOI: 10.1201/9781003214755-3

Cuando lanzamos un nuevo proyecto por primera vez, Unity muestra una escena de ejemplo simplemente con una cámara y una luz.

Creación, carga y guardado de escenas

Comencemos por la creación y carga de la escena.

- **Creación de una escena:** existen varios métodos para crear una nueva escena:
  - Para crear una nueva escena a partir de una plantilla de escena específica, utilice el cuadro de diálogo Nueva escena.
  - Sin acceder al cuadro de diálogo Nueva escena, utilice el menú o la ventana del proyecto para crear nuevas escenas a partir de la plantilla de escena básica de su proyecto.
  - Cree una escena directamente desde un script de C# utilizando una plantilla específica.

Uso del cuadro de diálogo Nueva escena para crear una nueva escena: Use el cuadro de diálogo Nueva escena para generar nuevas escenas a partir de plantillas de escena específicas en nuestro Proyecto. El cuadro de diálogo Nueva escena también se puede usar para identificar y administrar plantillas de escena. Consulte el cuadro de diálogo Nueva escena para obtener más información.

El cuadro de diálogo Nueva escena aparece por defecto cuando creamos una nueva escena a través del menú (Archivo > Nueva escena) o usando un atajo de teclado (Ctrl/Cmd + n).

Para crear una nueva escena, siga estos pasos:

- Elija una plantilla del menú desplegable.
- Habilite Load Additively si queremos que Unity cargue la nueva escena de forma aditiva.
- Para crear una nueva escena a partir de la plantilla, haga clic en Crear.

Si no hay dependencias clonables en la plantilla,
Unity carga la nueva escena en la memoria pero no
guardarlo

Si la plantilla contiene dependencias clonables, Unity nos invita a guardarla en un lugar del Proyecto. Unity produce una carpeta con el mismo nombre y ubicación que la nueva escena cuando guardamos la escena. Las dependencias clonables luego se copian en la nueva carpeta y la nueva escena se actualiza para usar los activos clonados en lugar de los activos originales usados por la escena de la plantilla.

- **Uso del menú para crear una nueva escena:** para crear una nueva escena sin activar el cuadro de diálogo Nueva escena, use el menú (Activos > Crear > Escena).

Cuando seleccionamos Nueva escena en el menú, Unity copia la plantilla básica del proyecto y la agrega toda a la carpeta que ahora está abierta en la ventana de la aplicación.

- **Uso de la ventana de proyecto para crear una nueva escena:** Use la barra de menú en la ventana Proyecto para crear una nueva escena sin abrir el cuadro de diálogo Nueva escena.

Navegar a la carpeta en la que deseamos guardar el nuevo
escena.

Seleccione Crear > Escena en el menú contextual haciendo clic derecho en la carpeta en el panel de la izquierda o en un espacio vacío en el panel de la derecha.

Al crear una nueva escena desde el menú, Unity replica la plantilla básica del proyecto y agrega la nueva escena a la carpeta que especificamos.

- **Creación de una nueva escena a partir de un script de C#:** utilice la función Instanciar para generar una nueva escena a partir de un script de C# que utilice una plantilla de escena específica.

```
Tupla < Escena, Elemento de escena >
Plantilla de escena.
Instanciar (SceneTemplateAsset sceneTemplate,
bool loadAdditively, string newSceneOutputPath = null);
```

La función Instanciar crea una nueva escena basada en una plantilla de escena. Devuelve el identificador de escena recién formado, así como el SceneAsset que le corresponde. Este escenario puede crearse de manera aditiva. Si la escena contiene activos que se deben clonar, debemos especificar una ubicación para que Unity guarde la escena en el disco.

- **Nuevos eventos de escena:** cuando creamos una nueva escena usando una plantilla, ya sea a través de un script o usando el cuadro de diálogo Nueva escena, Unity genera un evento. Unity activa este evento una vez que se crea una instancia de la plantilla y después del EditorSceneManager.
  Eventos newSceneCreated y EditorSceneManager.scene Opened.

```
clase pública SceneTemplate1
{
delegado público vacío NewTemplateInstan
tiated(SceneTemplateAsset
sceneTemplateAsset, Escena escena,
SceneAsset sceneAsset, bool aditivoLoad);
```

```
evento público estático
NewTemplateInstantiated
newSceneTemplateInstantiated;
}
```

- **Las escenas se están cargando:** inicie una escena realizando una de las siguientes acciones:
  - Haga doble clic en el activo de la escena en el Proyecto ventana.
  - Elija Archivo > Nueva escena en el menú.
  - Seleccione Archivo > Escenas recientes > [NOMBRE DE LA ESCENA]. del menú.

  Si tenemos modificaciones sin guardar en nuestra escena actual, Unity nos pedirá que guardemos la escena o eliminemos los cambios.

- **Abrir Varias Escenas al Mismo Tiempo:** Podemos editar varias escenas al mismo tiempo.

- **Guardar escenas:** elija Archivo > Guardar escena en el menú, o presione Ctrl + S (Windows) o Cmd + S (Mac) para guardar la escena en la que estamos trabajando ahora (macOS).

Edición multiescena

Esta función nos permite tener varias escenas abiertas en el editor simultáneamente, lo que facilita el manejo de escenas en tiempo de ejecución.

La opción de acceder a varias escenas en el editor nos permite construir mundos de transmisión masivos y mejora la eficiencia mientras trabajamos juntos en la edición de escenas.

Esta sesión repasará:

- La integración de edición multiescena del Editor.
- La interfaz de programación de aplicaciones (API) para secuencias de comandos de Editor y secuencias de comandos de tiempo de ejecución.
- Inquietudes actuales que se conocen.

Seleccione Abrir aditivo de escena en el menú que se muestra para un activo de escena en el editor, o arrastre una o incluso más escenas desde la ventana Proyecto a la Ventana de jerarquía para abrir una nueva escena y agregarla a la lista más reciente de escenas en Jerarquía.

Cuando tenemos muchas escenas abiertas en el editor, el panel de jerarquía muestra el contenido de cada escena por separado. El contenido de cada escena se muestra detrás de una barra divisoria de escenas que muestra el nombre de la escena y guarda
estado.

Las escenas se pueden cargar y descargar mientras están presentes en la jerarquía para exponer u ocultar los objetos del juego almacenados dentro de cada escena. Esto no es lo mismo que agregarlos o eliminarlos del panel de jerarquía.

Si tenemos muchas escenas cargadas, los divisores de escena se pueden comprimir en la jerarquía al contenido de la escena, lo que puede ayudarnos a atravesar nuestra jerarquía.

Al trabajar en varias escenas, cada escena actualizada debe tener sus cambios guardados; por lo tanto, numerosas escenas no guardadas se pueden ver simultáneamente. En la barra divisoria de escenas, las escenas con cambios no guardados tendrán un asterisco junto al nombre.

El menú contextual de la barra divisoria nos permite guardar cada Escena de forma independiente. Guardar los cambios en todas las escenas abiertas es tan simple como seleccionar "Guardar escena" en el menú de archivo o presionar Ctrl/Cmd + S.

La barra de menú en las barras divisorias de escenas nos permite realizar acciones adicionales en la escena seleccionada actualmente.

Para las escenas cargadas, aparece el menú divisor de escenas:

- **Establecer Escena Activa:** Esto nos permite elegir el escenario en el que se crean/instancian nuevos Objetos de Juego. Siempre se debe designar una escena como la escena actual
- **Guardar Escena:** Solo se guardan las modificaciones a la escena especificada.
- **Guardar escena como:** guarda la escena elegida actualmente (junto con los cambios actuales) en un nuevo recurso de escena.
- **Guardar todo:** se guardan los cambios en todas las escenas.
- **Descargar Escena:** La escena se descarga, pero permanece en la ventana Jerarquía.
- **Eliminar escena:** la escena se descarga y se elimina desde la ventana Jerarquía.
- **Seleccionar recurso de escena:** en la ventana Proyecto, seleccione el recurso para la escena.
- **GameObject:** proporciona un submenú para crear GameObjects en la escena especificada. El menú replica los elementos que se pueden crear en el menú principal de GameObject de Unity.

El menú divisor de escenas para escenas descargadas es el siguiente:

- **Cargar Escena:** Carga el contenido de la escena.
- **Quitar Escena:** Saca la escena de la Jerarquía ventana.
- **Elija un recurso de escena:** en la ventana del proyecto, seleccione el recurso para la escena.

Hornear mapas de luz en múltiples escenas

Para hornear datos de mapa de luz para muchas escenas a la vez, abra las escenas que desea hornear, desactive el modo "Auto" en la ventana de iluminación y luego haga clic en el botón Generar.

Los cálculos de iluminación utilizan geometría estática y luces de todas las escenas. Como resultado, las sombras y los rebotes de luz GI funcionarán en todos los escenarios. Por otro lado, los mapas de luz y los datos GI en tiempo real se dividen en datos que se cargan y descargan de forma independiente para cada escena.

Los mapas de luz y los atlas de datos GI en tiempo real se dividen en escenas. Esto implica que los mapas de luz entre escenas nunca se intercambian y se pueden vaciar de forma segura cuando se descarga una escena.

Por el momento, los datos de las sondas de luz siempre se comparten, y todas las sondas de luz para todas las escenas preparadas juntas se cargan en el Mismo tiempo.

Alternativamente, podemos usar la herramienta Lightmapping para automatizar la creación de mapas de luz para varias situaciones.

En un script de editor, use el método BakeMultipleScenes.

Hornear datos de Navmesh con una variedad de escenas

Para hornear datos de navmesh para varias escenas a la vez, abra cada escena y haga clic en el botón Hornear en la barra de navegación.

Ventana. Los datos de la malla de navegación se integrarán en un único objeto que compartirán todas las escenas cargadas. Los datos se guardan en la carpeta con el mismo nombre que la escena actualmente activa (por ejemplo, ActiveSceneName/NavMesh.
activo). Este componente de malla de navegación será compartido por todas las escenas cargadas. Guarde las escenas afectadas después de hornear la malla de navegación para mantener la referencia de escena a malla de navegación de forma permanente.

Alternativamente, podemos usar NavMeshBuilder.
Método BuildNavMeshForMultipleScenes en un script de editor para automatizar la creación de datos de navmesh para numer
nuestras escenas.

Datos de horneado para eliminación de oclusión con varias escenas

Para hornear datos de eliminación de oclusión para muchas escenas simultáneamente, abra las escenas que desea hornear, abra la ventana Eliminación de oclusión (menú: Ventana > Renderización > Eliminación de oclusión) y haga clic en el botón Hornear. Los datos de eliminación selectiva de oclusión se guardan como un activo denominado OcclusionCullingData.
activo en una carpeta con el nombre de la escena actualmente activa.
Activos/ActiveSceneName/OcclusionCullingData.asset, por ejemplo. En cada Escena abierta, se agrega una referencia a los datos. Guarde las escenas afectadas después de hornear los datos de selección de oclusión para que la referencia de datos de escena a oclusión sea duradera.

Si una escena se carga de forma acumulativa y tiene la misma referencia de datos de oclusión que la escena actual, los datos de oclusión se utilizan para inicializar los renderizadores estáticos y los portales que seleccionan información para esa escena. Después de eso, el mecanismo de selección de oclusión se comporta como si todos los renderizadores y portales estáticos se hubieran integrado en una sola escena.

Modo de juego

Modo In-Play, cuando hay varias escenas en la Jerarquía, aparecerá una escena adicional llamada DontDestroyOnLoad.

Antes de Unity 5.3, cualquier objeto construido en Playmode y marcado como "DontDestroyOnLoad" permanecería en la jerarquía. Estos elementos no se consideran parte de ninguna escena, pero ahora se presentan como parte de la escena única DontDestroyOnLoad para que Unity los muestre y usted los estudie.

La escena DontDestroyOnLoad no es accesible para nosotros, y no está disponible durante el tiempo de ejecución.

Configuraciones específicas de la escena

Cada escenario tiene su propio conjunto de configuraciones. Son los siguientes:

Configuración de Navmesh Ajustes de escena en las secciones RenderSettings y LightmapSettings de la ventana Occlusion Culling (ambas ubicadas en la ventana de iluminación)

Cada escena administra su configuración y solo los parámetros conectados con esa escena se guardan en el archivo de escena.

Si tenemos varias escenas abiertas, se utilizan los ajustes de renderizado y malla de navegación de la escena activa. Esto implica que si deseamos modificar la configuración de una escena, debemos abrir solo una escena y hacerlo o convertir la escena en cuestión en la escena activa y hacerlo.

Cuando cambiamos la escena actual en el editor o durante el tiempo de ejecución, se aplican todas las configuraciones de la nueva escena y reemplazan todas las configuraciones anteriores.

secuencias de comandos

- **Creación de secuencias de comandos por un editor:** proporcionamos una estructura de escena, una API EditorSceneManager y una clase de utilidad SceneSetup para la creación de secuencias de comandos del editor.

Se puede acceder a la estructura de la escena tanto en el editor como en el tiempo de ejecución, e incluye algunos valores de solo lectura relacionados con la escena en sí, como su nombre y la ruta del recurso.

La clase EditorSceneManager solo se puede encontrar en el editor. Se deriva de SceneManager y contiene varios métodos para usar secuencias de comandos del editor a fin de lograr todas las funciones de edición de varias escenas mencionadas anteriormente.

La clase SceneSetup es una clase de utilidad simple que almacena información sobre la escena actual en la jerarquía.

Las clases Undo y PrefabUtility ahora admiten varias escenas. Ahora podemos usar [PrefabUtility.
InstanciatePrefab] para instanciar un prefabricado en una escena específica, y podemos usar (Undo.MoveGameObjectToScene)
[ScriptRef:Undo.MoveGameObjectToScene]] para mover objetos a la raíz de una escena de forma imposible.

- **Scripting en tiempo de ejecución:** la clase SceneManager tiene métodos para trabajar con numerosas escenas en tiempos de ejecución, como LoadScene y UnloadScene.

- **Recuerde:** la opción Guardar escena como en el menú Archivo solo guardará la escena actualmente activa. Guardar escena guardará todas las escenas actualizadas y nos ofrecerá nombrar la escena sin título, si existe.

- **Trucos y consejos:** Mantener presionada la tecla Alt mientras arrastra nos permite agregar una escena a la jerarquía manteniéndola vacía. Esto nos permite cargar el escenario más tarde si es necesario.

La opción Crear en la ventana del proyecto se puede usar para crear nuevas escenas. La configuración predeterminada de Game Objects se utilizará en nuevos escenarios.

Podemos utilizar EditorSceneManager para evitar configurar nuestra jerarquía cada vez que reiniciamos Unity o facilitar el mantenimiento de varias configuraciones.

Utilice GetSceneManagerSetup para recuperar una lista de objetos SceneSetup que describen la configuración actual. Luego podemos serializarlos en un ScriptableObject o algo más, junto con cualquier información adicional sobre la configuración de nuestra escena que deseemos guardar.

Para adquirir la lista de escenas cargadas en tiempo de ejecución, use scene Cuente y recorra las escenas usando GetSceneAt.

GameObject.scene devuelve la escena a la que pertenece un GameObject y (SceneManager.MoveGame ObjectToScene).) mueve un GameObject a la raíz de un
escena.

Es mejor evitar el uso de DontDestroyOnLoad para persistir en la administración de GameObjects que deseamos mantener entre cargas de escena. En su lugar, use SceneManager para construir una escena de administrador con todos sus administradores. SceneManager y LoadScene(ruta>, LoadSceneMode.Additive). Para controlar el progreso de nuestro juego, usa UnloadScene.

Plantillas de escena

Unity replica plantillas de escenas para generar nuevas escenas. Considere una plantilla de escena como una escena preconfigurada con todas las cosas con las que deseamos comenzar. La plantilla básica, por ejemplo, generalmente incluye una cámara y una luz.

Podemos adaptar los tipos de escenas nuevas creadas en un proyecto creando nuestras plantillas de escena. Cree plantillas para cada nivel de un juego con fines ilustrativos, de modo que todos los que trabajen en el proyecto puedan comenzar sus escenas con los materiales y escenarios apropiados.

Se puede crear una plantilla a partir de cualquier escena de Unity. Después de crear una plantilla, podemos usarla para generar un número ilimitado de nuevos escenarios.

Creación de plantillas de escenas

Podemos hacer una nueva plantilla de escena de una de tres maneras:

- Comience con una plantilla vacía.
- Cree una plantilla a partir de un recurso de escena preexistente.
- Haga una plantilla a partir del escenario actual.

Después de crear una plantilla, podemos modificar sus propiedades o usarla para construir nuevos escenarios.

Creación de una plantilla de escena en blanco Podemos crear plantillas de escena en blanco y luego personalizarlas. Una plantilla vacía no se muestra en el cuadro de diálogo Nueva escena a menos que se editen sus atributos para conectarla con una
activo de la escena.

Para crear una plantilla de escena vacía en la carpeta del proyecto actual, haga lo siguiente:

- Seleccione Activos > Crear > Plantilla de escena en la
  menú.

Para crear una plantilla de escena vacía en una carpeta de proyecto específica, haga lo siguiente:

Elige una de las siguientes opciones:

- Haga clic derecho en la carpeta en la ventana del proyecto para abrir el menú contextual.
- Haga clic con el botón derecho en el panel de recursos para acceder al menú contextual y luego abra la carpeta en la ventana Proyecto.
- Elija Crear > Plantilla de escena en el menú.

Creación de una plantilla a partir de un activo existente en una escena Cualquier escena actual se puede convertir en una plantilla de escena. Después de crear una plantilla a partir de una escena existente, es posible que deseemos actualizar sus atributos para designar cuál de sus dependencias clona Unity al crear una nueva escena a partir de ella.

Para crear una plantilla a partir de un recurso de escena existente, acceda a la ventana Proyecto y elija una de las siguientes opciones:

- Para abrir el menú contextual, haga clic con el botón derecho en un recurso de escena. Luego vaya a Plantillas de escena > Crear plantilla de escena a partir de escena.
- Seleccione el activo de la escena, luego elija Activos > Crear > Plantilla de escena de Escena del menú principal.

Crear una plantilla a partir de la escena actual Seleccionar archivo > Guardar como plantilla de escena en el menú para generar una plantilla de escena a partir de la escena actual.

Si tenemos alguna modificación sin guardar, Unity le indicará nosotros para guardar la escena antes de guardar la plantilla.

Después de crear una plantilla a partir de la escena actual, es posible que deseemos actualizar sus atributos para designar sus dependencias en los clones de Unity al crear una nueva escena.

• **Creación de plantillas de escenas a partir de secuencias de comandos de C#:** las plantillas de escenas se pueden crear a partir de secuencias de comandos de C#.

Use la función CreateSceneTemplate para generar una plantilla de escena vacía.

```
SceneTemplate.CreateSceneTemplate(string
sceneTemplatePath)
```

Utilice la función CreateTemplateFromScene para construir una plantilla a partir de una escena existente. Unity conecta la escena con la plantilla y extrae las dependencias de la escena automáticamente.

```
EscenaTemplate.CreateTemplateFromScene(
SceneAsset sourceSceneAsset, string
sceneTemplatePath);
```

Modificación de plantillas de escena

Para editar una plantilla de escena, ábrala en una ventana Inspector después de seleccionarla en la ventana Proyecto.

La plantilla de escena del Inspector tiene las siguientes secciones:

- **Detalles:** especifica la escena que utilizará la plantilla y la descripción de la plantilla que se mostrará en el cuadro de diálogo Nueva escena.
- **Miniatura:** Esto nos permite crear una imagen de vista previa para la plantilla.

- **Tubería de plantilla de escena:** define un script personalizado opcional que se ejecutará cuando Unity cree una nueva escena a partir de la plantilla.
- **Dependencias:** utilizan el elemento dependenciesdependen cies** en sus manifiestos para especificar la colección de paquetes que necesitan. Estas se consideran dependencias directas para proyectos, pero dependencias indirectas o transitivas para paquetes.

Detalles La sección Detalles define qué escena usar como plantilla y cómo aparece la plantilla en el cuadro de diálogo Nueva escena.

Propiedad	**Descripción**
Escena de plantilla	Especifique qué escena se utilizará como plantilla. Esto podría ser de cualquier escenario en el Proyecto.
Título	El nombre de la plantilla. El nombre que proporcione aquí se mostrará en el cuadro de diálogo Nueva escena.
Descripción	La descripción de la plantilla. La explicación que ponemos aquí se mostrará en el cuadro de diálogo Nueva escena.
Anclar en el cuadro de diálogo Nueva escena	Esta configuración determina si esta plantilla se fija o no en el cuadro de diálogo Nueva escena. Las plantillas ancladas siempre se muestran en la parte superior de la lista Plantillas de escena en la lista Proyecto.

Miniatura La sección Miniatura tiene opciones para producir una imagen de vista previa de plantilla. En el cuadro de diálogo Nueva escena, se muestra la imagen de vista previa.

Propiedad	Descripción
Textura	Esta propiedad especifica un recurso de textura que se utilizará como miniatura para esta plantilla. En el Proyecto, podemos utilizar cualquier activo de Textura. Si no proporcionamos una textura, la plantilla utilizará el icono de activo de la plantilla de escena predeterminada.
[Vista previa en miniatura]	Si la plantilla contiene una textura en miniatura, se mostrará.
Instantánea	Esta plantilla tiene opciones para capturar una imagen en miniatura.
Vista	Especifica si se debe capturar la vista de la cámara principal o la vista del juego.
Tomar instantáneas	Para guardar la vista seleccionada, haga clic en este botón.

Canalización para plantillas de escena Para agregar un guión de canalización de plantilla de escena a esta plantilla, utilice esta configuración.

Un modelo de escena Cuando creamos una nueva escena a partir de una plantilla, el script de canalización nos permite ejecutar código personalizado. Consulte Personalización de la creación de escenas nuevas para obtener más información.

Dependencias Esta sección incluye todas las dependencias de la escena de la plantilla. Cuando construimos una nueva escena usando la plantilla, podemos elegir si clonar o no cada dependencia.

Para clonar o hacer referencia a una dependencia, gire el botón Clonar opción activada o desactivada para cada dependencia de la lista.

Cuando construimos una nueva escena a partir de una plantilla, Unity examina la escena de la plantilla para ver si contiene dependencias clonables. Si es así, Unity genera una carpeta con el mismo nombre que la nueva escena y almacena en ella las dependencias copiadas.

Personalización de la creación de nuevas escenas

Cree un script de canalización de plantilla de escena y vincúlelo a la plantilla para ejecutar código personalizado cada vez que Unity cree una nueva escena a partir de una plantilla. Unity produce una nueva instancia del script de canalización cada vez que crea una nueva escena a partir de la plantilla.

Para vincular el script a una plantilla, haga lo siguiente:

- Edite las propiedades de la plantilla inspeccionándola.
- Establezca el atributo Tubería de plantilla de escena en nuestra Escena Ruta del script de canalización de plantilla.

También podemos vincular el script a la plantilla usando C# usando la función SceneTemplateAsset.templatePipeline.

Una secuencia de comandos de canalización de plantilla de escena debe basarse en [ISceneTemplatePipeline] o Interfaz [SceneTemplatePipelineAdapter]. Debería implementar los eventos a los que deseamos reaccionar, como BeforeTemplateInstantiation o AfterTemplateInstantiation en el código a continuación.

Ejemplo:

```
utilizando UnityEditor.SceneTemplate;
utilizando UnityEngine;
usando UnityEngine.SceneManagement;
clase pública
DummySceneTemplatePipeline1 :
ISceneTemplatePipeline
{
```

```
        vacío público BeforeTemplateInstantiat
ion(SceneTemplateAsset sceneTemplateAsset, bool isAdditive,
string sceneName)
        {
                si (elemento de plantilla de escena)
                {
                        Debug.Log($"Antes del canal de plantilla
{sceneTemplateAsset.name} isAdditive: {isAdditive}
sceneName: {sceneName}");

                }
        }

        vacío público AfterTemplateInstantiati
on(SceneTemplateAsset escenaTemplateAsset,
Escena escena, bool isAdditive, string sceneName)

        {
                si (elemento de plantilla de escena)
                {
                        Debug.Log($"Después de la plantilla
Pipeline {sceneTemplateAsset.name} escena: {escena}
isAdditive: {isAdditive} sceneName: {sceneName}");

                }
        }
}
```

La secuencia de creación de instancias de plantilla de escena

Unity ejecuta varias acciones de archivo cuando crea una nueva escena a partir de una plantilla con dependencias clonables. La mayoría de estas acciones generan eventos de Unity, que puede escuchar y responder en secuencias de comandos.

La siguiente es la secuencia de instanciación:

1. En el cuadro de diálogo Nueva escena, seleccione Crear. La unidad se refiere a lo siguiente:

   • La plantilla de escena es una ventaja.

   • **Plantilla de Escena:** Esta es la Escena de Unity que corresponde a la plantilla.

   • **Una nueva escena:** Esta es una nueva instancia de la plantilla de escena.

   Unity activa ISceneTemplatePipeline. El evento BeforeTemplateInstantiation del recurso de plantilla vincula el recurso a un script ISceneTemplatePipeline que activa

2. Unity activa la SceneTemplate.

3. El evento NewTemplateInstantiating.

4. Unity genera una nueva escena que es un duplicado de la escena de la plantilla.

5. Unity genera una carpeta con el mismo nombre que la nueva escena y transfiere todas las dependencias clonables en ello.

6. Unity carga la nueva escena en la memoria y activa los siguientes eventos:

   • EditorSceneManager.sceneOpening

   • MonoBehavior.OnValidate

   • EditorSceneManager.sceneOpened

7. Unity reasigna todas las referencias de activos clonables, por lo que nueva escena se refiere a los clones.

8. Unity guarda la nueva escena y provoca lo siguiente acciones a ocurrir:

   • EditorSceneManager.sceneSaving

   • EditorSceneManager.sceneSaved

9. Unity activa ISceneTemplatePipeline.After TemplateInstantiation para el activo de plantilla y vincula el activo a un script ISceneTemplatePipeline que activa.

10. Unity activa SceneTemplate.NewTemplate Evento instanciado.

Configuración de la plantilla de escena

Abra la ventana Opciones de proyecto (menú: Editar > Configuración del proyecto) y seleccione Plantilla de escena en la lista de categorías para ver la configuración del proyecto de la plantilla de escena.

Nueva configuración de escena

La configuración de Nueva escena regula lo que ocurre cuando crea una nueva escena desde el menú Archivo (Archivo > Nueva escena) o usando el atajo de teclado Ctrl/Cmd + n.

Opción:	**Descripción:**
Nuevo menú Escena	
Cuadro de diálogo Nueva escena	Se muestra el cuadro de diálogo Nueva escena.
Escena integrada	Sin abrir el cuadro de diálogo Nueva escena, esta El comando crea una nueva escena. La nueva escena es un clon de la plantilla básica del proyecto.

Configuración de tipos por defecto

Las opciones de Tipos predeterminados determinan si Unity clona automáticamente tipos particulares de activos al crear una nueva escena a partir de una plantilla de escena.

Habilite la opción Clonar para ese tipo de activo en la lista para que Unity lo clone de manera predeterminada.

Deshabilite la opción Clonar para ese tipo de activo en la lista para haga que Unity haga referencia a ese tipo de activo de forma predeterminada.

La opción Clonar para todas las demás categorías, ya sea habilitada o deshabilitada, establece el comportamiento predeterminado de clonación/ referencia para las categorías de activos que no aparecen en la lista.

Haga clic en el botón Eliminar para eliminar un tipo de activo de la lista.

Para agregar un tipo de activo a la lista, realice una de las siguientes acciones: En el campo Agregar tipo, ingrese un tipo de activo específico. Haga clic en el botón Examinar para iniciar una ventana de búsqueda para buscar y seleccionar un tipo de activo en particular.

Luego, para agregar el tipo de activo a la lista, haga clic en el botón Agregar.

Haga clic en el botón Restablecer valores predeterminados para volver a la lista y la configuración de tipos de activos predeterminados de Unity.

¿QUÉ SON LOS GAMEOBJECTS?

La noción más significativa del Editor de Unity es el GameObject.

Cada objeto de tu juego, desde personas y coleccionables hasta luces, cámaras y efectos especiales, es un GameObject.
Un GameObject, por otro lado, no puede hacer nada por sí mismo; hay que darle características antes de que pueda convertirse en un personaje, un entorno o un efecto especial.

Los objetos principales de Unity son GameObjects, que representan personajes, accesorios y escenarios. No realizan nada por sí mismos, pero sirven como contenedores para los Componentes, que implementan la funcionalidad.

Los componentes se utilizan para proporcionar a un GameObject los atributos que necesita para convertirse en una luz, un árbol o un cámara.

Podemos agregar diferentes combinaciones de componentes a un GameObject dependiendo del tipo de elemento que queramos construir.

Unity incluye una plétora de tipos de componentes incorporados, e incluso podemos crear los nuestros propios usando Unity Scripting API.

Un objeto Light, por ejemplo, se genera conectando un componente Light a un GameObject.

Un objeto de cubo sólido contiene un elemento Mesh Filter y Mesh Renderer para representar la superficie del cubo y un componente Box Collider para expresar el volumen sólido del objeto en términos físicos.

Especificaciones

Un componente Transform (que representa la posición y la orientación) está constantemente asociado con un GameObject y no se puede eliminar. Los componentes adicionales que proporcionan la funcionalidad del objeto se pueden agregar usando el menú Componente del editor o un script. También hay numerosos objetos útiles preconstruidos (formas básicas, cámaras, etc.) a los que se puede acceder desde el menú GameObject > 3D Object; consulte Objetos primitivos para obtener más información.

Debido a que los GameObjects son un elemento tan vital de Unity, hay una gran cantidad de manuales de contenido que los detallan con gran detalle. Puede encontrar más información sobre cómo utilizar GameObjects en Unity en las siguientes secciones.

- Transforma.
- Introducción a los componentes.
- Objetos primitivos y marcadores de posición.
- Uso de Componentes.
- Desactivación de GameObjects.
- Etiquetas.
- Creación de componentes con scripting.
- GameObjects estáticos.
- Guardar su trabajo.

Transforma

Transform se usa para mantener la posición, la rotación, la escala y el estado de crianza de un GameObject y, por lo tanto, es muy importante. Un componente Transform siempre está asociado con un GameObject; es imposible borrar o construir un GameObject sin uno.

El componente de la transformación
Cada elemento en la escena. Cada GameObject tiene una Transformación que gobierna su Posición, Rotación y Escala.

Propiedades

Propiedad:	**Función:**
Posición	La posición de Transform en coordenadas X, Y y Z.
Rotación	Rotación de la Transformada en grados alrededor de los ejes X, Y y Z.
Escala	La escala de Transform a lo largo de los ejes X, Y y Z. El tamaño original está representado por el valor "1". (El tamaño al que se importó el objeto).

Los valores de posición, rotación y escala de un Transform se miden en relación con el padre del Transform. Si no hay padre para la transformación, los atributos se miden en el espacio mundial.

Edición de transformación

Las transformaciones se pueden controlar en un espacio tridimensional (3D) a lo largo de los ejes X, Y y Z o en un espacio bidimensional (2D) a lo largo de los ejes X e Y. Estos ejes están representados en Unity por los colores rojo, verde y azul, respectivamente.

Las transformaciones se pueden modificar en la Vista de escena o cambiando sus atributos en el Inspector. Las transformaciones en la escena se pueden modificar usando las herramientas Mover, Rotar y Escalar. Estas herramientas se pueden encontrar en la esquina superior izquierda del Editor de Unity.

Las herramientas son aplicables a cualquier elemento de la escena. Cuando hacemos clic en un objeto, la herramienta gizmo aparecerá dentro de él. El aspecto del gadget está determinado por la herramienta elegida.

Cuando hacemos clic y arrastramos sobre uno de los tres ejes del gizmo, su color será amarillo. El elemento se trasladará, rotará o escalará a lo largo del eje especificado a medida que movemos el mouse. Cuando soltamos el botón del ratón, el eje queda seleccionado.

En el modo de traducción, también existe la posibilidad de bloquear el movimiento en un plano particular (es decir, permitir arrastrar dos ejes mientras se mantiene el tercero sin cambios). Los tres pequeños cuadrados de colores en el centro del gizmo de traducción activan el bloqueo de cada plano; los colores corresponden al eje, que se bloqueará cuando se haga clic en el cuadrado (por ejemplo, el azul bloquea el eje Z).

Crianza de los hijos

Cuando se utiliza Unity, una de las cosas más importantes que se deben comprender es la crianza de los hijos. Cuando un GameObject es el padre de otro GameObject, el GameObject secundario se mueve, rota y escala de la misma manera que su padre. La crianza de los hijos puede compararse con la interacción entre nuestros brazos y nuestro cuerpo; cada vez que nuestro cuerpo se mueve, nuestros brazos también se mueven. Los elementos secundarios pueden tener sus elementos secundarios, y así sucesivamente. Entonces, nuestras manos pueden verse como “hijos” de nuestros brazos, cada mano con numerosos dedos, y así sucesivamente. Puede haber varios descendientes para cada elemento, pero solo un padre.
Estas varias capas de interacciones padre-hijo forman una jerarquía de transformación.

La raíz es el objeto en la parte superior de una jerarquía (por ejemplo, el único elemento de la jerarquía que no tiene un padre).

Arrastre cualquier GameObject en la Vista de jerarquía a otro para crear un Padre. Los dos GameObjects formarán una conexión padre-hijo como resultado de esto.

Vale la pena señalar que los valores de Transformación en el Inspector para cualquier GameObject secundario se muestran en relación con los valores de Transformación del Padre. Estos valores se conocen como coordenadas locales. Volviendo a la analogía del cuerpo y los brazos, la posición de nuestro cuerpo puede cambiar mientras

caminamos, pero nuestros brazos permanecerán unidos en la misma posición relativa. Trabajar con coordenadas locales para elementos secundarios suele ser adecuado para el desarrollo de la escena. Aun así, los juegos suelen ser útiles para descubrir su posición real en el espacio mundial o las coordenadas globales. La API de secuencias de comandos del componente Transform contiene distintas propiedades para la posición, la rotación y la escala locales y globales, y la capacidad de transformar cualquier punto entre las coordenadas locales y globales.

Limitaciones de escalado no uniforme

Cuando la escala en una transformación tiene valores distintos para x, y y z, esto se conoce como escalado no uniforme (2, 4, 2).
Por otro lado, la escala Uniforme tiene el mismo valor para x, y y z, por ejemplo (3, 3, 3). El escalado no uniforme puede ser ventajoso en algunas instancias específicas, aunque introduce algunas anomalías que el escalado uniforme no presenta:

- Algunos componentes no admiten completamente el escalado no uniforme. Algunas funciones, como Sphere Collider, Capsule Collider, Light y Audio Source, tienen un elemento circular o esférico especificado por un atributo de radio. En tales circunstancias, la forma circular no se volverá elíptica debido a una escala no uniforme, sino que permanecerá redonda.

- Cuando un elemento secundario se rota en relación con un artículo que no es uni padre formalmente escalado, puede parecer sesgado o "cortado". Algunos componentes aceptan una escala básica no uniforme, pero no funcionan correctamente cuando se inclinan de esta manera. Un Box Collider sesgado, por ejemplo, no coincidirá correctamente con la forma de la malla mostrada.

- La escala de un objeto secundario de un elemento principal con una escala no uniforme no se actualizará automáticamente a medida que gira por motivos de rendimiento. Como resultado, cuando la escala finalmente se actualiza, por ejemplo, si el objeto secundario se desconecta del principal, la forma del elemento secundario puede parecer que cambia abruptamente.

Importancia de la escala

La escala de transformación define la diferencia de tamaño entre una malla en su aplicación de modelado y una malla en Unity.

El tamaño de la malla en Unity (y, por lo tanto, la escala de Transform) es crítico, especialmente durante el modelado de física. El motor de física piensa que una unidad en el espacio mundial equivale a un metro por defecto. Si un artículo es particularmente enorme, puede parecer que cae en “cámara lenta”; la simulación es precisa ya que vemos una gran cosa caer desde una gran distancia.

El tamaño de nuestro artículo puede verse afectado por tres factores:

1. En su aplicación de modelado 3D, el tamaño de nuestro malla.
2. El parámetro Factor de escala de malla en la Configuración de importación del elemento.
3. Los valores de escala de nuestro componente de transformación.

Idealmente, no deberíamos cambiar la Escala de su objeto en el Componente de Transformación. El método ideal es desarrollar sus modelos a una escala de la vida real para que no tengamos que ajustar la escala de nuestro Transform. El siguiente mejor enfoque es cambiar la escala a la que se importa su malla en la Configuración de importación de nuestra malla individual. Optimizaciones específicas

ocurren en función del tamaño de importación, y la creación de instancias de un objeto con un valor de escala modificado puede reducir la velocidad.

Trabajar con transformaciones: algunos indicadores

- Es útil establecer la posición de los padres en 0,0,0> antes de agregar al niño al criar Transforms. Esto implica que las coordenadas locales del niño serán las mismas que las coordenadas globales, lo que facilita asegurarse de que el niño esté en el lugar correcto.
- Si vamos a utilizar Rigidbodies para el modelado físico, asegúrese de conocer la propiedad Scale en la página de referencia del componente Rigidbody.
- Los colores de los ejes de transformación (y otros componentes de la interfaz de usuario) se pueden cambiar en la configuración (Menú: Unity > Preferencias, luego seleccione el panel Colores y teclas).
- La ubicación de los morfos secundarios se ve afectada al cambiar la Escala. Escalar el padre a (0,0,0), por ejemplo, colocará a todos los hijos en (0,0,0) en relación con el padre.

Se introducen los componentes

Un GameObject es un objeto de Unity Editor que contiene componentes. Los componentes describen cómo se comporta un GameObject.

Esta sección explica cómo ver e interactuar con componentes en Unity y una descripción general rápida de las configuraciones de componentes más típicas.

Para ver los componentes de un GameObject, selecciónelo en las ventanas Escena o Jerarquía, luego vaya a la ventana Inspector para obtener una lista de sus componentes y sus configuraciones.

Se puede interactuar con los componentes directamente en el Editor o mediante un script. Para obtener información sobre cómo controlar e interactuar con los componentes mediante una secuencia de comandos, consulte la Guía de secuencias de comandos. sección.

Configuraciones de componentes comunes

Esta sección describe algunas de las configuraciones básicas de componentes predeterminados de Unity.

Transformación de componentes

En Unity, cada GameObject tiene un componente Transform. Este componente especifica la ubicación, la rotación y la escala del GameObject en el mundo del juego y la vista de escena. Este componente no se puede eliminar.

El componente Transform también admite la idea de la paternidad, lo que nos permite convertir un GameObject en un elemento secundario de otro GameObject y controlar su posición mediante el componente Transform del elemento principal. Este es un componente fundamental para trabajar con GameObjects en Unity.

Componentes de la cámara principal GameObject

Cada nueva escena comienza con un GameObject llamado Cámara principal de forma predeterminada. Este GameObject está configurado para ser la cámara principal de nuestro juego. Incluye el componente Transform, el componente Camera y un Audio Listener para capturar audio en nuestra aplicación.

Hacer uso de componentes

Los componentes son los elementos básicos de los objetos y acciones de un juego. Son los componentes esenciales de cada GameObject. Antes de continuar, lea GameObjects

página si aún no comprende el vínculo entre Componentes y GameObjects.

Un GameObject es un contenedor para una variedad de componentes. De forma predeterminada, todos los GameObjects incluyen un componente de transformación. Esto se debe a que Transform determina dónde se coloca el GameObject y cómo se gira y escala. Si GameObject no tuviera un componente de transformación, no tendría una posición en el mundo. Como ejemplo, intente hacer un GameObject vacío ahora. Seleccione GameObject->Crear vacío en el menú. Examine el Inspector después de seleccionar el nuevo GameObject.

Siempre podemos usar el Inspector para examinar qué Componentes están asociados con el GameObject especificado.
El Inspector siempre le mostrará qué componentes están actualmente conectados cuando se agregan y eliminan. El Inspector se utilizará para modificar todas las características de cualquier Componente.

Adición de componentes

El menú Componentes nos permite agregar Componentes al GameObject especificado. Lo intentaremos ahora adjuntando un Rigidbody al GameObject vacío que acabamos de crear. Selecciónelo y luego vaya al Componente->
Menú Física->Cuerpo rígido. Cuando hagamos esto, las características de Rigidbody se mostrarán en el Inspector.
Podemos recibir una sorpresa si presiona Reproducir mientras aún se elige el GameObject vacío. Pruébelo y observe cómo Rigidbody ha proporcionado la funcionalidad de GameObject, que de otro modo estaría vacía. (La posición Y de la transformación de GameObject comienza a disminuir. Esto se debe a que el motor de física de Unity obliga a GameObject a caer bajo la gravedad).

El Explorador de componentes, al que se puede acceder a través del botón Agregar componente en el inspector del objeto, es otra alternativa.

El navegador nos permite recorrer rápidamente los componentes por categoría y también contiene una barra de búsqueda que podemos usar para encontrar componentes por nombre.

Un solo GameObject puede tener cualquier número o combinación de Componentes adjuntos. Algunos componentes funcionan mejor cuando se combinan con otros. El Rigidbody, por ejemplo, puede usarse con cualquier Colisionador. El motor de física NVIDIA PhysX controla el Rigidbody, y el Collider permite que el Rigidbody colisione e interactúe con otros Colliders.

Si deseamos obtener más información sobre el uso de un Componente específico, podemos hacerlo visitando la página de Referencia del Componente correspondiente.

También podemos ver la página de referencia de un Componente de Unity haciendo clic en el pequeño? en el encabezado del Componente en el Inspector.

Edición de componentes

La versatilidad de los componentes es una de sus mejores características. Cuando conectamos un Componente a un GameObject, el Componente tiene varios valores o Propiedades que se pueden cambiar en el editor al desarrollar un juego o mediante scripts mientras se ejecuta el juego. Las propiedades se clasifican en dos tipos: Valores y Referencias.

Es un GameObject en blanco con un componente de fuente de audio adjunto. Todos los ajustes de la fuente de audio en el Inspector son los predeterminados.

Hay una sola propiedad de Referencia y siete propiedades de Valor en este Componente. La propiedad de referencia es

un clip de audio. Cuando esta fuente de audio comience a reproducirse, reproducirá el archivo de audio especificado en el atributo Clip de audio. Se producirá un error si no se crea ninguna referencia, ya que no hay audio para reproducir. Dentro del Inspector, debe hacer referencia al archivo. Es tan simple como arrastrar un archivo de audio desde la Vista del proyecto a la Propiedad de referencia o usar el Selector de objetos para hacer esto.

En los Componentes se pueden incluir referencias a cualquier otro tipo de Componente, GameObject o Activo. Puede encontrar más información sobre la asignación de referencias en la página Propiedades de edición.

Las características restantes del clip de audio son todas propiedades de valor. Estos se pueden cambiar directamente en el Inspector. Los atributos de valor del clip de audio son todos conmutadores, valores numéricos y campos desplegables, pero también pueden ser cadenas de texto, colores, curvas y otros tipos. Puede encontrar más información sobre esto, así como sobre cómo modificar las propiedades de los valores, en el artículo sobre la edición de las propiedades de los valores.

Comandos del menú contextual del componente

El menú contextual de un componente contiene varios comandos esenciales.

También se puede acceder a las instrucciones exactas en el inspector a través del ícono del menú kebab (tres puntos verticales) en el extremo superior derecho del panel del componente.

- **Restablecer:** este comando devuelve las propiedades del componente a su configuración anterior antes de la sesión de edición más reciente.
- **Eliminar:** si ya no necesitamos el componente asociado con el GameObject, podemos eliminarlo con

el comando Quitar componente. Vale la pena señalar que algunas combinaciones de componentes dependen unas de otras (por ejemplo, una articulación de bisagra solo funciona si también se adjunta un Rigidbody); la eliminación de componentes en los que otros confían dará como resultado un mensaje de advertencia.

- **Mover arriba/abajo:** para modificar el orden de los componentes de un GameObject en el Inspector, utilice los comandos Mover arriba y Mover abajo.

- **Copiar/Pegar:** el comando Copiar componente copia el tipo de componente y la configuración de propiedades actual. Con Pegar valores de componentes, estos pueden luego copiarse en otro elemento del mismo tipo. También podemos usar Pegar componente como nuevo para crear un nuevo componente con los datos copiados en un objeto.

Experimentación de propiedades

Mientras nuestro juego está en modo de juego, podemos alterar los atributos de cualquier GameObject en el Inspector. Por ejemplo, podríamos desear experimentar con diferentes alturas de salto. Si agregamos un atributo de altura de salto a un script, podemos probarlo ingresando al modo de reproducción, cambiando el valor y presionando el botón de salto para ver qué ocurre. Luego, sin salir del Modo Juego, podemos hacer otra modificación y ver las consecuencias en segundos. Cuando salimos del Modo de reproducción, nuestras propiedades se restaurarán a su configuración previa al Modo de reproducción, asegurando que no se pierda ningún esfuerzo. Este método brinda una flexibilidad impresionante para explorar, modificar y perfeccionar su juego sin comprometerse con largos ciclos de iteración.

Objetos que son primitivos o marcadores de posición

Unity puede trabajar con modelos 3D de cualquier forma desarrollados con herramientas de modelado. Sin embargo, varios tipos de objetos elementales, como el Cubo, la Esfera, la Cápsula, el Cilindro, el Plano y el Cuádruple, se pueden producir directamente dentro de Unity. Estos objetos suelen ser útiles por sí mismos (por ejemplo, un plano suele utilizarse como una superficie de terreno nivelado), pero también proporcionan un método rápido para generar marcadores de posición y prototipos por motivos de prueba. Usando el elemento relevante del menú GameObject > 3D Object, se pueden agregar primitivos a la escena.

Cubo

Este es un cubo básico con lados de una unidad de largo que tienen una textura tal que la imagen se repite en cada una de las seis caras.
Un cubo no es un objeto particularmente frecuente en la mayoría de los juegos, pero cuando se escala, puede ser útil para paredes, postes, cajas, escalones y otros objetos similares. También es un objeto de marcador de posición útil para que los programadores lo utilicen durante el desarrollo cuando un modelo completo no está disponible. La carrocería de un automóvil, por ejemplo, se puede recrear toscamente usando una caja extendida de casi el tamaño adecuado. Aunque esto no es apropiado para todo el juego, funciona bien como un pequeño objeto simbólico para probar el código de control del automóvil.

Debido a que las aristas de un cubo tienen una unidad de largo, podemos examinar las proporciones de una malla importada a la escena colocando un cubo cerca y comparando los tamaños.

Esfera

Esta es una esfera de un diámetro de unidad (0,5 de radio de unidad) que ha sido texturizada de tal manera que la imagen completa gira alrededor de una vez,

con la parte superior e inferior "pellizcada" en los polos. Las esferas son claramente útiles para representar bolas, planetas y misiles, pero también se puede usar una esfera semitransparente para mostrar el radio de un efecto en una interfaz gráfica de usuario (GUI).

Cápsula

Una cápsula es un cilindro con dos cubiertas hemisféricas en cada extremo. El artículo tiene un diámetro de una unidad y una altura de dos unidades. Tiene una textura tal que la imagen gira exactamente una vez, apretada en el vértice de cada hemisferio.

Si bien no hay muchas cosas del mundo real con esta forma, la cápsula es un práctico marcador de posición de prototipo. Para actividades específicas, la mecánica de un elemento redondeado es a veces superior a la de una caja.

Cilindro

Este es un cilindro primario de dos unidades de alto y una unidad de diámetro que ha sido texturizado de tal manera que la imagen envuelve una vez alrededor de la forma del tubo del cuerpo pero también aparece de forma independiente en los dos extremos circulares planos. Los cilindros ayudan a hacer postes, varillas y ruedas, pero tenga en cuenta que la forma del colisionador es una cápsula (no existe un colisionador de cilindro primitivo en Unity). Si necesita un colisionador cilíndrico exacto para fines físicos, debe generar una malla de la forma adecuada en una aplicación de modelado y adjuntar un colisionador de malla.

Plano

Este es un cuadrado plano con bordes de diez unidades de largo alineados en el plano XZ del espacio de coordenadas locales. Tiene una textura tal que la imagen completa solo se muestra una vez dentro del cuadrado.

La mayoría de las superficies planas, como pisos y paredes, se benefician de

el uso de un avión. También se requiere una superficie para mostrar imágenes o videos en GUI y efectos especiales. Aunque se puede utilizar un avión para esto, la primitiva cuádruple más simple generalmente es una mejor opción para el trabajo.

Patio

La primitiva cuádruple es similar al plano, excepto que sus bordes tienen solo una unidad de largo y la superficie está orientada en el plano XY del espacio de coordenadas local. Además, un cuadrilátero se compone de sólo dos triángulos, mientras que un plano tiene doscientos. Un quad es apropiado cuando un objeto de escena solo se utiliza como pantalla de visualización para una imagen o video. Los quads se pueden usar para crear GUI simples y pantallas de información y partículas, sprites y gráficos "impostores" que representan objetos sólidos cuando se ven desde la distancia.

GameObjects 2D primitivos

Unity incluye GameObjects primitivos en 2D para permitirnos crear prototipos rápidamente de nuestro proyecto sin importar nuestros activos. Para hacer primitivo 2D, vaya a GameObject > Objeto 2D > Sprites y elige una de las siguientes opciones:

- Cuadrado.
- Circulo.
- Nueve Rebanadas.
- Diamante Isométrico.
- Cápsula.
- Punta Hexagonal.
- Hexágono Flat-Top.

Sprite y píxeles por unidad por defecto

El tamaño predeterminado de Sprite para la mayoría de las primitivas 2D es de 256 × 256 píxeles, con un tamaño de píxeles por unidad (PPU) de 256, lo que hace que su tamaño sea equivalente a una unidad en la escena. La primitiva Capsule, que tiene 256 × 512 píxeles (unidades 1:2), y la primitiva Isometric Diamond, que tiene 256 × 128 píxeles, son las excepciones (unidades 1:0,5).

Cuadrado

La primitiva Cuadrado 2D es un cuadrado blanco con una dimensión de 1 × 1 unidades de Unidad. Podemos usarlo para construir plataformas rápidamente o como marcador de posición para otros elementos, como barreras como cajas. Podemos interactuar con otros GameObjects y física 2D agregando el componente Box Collider 2D al GameObject. Seleccione la opción Nine-Sliced en su lugar para obtener un Sprite más escalable que cambia de tamaño dinámicamente.

Circulo

La primitiva Circle 2D es un círculo blanco con un diámetro de una unidad Unity. Se puede usar como marcador de posición para varios objetos en nuestra escena, como obstáculos o accesorios como camionetas o potenciadores. Podemos usar Circle Collider 2D para interactuar con otros objetos y física 2D al agregarlo a GameObject.

Cápsula

La primitiva Capsule 2D es una cápsula blanca con un tamaño de 1 × 2 unidades. Esta Cápsula se puede utilizar como marcador de posición para muchos aspectos de nuestra escena, como un obstáculo, un objeto o un personaje suplente. Podemos interactuar con otros objetos y física 2D agregando un Capsule Collider 2D al GameObject.

Diamante isométrico

El primitivo Isometric Diamond 2D es un Sprite blanco con forma de diamante de 1 × 0,5 unidades. Este Sprite está destinado a actuar como un marcador de posición para Isometric Tilemaps. Para optimizar el mosaico, los píxeles en la parte superior e inferior de este Sprite se han cortado significativamente.

Hexágono de parte superior plana

La primitiva 2D Hexagon Flat-Top es un hexágono regular con lados en la parte superior e inferior de 1 unidad de ancho. Está diseñado para usarse como un marcador de posición de Sprite básico para mosaicos en mapas de mosaicos hexagonales de parte superior plana. Al optimizar el mosaico, los píxeles a la izquierda y a la derecha de este Sprite se han cortado significativamente.

Hexágono de punta

La primitiva 2D Hexagon Point-Top es un hexágono regular de una unidad de altura con puntos en la parte superior e inferior. Está destinado a ser utilizado como un marcador de posición de Sprite básico para mosaicos en mapas de mosaicos hexagonales con punta en la parte superior. Para optimizar el mosaico, los píxeles en la parte superior e inferior de este Sprite se han cortado significativamente.

nueve rebanadas

El primitivo 2D Nine-Sliced es un cuadrado blanco de 1 × 1 unidad con esquinas redondeadas. Este Sprite tiene nueve cortes, con límites de 64 píxeles en cada lado. Está diseñado para usarse con los modos de dibujo Sliced y Tiled de Sprite Renderer. El Sprite de nueve cortes se puede usar como un marcador de posición versátil para numerosos elementos en nuestra escena y proyecto.

Para hacer que el Sprite interactúe con otros objetos y física 2D, agregue un Box Collider 2D con Auto Tiling habilitado.

La secuencia de comandos se utiliza para crear componentes

La secuencia de comandos (o la creación de secuencias de comandos) es el proceso de agregar nuestras propias modificaciones de código a las capacidades del Editor de Unity utilizando la API de secuencias de comandos de Unity.

Cuando construimos una secuencia de comandos y la conectamos a un GameObject, la secuencia de comandos se muestra en el Inspector de GameObject de la misma manera que lo hace un componente integrado. Esto se debe a que cuando guardamos un script en su proyecto, se convierte en un componente.

Técnicamente, cada script que escribimos se compila como un tipo de componente; por lo tanto, el editor de Unity trata nuestro script como si fuera un componente integrado. El Inspector expone los elementos del script que definimos, y el Editor realiza cualquier funcionalidad que hayamos creado.

Desactivación de GameObjects

Para eliminar un GameObject de la Escena momentáneamente, etiquételo como inactivo. Para hacerlo, vaya al Inspector y desmarque la casilla de verificación junto al nombre del GameObject o use el método SetActive en el script. Compruebe el atributo Self activo en el script para ver si un objeto está activo o

inactivo.

Desactivar un GameObject principal

Cuando desactiva un GameObject principal, todos sus GameObjects secundarios también se desactivan.

La desactivación anula la configuración de activeSelf en todos los GameObjects secundarios, lo que hace que toda la jerarquía esté inactiva desde el elemento principal hacia abajo. Porque esto no modifica la

valor de la propiedad activeSelf en los GameObjects secundarios, vuelven a su estado anterior cuando se reactiva el principal.

Esto implica que acceder a la propiedad activeSelf de un GameObject secundario no le indicará si está o no actualmente activo en la Escena porque incluso si está configurado como activo, uno de sus padres puede estar configurado como inactivo.

En cambio, si necesitamos saber si ahora está activo en la escena, use la propiedad activeInHierarchy, que considera la influencia principal de sus padres.

Etiquetas

Una etiqueta es un término que se puede asignar a uno o más GameObjects. Podríamos, por ejemplo, crear Etiquetas de "Jugador" para personajes controlados por jugadores y Etiquetas de "Enemigo" para personajes no controlados por jugadores. Se puede usar una etiqueta "Coleccionable" para identificar cosas que el jugador puede reunir en un Escena.

Las etiquetas ayudan en la identificación de GameObjects por razones de secuencias de comandos. Eliminan la necesidad de agregar GameObjects manualmente a los atributos públicos de un script mediante arrastrar y soltar, lo que ahorra tiempo al utilizar el mismo código de script en varios GameObjects.

Las etiquetas son esenciales en los scripts de control de Collider para determinar si el jugador está interactuando con un oponente, un accesorio o un objeto coleccionable, por ejemplo.

Podemos encontrar un GameObject instruyendo al método GameObject.FindWithTag() para buscar cualquier objeto que contenga la etiqueta deseada. GameObject se usa en el siguiente ejemplo: FindWithTag(). Crea el

respawnPrefab en la posición de GameObjects con el
Etiqueta "Reaparecer":

```
utilizando UnityEngine;
utilizando System.Collections;
Ejemplo de clase pública: MonoBehaviour {

        reaparición pública de GameObjectPrefab;
        reaparición pública de GameObject;
        vacío Inicio ()
{
                si (reaparecer = = nulo)
                        reaparecer = GameObject.
FindWithTag("Reaparecer");
                Instanciar (respawnPrefab, respawn.
transformar.posición, reaparecer.transformar.
rotación) como GameObject;
        }
}
```

Creación de nuevas etiquetas

El Inspector muestra las opciones desplegables Etiqueta y Capa directamente debajo del nombre de cualquier GameObject.

Para agregar una nueva etiqueta, haga clic en Agregar etiqueta.... Esto abre el Administrador de etiquetas y capas del Inspector. Es crucial tener en cuenta que una vez que se ha nombrado una etiqueta, no se puede cambiar el nombre.

Las capas, como las etiquetas, se usan para especificar cómo Unity debe representar GameObjects en la escena.

Usar una etiqueta

El Inspector muestra las opciones desplegables Etiqueta y Capa directamente debajo del nombre de cualquier GameObject. Para aplicar una etiqueta existente a un GameObject, ingrese las etiquetas

menú y seleccione la etiqueta deseada. Esta etiqueta ahora está conectada con el GameObject.

GameObjects que permanecen estáticos

Un GameObject estático no se mueve durante la ejecución. Un GameObject dinámico se mueve durante el transcurso de un juego.

En Unity, varios sistemas pueden precalcular información sobre GameObjects estáticos en el Editor. Debido a que los GameObjects no se mueven, los resultados de estos cálculos siguen siendo válidos durante el tiempo de ejecución. Esto implica que Unity puede ahorrar dinero en los cálculos de tiempo de ejecución y, al mismo tiempo, mejorar potencialmente el rendimiento.

Las banderas del editor estático de propiedades

La propiedad Static Editor Flags identifica los sistemas Unity que pueden usar un GameObject estático en sus cálculos previos. Seleccione qué sistemas deben incluir el GameObject en sus cálculos previos utilizando el menú desplegable. Estos sistemas no tienen ningún impacto cuando las banderas del editor estático se configuran en tiempo de ejecución.

Solo deberíamos incluir un GameObject en los cálculos previos si el sistema necesita saberlo. Incluir un GameObject en precálculos para un sistema que no necesita saber acerca de ese GameObject puede generar cálculos derrochadores, archivos de datos innecesariamente esenciales o un comportamiento inesperado.

El atributo Static Editor Flags se encuentra en el Inspector de un GameObject, en la esquina superior derecha. Consiste en una casilla de verificación que establece el valor en Todo o Nada y un menú desplegable que nos permite seleccionar qué valores incluir.

Estos son los siguientes valores disponibles:

Propiedad:	**Función:**
Nada	No incluya GameObject en los cálculos previos de ningún sistema.
Todo	Incluya GameObject en los cálculos previos para todos los sistemas enumerados a continuación.
Contribuir IG	Mientras habilitamos este parámetro, Unity tiene en cuenta el Mesh Renderer de destino al calcular la iluminación global. Estos cálculos se realizan durante el período de horneado cuando se calculan previamente los datos de iluminación. La propiedad ContributeGI expone la propiedad ReceiveGI. La función ContributeGI no tiene impacto a menos que activemos una opción de iluminación global para la escena de destino, como Iluminación global horneada o Global en tiempo real.
	Iluminación. Esta bandera se describe en detalle en una publicación del blog de Unity sobre la iluminación estática con sondas de luz.
oclusor estático	En el sistema de selección de oclusión, marque el GameObject como un oclusor estático.
ocluido estático	En el mecanismo de selección de oclusión, marque GameObject como Static Occludee.
Dosificación Estática	Combine la malla de GameObject con otras mallas adecuadas para minimizar potencialmente los costos de renderizado en tiempo de ejecución.
Navegación Estática	Cuando precompile los datos de navegación, incluya GameObject.
Enlace fuera de malla Generación	Cuando calcule previamente los datos de navegación, intente construir un enlace fuera de malla que comience con este GameObject.
Sonda de reflexión	Incluya este GameObject en los datos precalculados para Reflection Probes con el atributo Type establecido en Baked.

Manteniendo nuestro trabajo seguro

La mayoría de los datos guardados en Unity se dividen en modificaciones de escena y actualizaciones de todo el proyecto.

- Vaya a Archivo > Guardar para guardar todos los cambios de Escena y Proyecto (o Guardar como). Este es el método más rápido para guardar todo a la vez.

- Vaya a Archivo > Guardar proyecto para guardar los cambios en todo el proyecto pero no modificaciones de escena.

Tenga en cuenta que hay una excepción a esta regla al editar en modo prefabricado. Archivo > Guardar solo guarda las modificaciones en el Prefab actualmente abierto en esta situación. No almacena cambios en Escenas o Proyectos.

Mientras trabajamos en el Editor, Unity automáticamente almacena cierta información.

La escena cambia

Las alteraciones de la escena implican cambios en GameObjects dentro de la escena, como cuando uno mismo:

- Se puede agregar, mover o eliminar un GameObject.

- En el Inspector, modifica los parámetros de un GameObject.

Modificaciones de todo el proyecto

Las modificaciones de todo el proyecto en Unity afectan a todo el proyecto en lugar de a una sola escena. Vaya a Archivo > Guardar proyecto para guardar la configuración de todo el proyecto sin almacenar las modificaciones de la escena.

Esto puede ser beneficioso si, por ejemplo, queremos construir una escena temporal para probar algunos cambios.

Entre las modificaciones de todo el proyecto se encuentran:

Cuando guardamos un proyecto, Unity guarda cualquier cambio en la configuración del proyecto en la carpeta Biblioteca. Los ajustes se guardan en los siguientes archivos:

Entrada: InputManager.activo
Reproductor: ProjectSettings.asset
Etiquetas y capas: TagManager.asset
Hora: TimeManager.asset
Física: DynamicsManager.asset
Gráficos: GraphicsSettings.asset
Física 2D: Physics2DSettings.asset
Calidad: QualitySettings.asset
Editor: EditorUserSettings.asset
Red: NetworkManager.asset
Audio: AudioManager.activo

- **Configuración de compilación:** Unity almacena las modificaciones a la configuración de compilación como EditorBuildSettings.asset en la carpeta Biblioteca.

- **Activos modificados/cambiados:** Unity guarda los activos modificados no guardados durante un guardado que guarda la configuración de todo el proyecto. Esto se refiere principalmente a los tipos de activos que carecen de un botón Aplicar en su Inspector para guardarlos al instante.

- **Activos sucios:** Unity también guarda cualquier activo designado como sucio (lo que significa que algo lo ha tocado o modificado). Editores y scripts personalizados

se puede utilizar para indicar un activo como sucio de una de las siguientes maneras:

- Usar la clase SerializedObject en conjunto con propiedades serializadas.
- Para guardar los cambios, utilice la clase Deshacer.
- Si ninguno de los otros métodos funciona, podemos probar SetDirty.

Ahorro inmediato

Cuando ocurren ciertas modificaciones, Unity las guarda rápidamente en el disco. Estos son algunos ejemplos:

- **Nuevos activos:** Unity guarda los nuevos activos a medida que se crean, pero no las modificaciones futuras a esos activos.
- **Configuración de importación de activos: la configuración** de importación de la mayoría de los tipos de activos necesita que presionemos el botón "Aplicar" para que los cambios surtan efecto. Cuando hacemos clic en Aplicar, Unity guarda nuestras modificaciones.
- **Datos Horneados:** Algunos tipos de datos están "horneados" en nuestro Proyecto. Cuando se completa cada horneado, Unity almacena automáticamente estos datos. Esto incluye lo siguiente:
  - Datos de Iluminación Horneada.
  - Información de navegación al horno.
  - Datos para el sacrificio de oclusión horneada.
- **Cambios en el orden de ejecución de scripts:** Unity almacena automáticamente esta información en cada script. meta archivo cuando presionamos el botón Aplicar.

PREFABRICADOS

El sistema Prefab en Unity nos permite crear, configurar y guardar un GameObject como un activo reutilizable, repleto de todos sus componentes, valores de propiedad y GameObjects secundarios. El activo prefabricado sirve como plantilla para crear nuevas instancias prefabricadas en la escena.

Convertir un juego de GameObject de cierta manera, como un personaje que no es jugador (NPC), un accesorio o una pieza de escenografía. to a Prefab nos permite reutilizarlo en numerosos lugares de nuestra Escena o en varias Escenas de nuestro Proyecto. Esto es preferible a copiar y pegar el GameObject ya que el sistema Prefab automáticamente mantiene sincronizadas todas las copias.

Cualquier cambio que hagamos en un activo prefabricado se refleja instantáneamente en todas las instancias de ese prefabricado, lo que nos permite realizar cambios amplios en todo nuestro proyecto sin hacer cambios idénticos en cada copia del activo.

Los Prefabs se pueden anidar dentro de otros Prefabs para construir jerarquías de objetos complicadas que son fáciles de cambiar en numerosos niveles.

Sin embargo, esto no implica que todas las instancias de Prefab deban ser similares. Si queremos que ciertas instancias prefabricadas sean diferentes de otras, podemos anular la configuración en instancias prefabricadas específicas. También podemos construir variaciones de Prefab, que nos permiten organizar una colección de anulaciones en una versión significativa de un Prefab.

Los prefabricados son particularmente útiles cuando deseamos crear GameObjects durante el tiempo de ejecución que no existían en nuestra escena al principio, como hacer que aparezcan potenciadores, efectos especiales, proyectiles o NPC en los puntos de tiempo apropiados durante el juego.

Algunas aplicaciones típicas de la prefabricación incluyen:

- Activos ambientales, como una variedad particular de árbol que se utiliza varias veces alrededor de un nivel.
- NPC: por ejemplo, un tipo específico de robot puede aparecer varias veces en nuestro juego en varios niveles.
  Pueden diferir en su ritmo de viaje o en el sonido que emiten (a través de anulaciones).
- Los proyectiles, como el cañón de un pirata, pueden crear un prefabricado de bala de cañón cada vez que se dispara.
- El personaje principal del jugador, el jugador prefabricado,
  podría colocarse al comienzo de cada nivel (escena separada) de nuestro juego.

Creación de prefabricados

Los recursos prefabricados sirven como plantillas en el sistema prefabricado de Unity. Los recursos prefabricados se crean en el Editor y se almacenan como un recurso en la ventana Proyecto. Los activos prefabricados se pueden usar para crear una cantidad ilimitada de instancias prefabricadas. Las instancias prefabricadas se pueden producir en el editor y almacenar como parte de las escenas, o se pueden crear instancias durante el tiempo de ejecución.

Hacer activos prefabricados

Para crear un activo prefabricado, arrastre un GameObject desde la ventana de jerarquía a la ventana de proyecto. El GameObject, junto con todos sus componentes y GameObjects secundarios, se agrega a nuestra ventana de Proyecto como un nuevo Activo. Los recursos de Prefabs se representan en la ventana del Proyecto mediante una representación en miniatura del GameObject o el icono de Prefab de cubo azul, según cómo esté configurada nuestra ventana del Proyecto.

El GameObject original también se convierte en una instancia de Prefab durante el proceso de creación de Prefab Asset.
Ahora es un elemento secundario del recién formado Prefab Asset.
Las instancias de Prefab se muestran en texto azul en la Jerarquía, y el GameObject raíz del Prefab se indica con el icono de Prefab de cubo azul en lugar de los iconos de GameObject rojo, verde y azul.

Creación de instancias prefabricadas

Podemos construir instancias del Prefab Asset en el Editor arrastrándolo desde la vista de Proyecto a la vista de Jerarquía.

Las secuencias de comandos también se pueden usar para construir objetos Prefab en tiempo de ejecución.

Reemplazo de prefabricados existentes

Podemos cambiar un activo de Prefab en la ventana de Proyecto arrastrando un nuevo GameObject desde la ventana de Jerarquía y colocándolo encima de un objeto de Prefab existente.

Cuando reemplazamos un Prefab existente, Unity se esfuerza por mantener intactas las referencias al Prefab y sus componentes, como GameObjects y Componentes secundarios. Esto se logra haciendo coincidir los nombres de GameObjects entre el nuevo Prefab y el Prefab actual que se está reemplazando.

Debido a que esta coincidencia se realiza solo por nombre, es difícil predecir qué GameObject coincidirá si la jerarquía de Prefab contiene varios GameObjects con el mismo nombre.

Como resultado, si queremos guardar nuestras referencias mientras guardamos sobre un Prefab existente, debemos dar a cada GameObject en el Prefab un nombre único.

Además, si un solo GameObject dentro del Prefab tiene más de uno del mismo tipo de Componente conectado,

podemos tener una dificultad similar al guardar sobre un prefabricado existente para conservar las referencias a los componentes existentes. Es imposible prever cuál de ellos coincidirá con las referencias actuales en este escenario.

Edición prefabricada en modo prefabricado

Abra un activo prefabricado en modo prefabricado para modificarlo. El modo prefabricado nos permite examinar y modificar el contenido del activo prefabricado independientemente de cualquier otro GameObjects en nuestra escena. Los cambios realizados en el modo prefabricado tienen efecto en todas las instancias de ese prefabricado.

Entrada y salida del modo prefabricado

Un activo prefabricado se puede editar de forma aislada o en contexto.

- Cuando editamos un Prefab de forma aislada, Unity oculta el resto de nuestra Escena de trabajo actual y solo nos muestra los GameObjects que están relacionados con el Prefab.
- Cuando modificamos un Prefab en contexto, el resto de nuestra Escena de trabajo actual permanece visible pero no editable.

Edición de aislamiento

En el modo Prefab, podemos comenzar a editar un Prefab de varias maneras. Podemos abrir un activo prefabricado y cambiarlo por separado de las siguientes maneras:

- En la ventana Proyecto, haga doble clic en el activo prefabricado.
- En la ventana Proyecto, elija un Activo prefabricado y luego haga clic en el botón Abrir prefabricado en la ventana Inspector.

Cuando ingresamos al modo prefabricado por sí mismo, Unity muestra solo el contenido de ese prefabricado en la vista de escena y

el panel Jerarquía. La raíz de Prefab es un GameObject estándar; carece del indicador azul de instancia de Prefab.

La vista de escena en el modo prefabricado tiene una barra de navegación en la parte superior. El prefabricado que ahora está abierto es el de la derecha. Para volver a las Escenas principales u otros Activos prefabricados que podamos haber abierto, use la barra de navegación.

En la parte superior de la ventana Jerarquía, hay una barra de encabezado de Prefab que muestra el Prefab actualmente activo. Podemos retroceder un paso haciendo clic en la flecha hacia atrás en la barra de encabezado, de forma idéntica a hacer clic en la ruta de navegación en la barra de ruta de navegación en la vista Escena.

Edición contextual

También podemos abrir un activo prefabricado en contexto mediante el uso de una instancia de ese prefabricado. Hay varias formas de lograr esto, incluyendo:

En la ventana Inspector, elija una instancia de Prefab de el panel Jerarquía y haga clic en el botón Abrir.

En el panel Jerarquía, elija una instancia de Prefab y presione P en el teclado. Este es el enlace de teclado estándar.

En el panel Jerarquía, haga clic en el botón de flecha junto a la instancia de Prefab.

Unity muestra la representación visual del contexto en escala de grises de manera predeterminada para separarlo visualmente del contenido de Prefab que cambia. Sin embargo, podemos utilizar el control Context: de la barra Prefab para cambiarlo a cualquiera de los siguientes estados:

- **Normal:** muestra el contexto en sus colores predeterminados.
- Escala de **grises:** muestra el contexto en escala de grises.
- **Oculto:** Oculta el contexto completamente, revelando solo el contenido de Prefab.

Los GameObjects que forman parte del contexto no se pueden seleccionar, ni aparecen en la Jerarquía. Esto nos permite centrarnos en desarrollar nuestro Prefab sin elegir involuntariamente GameObjects no relacionados o tener un panel de Jerarquía desordenado. Cuando movemos GameObjects que forman parte del contenido de Prefab, podemos utilizar la funcionalidad de ajuste de Unity para ajustar a GameObjects en el contexto, siempre que el contexto no esté configurado como Oculto.

En el modo Prefab en contexto, Unity muestra el contenido de Prefab en el mismo lugar que la instancia de Prefab desde la que se accedió. Esto implica que podemos ver una vista previa de la transformación base de los contenidos de Prefab con diferentes valores de posición y rotación que los que posee el Prefab Asset.

Estos ajustes no se pueden editar en el modo prefabricado de Context. Si necesitamos hacer cambios, podemos abrir Prefab de forma aislada o seleccionar Prefab Asset en la ventana Project y hacer cambios en el Inspector.

Además de los atributos de Transformación raíz, también podemos anular características adicionales de una instancia de Prefab, lo que puede alterar radicalmente su aspecto en comparación con el Activo de Prefab del que es una instancia. Para ver estos valores anulados de la instancia de Prefab, active la casilla de verificación Mostrar anulaciones en la barra de Prefab cuando esté en Modo Prefab en Contexto.

Todas las propiedades que reemplazamos en la instancia de Prefab se previsualizan de la misma manera en los contextos de Prefab mientras esta opción está activa y no podemos modificarlas. Deshabilite la opción Mostrar anulaciones una vez más para cambiar esas propiedades.

Guardar automáticamente

En la esquina superior derecha de la vista de escena, se está configurando un guardado automático para el modo prefabricado. Cuando lo activamos, Unity almacena

cualquier cambio que hagamos en un Prefab al Activo Prefab. De forma predeterminada, el guardado automático está habilitado.

Deshabilite la opción Guardar automáticamente si queremos realizar cambios sin almacenarlos inmediatamente en el Activo preestablecido. Cuando salimos del modo prefabricado para el prefabricado actual, Unity pregunta si guardaremos las modificaciones no guardadas. Desactivar Auto Save puede ayudar si la edición de un Prefab en el modo Prefab parece lenta.

Cambiar del modo de aislamiento al modo de contexto

Cuando lanzamos el modo prefabricado desde un recurso prefabricado, Unity aísla el contenido del prefabricado. Cuando activamos el Modo Prefabricado a través de una instancia de Prefab en la ventana de Jerarquía, se lanza el Modo Prefabricado en Contexto.

Cuando iniciamos el modo prefabricado de esta manera, el contexto de la instancia prefabricada es visible en la vista de escena, aunque no esté cambiando la instancia sino el activo prefabricado en sí. Por ejemplo, si activamos el modo Prefab en contexto a través de una instancia de Prefab en una escena, podemos ver los alrededores de la escena mientras editamos el Prefab. Las condiciones de iluminación en el Prefab también son las mismas que en la Escena.

Si deseamos acceder a una instancia de Prefab de forma aislada en lugar de contextual, mantenga presionada la tecla Alt y haga clic en el botón Activar o en el botón de flecha para abrir el Modo prefabricado. También podemos crear un acceso directo personalizado en el cuadro Accesos directos usando el comando Escenario > Editar prefabricado en aislamiento.

Deshacer

Cuando realizamos cambios en un activo prefabricado mientras está en modo prefabricado, solo puede deshacer esos cambios mientras aún está en modo prefabricado. Cuando salimos del modo prefabricado para un

Activo Prefabricado específico, nuestras modificaciones para ese Activo Prefabricado ya no son visibles en el historial de deshacer.

Entorno para editar

En aislamiento, podemos utilizar una escena como entorno de edición para el modo prefabricado. Esto nos permite cambiar su Prefab contra un fondo personalizado en lugar de una Escena vacía. Esto podría ser beneficioso si queremos examinar cómo se ve nuestro prefabricado en un contexto específico de nuestra elección. Cuando ingresa al modo prefabricado en aislamiento, Unity solo utiliza este entorno de edición.

En el Modo Prefabricado, no puedes elegir los GameObjects en la Escena que hemos asignado como entorno de edición, ni aparecen en la Jerarquía. Esto nos permite centrarnos en desarrollar nuestro Prefab sin elegir involuntariamente GameObjects no relacionados o tener un panel de Jerarquía desordenado.

Abra la ventana Editor (menú superior: Editar > Configuración del proyecto, luego elija la categoría Editor) y navegue hasta la sección Entorno de edición prefabricado para designar una escena como entorno de edición.

Para prefabricados "sin interfaz de usuario", use la configuración de entorno normal, mientras que para prefabricados de interfaz de usuario, use la configuración de entorno de interfaz de usuario. Los prefabricados de interfaz de usuario tienen un componente Rect Transform en la raíz en lugar de un componente Transform estándar. Los prefabricados con un componente de transformación regular se consideran "sin interfaz de usuario".

Anulaciones para instancias

Las anulaciones de instancias nos permiten definir las diferencias entre las instancias de Prefab sin dejar de conectarlas al mismo activo de Prefab.

Cuando hacemos modificaciones a un activo prefabricado, se reflejan en todas sus instancias. Sin embargo, también podemos hacer

cambios en una instancia específica. Esto establece una anulación de instancia en esa instancia en particular.

Por ejemplo, supongamos que tuviéramos un activo prefabricado llamado "Robot" que usamos en muchos niveles de nuestro juego. Cada instancia de "Robot", por otro lado, tiene un valor de velocidad diferente y un clip de audio asignado.

Las anulaciones de instancias se clasifican en cuatro tipos:

1. Anular el valor de una propiedad.
2. Incluyendo un nuevo componente.
3. Sacar un componente.
4. Creando un nuevo GameObject hijo.

Las instancias de Prefab tienen varias limitaciones: no puede volver a vincular un GameObject que sea parte de un Prefab, y no puede eliminar un GameObject que sea parte de un Prefab. Sin embargo, podemos desactivar un GameObject, que es una opción aceptable para desinstalar un GameObject (esto cuenta como una anulación de propiedad).

En la ventana Inspector, las anulaciones de instancias se resaltan con una etiqueta de nombre en negrita y una línea azul en el margen izquierdo. La línea azul en el margen cubre todo el componente cuando agregamos un nuevo elemento a una instancia de Prefab.

Los componentes agregados y eliminados tienen insignias de más y menos en sus íconos de Inspector, mientras que los GameObjects agregados tienen una insignia de más en su ícono de Jerarquía.

Las anulaciones tienen prioridad

El valor de propiedad anulado de una instancia de Prefab siempre tiene prioridad sobre el valor del activo de Prefab. Esto implica

que cambiar la propiedad de un activo prefabricado no afecta las instancias cuando se anula esa propiedad.

Ya sea que hagamos una modificación a un activo prefabricado y no actualice todas las instancias según lo previsto, debemos verificar si esa propiedad en la instancia se anula. Es aconsejable utilizar anulaciones de instancias solo cuando sea necesario.

Si nuestro Proyecto tiene una cantidad significativa de anulaciones de instancias, podría ser un desafío saber si nuestras modificaciones al Activo prefabricado afectarán o no a todos.
instancias.

La alineación es exclusiva de la instancia prefabricada

La alineación de una instancia de Prefab es una circunstancia específica que se trata de manera diferente a otros atributos. Los valores de alineación nunca se envían desde el activo prefabricado a las instancias prefabricadas. Esto implica que siempre pueden desviarse de la alineación del activo prefabricado sin requerir una anulación de instancia explícita. En particular, la alineación se refiere a las propiedades Posición y Rotación de la Transformación raíz de la instancia de Prefab, y para una Transformación Rect, esto también incluye los valores de Ancho, Alto, Márgenes, Anclas y Pivote.

Esto se debe a la rareza de requerir numerosas copias de un Prefab para adoptar la misma posición y rotación. La mayoría de las veces, querrá que nuestras instancias prefabricadas estén en varias ubicaciones y rotaciones, por lo que Unity no las considera anulaciones prefabricadas.

Cambiar las ocurrencias de un prefabricado

El Inspector para la raíz de una instancia de Prefab contiene tres controles adicionales que un GameObject estándar: Abrir, Seleccionar y Anular.

El botón Abrir en el Modo prefabricado abre el Activo prefabricado del que se deriva la instancia, lo que nos permite editar el Activo prefabricado y, por lo tanto, cambiar todas sus instancias. El botón Seleccionar en la ventana Proyecto selecciona el Activo prefabricado a partir del cual se crea esta instancia. El botón Anulaciones activa el menú desplegable Anulaciones.

Anulaciones desplegables

El panel desplegable Personalizaciones muestra todas las anulaciones de la instancia de Prefab. También nos permite agregar anulaciones de instancias al activo prefabricado o revertir las anulaciones de instancias a los valores del activo prefabricado. El botón desplegable Anulaciones solo se muestra para la instancia raíz de Prefab, no para Prefabs dentro de Prefabs.

El cuadro desplegable Anulaciones nos permite aplicar o revertir anulaciones prefabricadas individuales o aplicar o revertir todas las anulaciones prefabricadas a la vez.

- El activo prefabricado se modifica cuando se aplica una anulación.

  Esto agrega la anulación (que actualmente solo está disponible en nuestra instancia de Prefab) al Activo.
  Esto indica que el activo de Prefab ahora tiene esa modificación y la instancia de Prefab ya no la contiene como anulación.

- La instancia de Prefab se modifica cuando se revierte una anulación. Esto descarta efectivamente nuestra anulación y devuelve el Prefab Asset a su estado original.

El panel desplegable muestra una lista de modificaciones realizadas en la instancia, como componentes cambiados, agregados y eliminados y nuevos GameObjects.

Para ver una entrada, haga clic en ella. Esto abre una ventana flotante que muestra la modificación y le permite revertirla o aplicarla.

La vista compara los valores del componente en el activo de Prefab con el componente modificado en la instancia de Prefab para los componentes con valores modificados. Esto nos permite comparar los valores originales de Prefab Asset con las anulaciones actuales y determinar si revertir o aplicar esos valores.

El GameObject secundario "GermOBlaster" del ejemplo existe tanto en el activo prefabricado como en la instancia prefabricada, pero su escala se ha aumentado en la instancia.
Este aumento en la escala es una anulación de instancia y se puede ver en el panel desplegable Anulaciones como una comparación en paralelo.

Las opciones Revertir todo y Aplicar todo ahora están disponibles en el menú desplegable Anulaciones para revertir o aplicar todas las modificaciones a la vez. Si tenemos prefabricados dentro de prefabricados, el botón Aplicar todo siempre se aplica al prefabricado más externo, que tiene el botón desplegable Anulaciones en su GameObject raíz.

Cuando elegimos varios elementos al mismo tiempo, los botones Revertir todo y Aplicar todo se reemplazan por los botones Revertir seleccionado y Aplicar seleccionado. Estos se pueden usar para revertir o aplicar numerosas anulaciones al mismo tiempo. El botón Aplicar seleccionado, al igual que el botón Aplicar todo, siempre se aplica al prefabricado más externo.

Menús en contexto

En lugar de utilizar la ventana desplegable Anulaciones, puede retroceder y aplicar anulaciones específicas utilizando el menú contextual del Inspector.

Los atributos anulados se muestran en negrita. Se pueden deshacer o aplicar a través de un menú contextual:

Los componentes modificados se pueden revertir o aplicar usando el contexto o el botón desplegable de la rueda dentada del encabezado del componente menú:

Los componentes que se han agregado tienen una insignia más que aparece sobre el ícono. Se pueden revertir o aplicar mediante el botón desplegable de la rueda dentada del encabezado del componente.

Los componentes que se han eliminado tienen un signo menos que aparece sobre el icono. La eliminación se puede revertir o aplicar mediante el menú contextual o el botón desplegable de la rueda dentada del encabezado del componente. Revertir la eliminación restaura el componente, mientras que realizar la eliminación lo elimina del activo prefabricado.

Cuando un GameObject (incluidos otros Prefabs) se agrega como elemento secundario a una instancia de Prefab, aparece una insignia más sobre el ícono en la Jerarquía. Se pueden invertir o aplicar a través del menú contextual en el objeto Jerarquía.

Prefabricados que están anidados

Las instancias de Prefab se pueden incluir dentro de otros Prefabs. Esto se conoce como prefabricados anidados. Los prefabricados anidados mantienen vínculos con sus respectivos activos prefabricados y, al mismo tiempo, son un componente de otro activo prefabricado.

En modo prefabricado, agregue un prefabricado anidado

En el Modo Prefabricado, podemos agregar y operar con instancias Prefabricadas de la misma manera que lo haríamos en Escenas. Podemos mover un activo prefabricado de la ventana Proyecto a la

Ventana de jerarquía o vista de escena arrastrándola desde la ventana de Proyecto para crear una instancia de Prefab a partir de ese Activo dentro del Prefab en el que está trabajando.

Tenga en cuenta que el GameObject raíz, el icono de prefabricado de cubo azul no se muestra para el prefabricado que está abierto en modo prefabricado, pero se muestra para cualquier instancia de otros prefabricados.

Al igual que con las instancias de Prefab en escenas, también podemos aplicar anulaciones a estas instancias de Prefab.

Los prefabricados se pueden anidar usando sus instancias

También podemos agregar una instancia de Prefab como elemento secundario a otra instancia de Prefab en la escena sin ingresar al modo Prefab, como cualquier otro GameObject. En la Jerarquía, una instancia de Prefab adicional de este tipo tiene una insignia más superpuesta en el ícono, lo que indica que es una anulación de esa instancia específica de la Prefab externa.

El Prefab adicional se puede revertir o aplicar al Prefab externo de la misma manera que otras anulaciones (a través de la ventana desplegable Anulaciones o el menú contextual en GameObject en la Jerarquía) detallado en Edición de un Prefab a través de sus instancias. Solo el prefabricado exterior tiene el botón desplegable Anulaciones. Una vez que se aplica, el Prefab ya no muestra la insignia más, ya que ya no es una anulación, sino que está anidado dentro del Prefab Asset externo. Mantiene su símbolo de cubo azul porque es una instancia de Prefab por derecho propio, y mantiene su enlace a su Prefab Asset.

Prefabricados que están anidados

Las instancias de Prefab se pueden incluir dentro de otros Prefabs. Esto se conoce como prefabricados anidados. Los prefabricados anidados mantienen lazos con

sus respectivos Activos prefabricados y, al mismo tiempo, ser un componente de otro Activo prefabricado.

En modo prefabricado, agregue un prefabricado anidado

En el Modo Prefabricado, podemos agregar y operar con instancias Prefabricadas de la misma manera que lo haríamos en Escenas. Arrastre un activo prefabricado desde la ventana del proyecto a la ventana de jerarquía o vista de escena para crear una instancia prefabricada a partir de ese activo dentro del prefabricado abierto.

Tenga en cuenta que el ícono de Prefab de cubo azul no aparece en el GameObject raíz de un Prefab que está abierto en Modo Prefab, pero sí aparece en cualquier instancia de otros Prefabs.
Al igual que con las instancias de Prefab en escenas, también podemos aplicar anulaciones a estas instancias de Prefab.

Los prefabricados se pueden anidar usando sus instancias

También podemos agregar una instancia de Prefab como elemento secundario a otra instancia de Prefab en la escena sin ingresar al modo Prefab, como cualquier otro GameObject. En la Jerarquía, una instancia de Prefab adicional de este tipo tiene una insignia más superpuesta en el ícono, lo que indica que es una anulación de esa instancia específica de la Prefab externa.

El Prefab adicional se puede revertir o aplicar al Prefab externo de la misma manera que otras anulaciones (a través de la ventana desplegable Anulaciones o el menú contextual en GameObject en la Jerarquía) detallado en Edición de un Prefab a través de sus instancias. Solo el prefabricado exterior tiene el botón desplegable Anulaciones. Una vez que se aplica, el prefabricado ya no muestra la insignia de más, ya que ya no es una anulación, sino que está anidado dentro del propio activo prefabricado externo.

Mantiene su símbolo de cubo azul porque es una instancia de Prefab por derecho propio, y mantiene su enlace a su Prefab Asset.

Variaciones prefabricadas

Las variantes prefabricadas son útiles cuando necesitamos una colección de versiones predefinidas de un prefabricado.

Por ejemplo, podríamos desear incluir una variedad de GermSlimeTargets en nuestro juego, todos los cuales están construidos en el mismo núcleo GermSlimeTarget Prefab. Sin embargo, es posible que deseemos que algunos GermSlimeTargets transporten objetos, viajen a varias velocidades o produzcan efectos de sonido adicionales.

Para hacer esto, podríamos configurar nuestro primer GermSlimeTarget Prefabricado para ejecutar todas las actividades principales que todos GermSlimeTarget debe compartir y luego construir numerosas variantes prefabricadas para:

- Usar una anulación de propiedad en un script para ajustar el ritmo de un GermSlimeTarget.
- Adjunte un GameObject adicional a un GermSlimeTarget brazo para que sostenga un objeto.
- Agregue un componente AudioSource que reproduzca un sonido silenciador a GermSlimeTarget para darle un silenciador similar a una babosa.

Una variante prefabricada hereda los atributos de otra prefabricada, que se denomina base. Las anulaciones aplicadas a la variante Prefab tienen prioridad sobre la configuración de la Prefab básica. Una variante prefabricada se puede basar en cualquier otro prefabricado, incluidos modelos prefabricados y diferentes variantes prefabricadas.

Desarrollo de una variante prefabricada

Hay varios métodos para crear una variante prefabricada basada en otra prefabricada.

En la vista Proyecto, haga clic con el botón derecho en un prefabricado y seleccione Crear > Variante prefabricada. Esto genera una variación del Prefab especificado sin anulaciones al principio. Para comenzar a agregar anulaciones a la variante prefabricada, ábrala en modo prefabricado.

Además, podemos arrastrar una instancia de Prefab desde la Jerarquía a la ventana del Proyecto. Cuando hacemos esto, aparece un cuadro de diálogo que nos pregunta si queremos crear un nuevo prefabricado original o una variante de prefabricado.

Cuando seleccionamos Prefab Variant, se crea una nueva Prefab Variant basada en la instancia de Prefab que arrastró. Todas las personalizaciones que teníamos en esa instancia ahora se han incluido en la nueva variante prefabricada. Podemos abrirlo en modo prefabricado para agregar más anulaciones y cambiar o eliminar las existentes.

Variantes prefabricadas que tienen el ícono Prefab azul con flechas

Edición de variantes prefabricadas

Cuando abrimos una variante de Prefab en modo Prefab, la raíz se muestra como una instancia de Prefab con el icono azul de Prefab. Esta instancia de Prefab representa el Prefab básico del que deriva la Variante de Prefab, no la Variante de Prefab en sí. Cualquier cambio que hagamos en la variante prefabricada se anula en la base de la variante.

Debido a que la instancia de Prefab es una instancia del Prefab GermSlimeTarget base y el botón Seleccionar siempre selecciona el Prefab Asset a partir del cual se crea un ejemplo, si elegimos GermSlimeTarget con GermOBlaster root GameObject y luego hacemos clic en el botón Seleccionar en el Inspector, seleccionará la base GermSlimeTarget prefabricada en lugar de la variante GermSlimeTarget con GermOBlaster.

Las anulaciones de Prefab, como valores de propiedad actualizados, componentes nuevos, componentes eliminados y GameObjects secundarios adicionales, se pueden usar en una variante de Prefab como cualquier otra instancia de Prefab. También existen las mismas restricciones: no podemos volver a generar GameObjects en la variante Prefab derivada de su Prefab original. Tampoco podemos eliminar un GameObject de una variante prefabricada si todavía está en su prefabricado base. Sin embargo, podemos desactivar GameObjects (a través de una anulación de propiedad) para obtener el mismo resultado que desinstalar un GameObject.

Al actualizar una variante prefabricada en modo prefabricado, tenga en cuenta que la aplicación de estas anulaciones (a través de la ventana desplegable o los menús contextuales de anulaciones) dará como resultado que las variantes de nuestra variante se apliquen al activo prefabricado subyacente. Con frecuencia esto no es lo que deseamos. El propósito de una variante prefabricada es brindar un método práctico para almacenar una colección valiosa y reutilizable de anulaciones; por lo tanto, generalmente deben permanecer como anulaciones y no aplicarse al activo prefabricado subyacente. Para demostrarlo, si agregamos GermOBlaster GameObject al activo prefabricado básico (el "GermSlimeTarget"), el activo prefabricado también incluiría GermOBlaster.

Toda la idea de GermSlimeTarget con versión de GermOBlaster es que es la única con una GermOBlaster; por lo tanto, el GermOBlaster adicional GameObject debe dejarse como anulación dentro de Prefab Variante.

Cuando abrimos la ventana desplegable Anulaciones, siempre podemos ver en el encabezado para qué objeto son las anulaciones y en qué contexto residen. El encabezado de una variante prefabricada indicará que las anulaciones son para el

prefabricado básico y residir en la variante prefabricada. Sin duda, el botón Aplicar todo también indica Aplicar todo a la base.

Múltiples capas de anulación

Las anulaciones pueden existir en numerosos niveles cuando se trabaja con prefabricados dentro de otros prefabricados o con variantes de prefabricados, y las mismas anulaciones se pueden aplicar a varios prefabricados distintos.

Aplicar selección de destino

Cuando tenemos una instancia de Prefab con Prefab anidados o una Variante de Prefab, podemos elegir a qué Prefab se debe aplicar una anulación.

Considere un "jarrón" prefabricado que está anidado dentro de un prefabricado "Mesa", y la escena tiene una instancia prefabricada de "Mesa".

Si se anula una propiedad en "Jarrón" en este caso, la anulación se puede aplicar al "Jarrón" o a la "Mesa".

El botón Aplicar, Todo en el menú desplegable Anulaciones solo nos permite aplicar la anulación al Prefab externo, en este ejemplo, la "Tabla". Sin embargo, al aplicar usando el menú contextual o la vista comparativa para componentes individuales en la ventana desplegable Anulaciones, podemos seleccionar un destino de aplicación.

Si elegimos Aplicar a Prefabricado "Jarrón" en este ejemplo, el valor se aplica al Activo Prefabricado "Jarrón" y se utiliza para todas las instancias del Prefabricado "Jarrón".

Además, si seleccionamos Aplicar como anulación en la "Tabla" prefabricada, el valor se convierte en una anulación del ejemplo de "Jarrón" contenido dentro de la Prefabricada "Mesa". La propiedad de la instancia en la Escena ya no está etiquetada como una

invalidar, pero si abrimos el prefabricado "Mesa" en el modo prefabricado, la propiedad en el ejemplo del prefabricado "Jarrón" se marca como anulada.

Cuando se anula como una anulación en el activo prefabricado "Mesa", el activo prefabricado "Jarrón" no se ve afectado. Esto implica que todas las instancias de la casa prefabricada "Mesa" tienen el valor actualizado en su instancia de la casa prefabricada "Jarrón", pero otras instancias de la casa prefabricada "Jarrón" que no forman parte de la casa prefabricada "Mesa" no se ven afectadas.

Si la propiedad en el prefabricado "Jarrón" se actualiza más adelante, afectará a todas las instancias del prefabricado "Jarrón" a menos que se anule esa propiedad. Debido a que se anula en la instancia de "Jarrón" dentro del prefabricado de "Mesa", la modificación no afectará a ninguna de las instancias de "Jarrón" que forman parte de las instancias de "Mesa".

Aplicar a casas prefabricadas internas puede tener un impacto en el exterior prefabricados laterales también.

Cuando se aplican una o más propiedades a un elemento prefabricado interno, sus anulaciones se revierten en los elementos prefabricados externos, lo que ocasionalmente genera cambios en los elementos prefabricados externos.

En nuestro ejemplo, si se selecciona Aplicar a Prefabricado "Jarrón" y el Prefabricado "Mesa" contiene una anulación del valor, la anulación en el Prefabricado "Mesa" se revierte y la propiedad simultáneamente en la instancia conserva el valor que se acaba de aplicar. Si este no fuera el caso, el valor del ejemplo cambiaría inmediatamente después de su aplicación.

Desempaquetado de instancias prefabricadas

Desempaquetar una instancia de Prefab devuelve el contenido de la instancia de Prefab a un GameObject estándar. Esto es exactamente lo contrario de generar (empaquetar) un prefabricado, excepto

que no elimina el activo de Prefab y solo afecta la instancia de Prefab.

Desempaquetar una instancia de Prefab es tan simple como hacer clic derecho sobre ella en la Jerarquía y seleccionar Desempaquetar Prefab. El GameObject generado en la escena ya no está vinculado a su activo prefabricado anterior. Este procedimiento no afecta al Activo prefabricado en sí, y aún puede haber instancias de este en nuestro Proyecto.

Si anulamos su instancia de Prefab antes de desempaquetarla, se "hornearán" en los GameObjects resultantes. Es decir, los valores permanecerán sin cambios, pero ya no tendrán el estado de anulaciones ya que no hay Prefab para anular.

Cuando desempaquetamos un Prefab que contiene Prefabs anidados, los Prefabs anidados siguen siendo instancias de Prefab con conexiones a sus respectivos Activos de Prefab. De manera similar, cuando desempaquetamos una variante de Prefab, se crea una nueva instancia de Prefab en la raíz que es una instancia de Prefab básica.

En general, desempaquetar una instancia de Prefab devuelve los mismos elementos que vemos cuando ingresamos al modo Prefab para ese Prefab. Esto se debe al hecho de que el modo Prefab muestra el contenido de un Prefab y al desempaquetar una instancia de Prefab se desempaqueta el contenido de un Prefab.

Si deseamos reemplazar una instancia de Prefab con GameObjects normales y eliminar cualquier vínculo con cualquier Activo de Prefab, haga clic con el botón derecho en él en la Jerarquía y seleccione Desempaquetar Prefab completamente. Esto es idéntico a desempaquetar Prefab y continuar desempaquetando cualquier instancia de Prefab que se produzca debido a Prefabs anidados o base.

Instancias prefabricadas que residen en Escenas o dentro de otras

Los prefabricados se pueden desempaquetar.

Fundamentos de Unity3D: una mirada rápida a la física del juego

Unity es un motor de juego innegablemente poderoso. Como resultado, desarrolla su método para modelar la física de una manera altamente eficiente. Unity lo define como:

Para identificar interacciones entre GameObjects, Unity usa Rigidbodies y Colliders. Debe recordarse que Unity contiene dos sistemas distintos que no pueden comunicarse entre sí: un sistema de Física 3D y un sistema de Física 2D.

- **En pocas palabras:**
  - **Rigidbody Element:** Este componente se encarga de aplicar simulaciones físicas a GameObjects como Gravity y atributos como Mass y Velocity. Es un componente necesario para identificar colisiones. En Unity, hay dos tipos de componentes Rigidbody: 3D y 2D. Es fundamental elegir el adecuado para el tipo de entorno en el que opera.
  - **Componente Colisionador:** Considere este componente como el volumen de espacio que ocupa nuestro elemento.
    Los colisionadores proporcionan un método para registrar colisiones entre objetos en la escena. Vienen en una variedad de formas y tamaños para adaptarse a nuestras necesidades. El Collider, como el Rigidbody, debe elegirse correctamente para que Unity detecte las colisiones con precisión. Si estamos trabajando en 3D, asegúrese de que nuestros GameObjects no estén utilizando colisionadores 2D accidentalmente.
    Afortunadamente, todos los Colliders están etiquetados explícitamente.

Veamos brevemente un ejemplo de física elemental aplicada a un GameObject. Haremos un cubo para que sirva como nuestro piso y una esférica para que actúe como una pelota para simulaciones de física:

Los colisionadores están vinculados tanto al suelo como a la pelota. El Colisionador de Esferas atado a nuestra bola se puede ver en el inspector, pero no pasa nada al pulsar el botón Play. Unity no tiene forma de saber qué debe hacer ese GameObject.

Démosle a la esfera un cuerpo rígido. No es esencial agregar uno al piso porque la esfera manejará todas las colisiones en este caso:

Elegimos "RigidBody" en lugar de "RigidBody2D" porque este es un proyecto 3D usando 3D Colliders.

Ahora, presionemos el botón Reproducir:

Nuestra pelota ahora es impactada por la Gravedad de Rigidbody y procede a caer hasta que encuentra un contacto. Este ejemplo choca con el suelo y es detenido por el Box Collider atado a él. El inspector muestra los números de velocidad, velocidad y centro mundial de masa de la pelota a medida que cambian durante su descenso.

Solo la pelota tiene un componente de cuerpo rígido vinculado a ella, y puede detectar el colisionador en el piso y simular su contacto.

Física

Unity nos permite simular la física en nuestro proyecto para verificar que los objetos se aceleren y se comporten adecuadamente ante colisiones, gravedad y otras fuerzas. Unity tiene muchas implementaciones de motores de física que podemos utilizar según las demandas de nuestro proyecto: 3D, 2D, orientado a objetos u orientado a datos.

Proyectos orientados a objetos con motores de física incorporados
Si nuestro proyecto está orientado a objetos, utilice el motor de física integrado con Unity que mejor se adapte a nuestras necesidades:

- Física 3D incorporada.
- Hay física 2D incorporada.

Para tareas orientadas a objetos, use física 3D
Esta sección describe los componentes principales a los que se puede acceder a través del motor de física 3D integrado de Unity, que se puede usar para programas orientados a objetos. Contiene los siguientes artículos:

- Un resumen de los principales conceptos de física: cuerpos rígidos, colisiones y articulaciones, controladores de caracteres, articulaciones físicas y articulaciones físicas.
- Algunos ejemplos de explicaciones para ciertos contextos físicos: Detección continua de colisiones, así como física multiescena.
- La visualización de depuración de física se describe en detalle.

Además, la parte de referencia de física 3D incluye una descripción completa de todos los componentes accesibles, y la sección de CÓMO de física incluye algunos consejos sobre actividades típicas relacionadas con la física.

Referencia de física 2D
Utilice los atributos anteriores como configuración global para Physics 2D. Si deseamos administrar los parámetros de física 3D globales, use la configuración del Administrador de física en su lugar.

Los parámetros de Physics 2D definen la precisión de la simulación física. Se requiere una mayor sobrecarga de procesamiento para una simulación más realista, y estas opciones nos permiten modificar el compromiso entre precisión y rendimiento que mejor se adapte a nuestro proyecto.

Capítulo 4

Animación en Unity

Animaciones reasignables, control completo sobre los pesos de animación en tiempo de ejecución, llamada de eventos desde la reproducción de la animación, jerarquías y transiciones de máquinas de estado avanzadas, formas combinadas para animaciones de rostros y muchas más capacidades están disponibles en Unity Animation. Echemos un vistazo más de cerca a las animaciones en Unity.

DESCRIPCIÓN GENERAL DEL SISTEMA DE ANIMACIÓN

Unity presenta un sistema de animación complejo y poderoso (conocido como "Mecanim"). Incluye:

- Configuración simple de flujo de trabajo y animación para todos los elementos de Unity, incluidos objetos, personajes y atributos.
- Animación humanoide que reorienta la capacidad de aplicar animaciones de un modelo de personaje a otro.

  Compatibilidad con clips de animación importados y animación desarrollada en Unity.

DOI: 10.1201/9781003214755-4

- Se ha simplificado el flujo de trabajo para alinear clips de animación.

- Cómoda vista previa de clips de animación, transiciones e interacciones. Esto permite a los animadores trabajar independientemente de los programadores, crear prototipos y obtener una vista previa de sus animaciones antes de que se conecte el código del juego.

- Se utiliza una herramienta de programación visual para administrar relaciones complejas entre animaciones.

- Animar varias secciones del cuerpo usando diferentes razonamientos.

- Las capas y las máscaras son características valiosas.

FLUJO DE TRABAJO PARA LA ANIMACIÓN

El sistema de animación en Unity se basa en la noción de clips de animación, que contienen información sobre cómo deberían cambiar la posición, la rotación u otros atributos de objetos específicos con el tiempo. Cada clip se puede considerar una sola grabación lineal. Los clips de animación externos los crean artistas o animadores que utilizan tecnologías de terceros, como Autodesk® 3ds Max® o Autodesk® Maya®, o se originan en estudios de captura de movimiento u otros.

fuentes.

Luego, los clips de animación se organizan en un diagrama de flujo. como un sistema conocido como Animator Controller.

El controlador de animación funciona como una "máquina de estado", que realiza un seguimiento de qué clip debe reproducirse en un momento dado y cuándo las animaciones deben cambiar o combinarse.

Un controlador de animación sencillo podría incluir uno o dos clips, por ejemplo, para gestionar el giro y el rebote de un encendido o para animar una puerta que se abre y se cierra en el momento adecuado. Un controlador de animación más potente puede tener cientos de animaciones humanoides para todos los movimientos del personaje principal y la capacidad de mezclar numerosos clips a la vez para dar un movimiento suave mientras el jugador camina por el área.

El sistema de animación de Unity también incluye varias características únicas para trabajar con personajes humanoides, como la capacidad de redirigir la animación humanoide desde cualquier fuente (por ejemplo, captura de movimiento, la tienda de activos u otra biblioteca de animación de terceros) a nuestro modelo de personaje, así como ajustar las definiciones musculares. El sistema Avatar de Unity, que asigna avatares humanoides a una estructura interna estándar, permite varias funcionalidades particulares.

El componente Animator conecta cada una de estas partes (los clips de animación, el controlador Animator y el avatar) a un GameObject. Este componente contiene una referencia a un Animator Controller y (si corresponde) el Avatar para este modelo. El controlador de animación, a su vez, almacena referencias a los clips de animación que emplea.

- Los clips de animación se importan desde otra fuente o se generan dentro de Unity.
- Un controlador de animación es donde se insertan y organizan los clips de animación. En la ventana de Animador,

esta es una vista de un Animator Controller. Los estados (que pueden representar animaciones o máquinas de subestado anidadas) se definen como nodos conectados por líneas. Este Controlador de Animador se puede encontrar en la ventana Proyecto como un Activo.

- El modelo de personaje amañado tiene una estructura ósea única que se transfiere al formato Avatar estándar de Unity. Esta asignación se guarda como un recurso de avatar como parte del modelo de personaje importado y también es visible en la ventana Proyecto.

- Al animar el modelo de personaje, se vincula un componente Animator. El Componente Animador, que tiene asignados tanto el Controlador Animador como el Avatar, se puede ver en la vista Inspector.
  El animador los combina para animar el modelo.

  Al animar una figura humanoide, la referencia de Avatar es solo esencial. Solo se requiere un controlador de animación para otros tipos de animación.

SISTEMA DE ANIMACIÓN LEGADO

Si bien se prefiere Mecanim en la mayoría de los casos, Unity ha mantenido su tecnología de animación heredada antes de Unity 4. Cuando se trabaja con contenido anterior generado antes de Unity 4, es posible que debamos utilizarlo.

Clips de Animación

Los clips de animación son un componente clave del sistema de animación de Unity. Unity permite la importación de animaciones de otras fuentes y la creación de clips de animación desde cero dentro del editor a través de la ventana Animación.

Animación de origen externo

Los clips de animación importados externamente pueden incluir:

- Animaciones humanoides filmadas en un estudio de captura de movimiento.
- Un artista crea animaciones desde cero en una aplicación tridimensional (3D) externa (como Autodesk® 3ds Max® o Autodesk® Maya®).
- Conjuntos de animación de bibliotecas de terceros (por ejemplo, de la tienda de recursos de Unity).
- Se utilizó una única línea de tiempo importada para recortar y segmentar varios clips.

Se utilizó Unity para crear y editar la animación

También podemos generar y editar clips de animación en la ventana de animación de Unity. Estos clips son capaces de animar:

- Ubicación, rotación y escala de GameObjects.
- Los atributos de los componentes incluyen el color del material, la intensidad de la luz y el volumen del sonido.
- Podemos utilizar variables flotantes, enteras, enumeradas, vectoriales y booleanas en nuestros scripts.
- El orden en que se llaman las funciones dentro de nuestros scripts.

ANIMACIÓN DE FUENTE EXTERNA

La animación externa se importa a Unity de la misma manera que los archivos 3D estándar. Estos archivos pueden contener datos de animación en forma de una grabación lineal de los movimientos de los objetos dentro del archivo, ya sean archivos FBX genéricos.

o formatos nativos de aplicaciones 3D como Autodesk® Maya®, Cinema 4D, Autodesk® 3ds Max® o BlenderTM.

En algunos casos, el elemento que se va a animar (por ejemplo, un personaje) y las animaciones asociadas con él se pueden encontrar en el mismo archivo. En otras circunstancias, las animaciones pueden almacenarse en un archivo diferente del objeto animado.

Es posible que las animaciones sean específicas del modelo y no se puedan reutilizar en otros modelos. Un gran jefe final de pulpo en su juego, por ejemplo, puede tener una disposición única de extremidades y huesos, así como su propio conjunto de movimientos.

En otros casos, podemos tener una biblioteca de animaciones que queremos utilizar en diferentes modelos en nuestro escenario. Por ejemplo, varias figuras humanoides pueden utilizar las mismas animaciones de caminar y correr. En estos casos, es típico incluir un pequeño modelo de marcador de posición en nuestros archivos de animación para obtener una vista previa. Alternativamente, los archivos de animación se pueden usar incluso si no contienen geometría y solo los datos de animación

Al importar varias animaciones, pueden residir como archivos diferentes dentro de nuestra carpeta de proyecto, o podemos extraer numerosos clips de animación de un solo archivo FBX si se producen como tomas de Motion Builder o con un complemento/ script para Autodesk® Maya®, Autodesk® 3ds Max® u otros productos 3D. Si nuestro archivo tiene numerosas animaciones diferentes agrupadas en una sola línea de tiempo, es posible que deseemos hacer esto. Una línea de tiempo con un movimiento largo grabado, por ejemplo, puede tener la animación para algunos movimientos de salto distintos, y es posible que deseemos recortar partes específicas de esto para utilizarlas como clips individuales y eliminar el resto. Cuando cargamos todas las animaciones en una línea de tiempo, Unity incluye herramientas de corte de animación que nos permiten establecer el rango de fotogramas para cada clip.

Importación de archivos de animación

Cualquier animación debe importarse a nuestro proyecto antes de que pueda utilizarse en Unity. Unity es compatible con Autodesk® nativo
Archivos Maya® (.mb o.ma), Autodesk® 3ds Max® (.max) y BlenderTM (.blend), así como archivos FBX genéricos generados a partir de la mayoría de los paquetes de animación. Cabe señalar que la importación desde archivos .blend requiere la instalación de BlenderTM localmente.

Los datos de los archivos de animación importados se pueden ver y copiar

En el panel Animación, podemos ver los fotogramas clave y las curvas de los clips de animación importados. Cuando estos clips importados incluyen muchos huesos y fotogramas clave, la cantidad de información puede parecer abrumadora.

Seleccione los huesos individuales que queremos ver para reducir la vista.

Los fotogramas clave o las curvas de esos huesos se muestran en la ventana Animación.

Al examinar los fotogramas clave de animación importados, la ventana de animación muestra los datos de animación en modo de solo lectura.

Para modificar estos datos, abra Unity y cree un nuevo clip de animación vacío (consulte Creación de un nuevo clip de animación), luego seleccione, copie y pegue los datos de animación del clip de animación importado en su nuevo clip de animación editable.

AVATARES CON HUMANOIDE

El sistema de animación de Unity incluye funciones diseñadas específicamente para trabajar con personajes humanoides.
Debido a que las figuras humanoides son omnipresentes en los juegos, Unity

consta de un flujo de trabajo específico y un completo conjunto de herramientas para animaciones humanoides.

El sistema Avatar de Unity determina si un modelo animado tiene un diseño humanoide y qué partes del modelo se correlacionan con las piernas, los brazos, la cabeza y el cuerpo.

Debido a que las estructuras óseas de numerosas figuras humanoides son idénticas, es posible transferir movimientos de una a otra, lo que permite la reorientación y la cinemática inversa__ (IK)__.

AGREGAR MOVIMIENTOS HUMANOIDES A UN MODELO

Este seminario web nos guiará a través de los pasos para importar un modelo para usarlo con el sistema de animación de Unity.

El sistema de animación admite dos tipos de modelos:

1. Un modelo humanoide es una estructura especializada que contiene al menos 15 huesos que están estructurados para adherirse libremente a un esqueleto humano real. Esta página ofrece información sobre cómo importar este tipo de modelo.

2. Todo lo demás es un modelo genérico. Esto puede variar desde una tetera hasta un dragón. Consulte Importación de un modelo con animaciones no humanoides para obtener detalles sobre cómo hacerlo.

Visión general

Cuando Unity importa archivos de modelo que contienen animaciones y equipos humanoides, debe reconciliar la estructura ósea del modelo con su animación. Logra esto asignando como ping cada hueso en el archivo a un Avatar Humanoide para que la Animación pueda reproducirse correctamente. Por todo esto,

es fundamental preparar nuestro archivo Modelo antes de importarlo a Unity a fondo.

- Crear el Avatar y definir el tipo de Rig.
- Corregir o confirmar el mapeo del Avatar.
- Cuando hayamos terminado con el mapeo óseo, podemos ir a la pestaña Músculos y configuraciones para cambiar la configuración muscular del Avatar.
- Si lo desea, podemos guardar el mapeo de los huesos de su esqueleto en el Avatar como un archivo de plantilla humana (.ht).
- Al diseñar una máscara de avatar, podemos limitar la animación que se importa en ciertos huesos.
- Habilite la opción Importar animación desde la pestaña Animación antes de configurar los demás ajustes específicos del activo.
- Si el archivo tiene numerosas animaciones o acciones, puede usar Clips de animación para especificar rangos de acción particulares.
    - Alterar la postura y transformación de raíces.
    - Se debe optimizar el bucle.
    - La animación debe replicarse en ambos lados del esqueleto humanoide.
    - Para animar los tiempos de otras cosas, agregue curvas al clip.
    - Agregue eventos al clip para activar acciones específicas en sincronía con la animación.

- Deseche una parte de la animación de la misma manera que lo haría una máscara de avatar en tiempo de ejecución, pero esta vez en el momento de la importación.
- Para impulsar la acción, seleccione un Root Motion diferente Nodo.
- Leer cualquier mensaje de Unity con respecto a la importación del clip.
- Ver un adelanto del clip de animación.

- Para guardar nuestros cambios, seleccione el botón Aplicar en la parte inferior de la ventana Configuración de importación o Revertir para revertirlos.

Configuración de avatares

Establezca el Tipo de animación en Humanoide en la pestaña Rig de la ventana Inspector. El atributo Definición de avatar está establecido en Crear a partir de este modelo de forma predeterminada. Si conservamos esta opción, Unity intentará asignar el conjunto de huesos descrito en el archivo a un avatar humanoide.

En determinadas circunstancias, podemos modificar esta opción a Copiar de otro avatar para utilizar un avatar que ya hemos creado para otro archivo de modelo. Por ejemplo, supongamos que construimos una Malla (piel) en nuestra aplicación de modelado 3D con múltiples animaciones individuales. En ese caso, puede exportar la malla a un único archivo FBX y cada animación a un archivo FBX independiente. Al importar estos archivos a Unity, necesitamos crear un Avatar para el primer archivo (generalmente la Malla). Podemos reutilizar ese Avatar para el resto de los archivos siempre que todos utilicen la misma estructura ósea (por ejemplo, todas las animaciones).

Si activamos esta opción, debemos configurar el atributo Fuente para designar el Avatar que deseamos utilizar.

También podemos cambiar el número máximo de huesos que pueden afectar a un solo vértice con la propiedad Skin Weights.
Este parámetro restringe el impacto a cuatro huesos por defecto, pero podemos seleccionar más o menos.

Unity intenta hacer coincidir la estructura ósea actual con la estructura ósea de Avatar al hacer clic en el botón Aplicar.
Puede lograr esto automáticamente en muchas circunstancias al examinar las relaciones entre los huesos en la plataforma.

Aparece una marca de verificación junto al menú Configurar si la coincidencia es exitosa. Además, Unity agrega un subactivo de avatar al activo del modelo, que se puede encontrar en la vista del proyecto.

Unity pudo hacer coincidir los huesos esenciales, lo que resultó en una coincidencia exitosa. Sin embargo, para obtener los mejores resultados, haga coincidir los huesos opcionales y coloque el modelo en una posición en T adecuada.

Si Unity no puede crear el avatar, se muestra al lado del botón Configurar y no hay ningún subactivo de avatar en la vista del proyecto.

Debido a que el Avatar es una parte tan crucial del sistema de animación, debemos configurarlo correctamente para nuestro Modelo.

Como resultado, independientemente de si la creación automática de Avatar tiene éxito o falla, debemos verificar constantemente que nuestro Avatar sea legítimo y esté configurado correctamente.

Configurar el avatar

Si deseamos verificar que Unity asignó correctamente los huesos de su modelo al Avatar, o si Unity no pudo construir el Avatar para su modelo, podemos ingresar al modo de configuración de Avatar haciendo clic en el botón Configurar... en la pestaña Rig.

Si Unity produce un avatar con éxito, el avatar aparece como un subactivo del modelo de activo. Seleccione el Activo de Avatar en la ventana del Proyecto y haga clic en el botón "Configurar Avatar" en el Inspector para ingresar al modo de configuración de Avatar. Este modo nos permite ver y modificar cómo Unity asigna los huesos de su modelo al diseño de Avatar.

Cuando ingresamos al modo de configuración de Avatar, aparece la ventana de Avatar en el Inspector, que muestra el mapeo de huesos.

Verifique que el mapeo óseo sea correcto y que Unity no asignó ningún hueso opcional para ser mapeado.

Para que Unity cree una coincidencia válida, nuestro esqueleto debe tener al menos los huesos necesarios en su lugar. Para aumentar nuestras posibilidades de encontrar una coincidencia para el Avatar, nombra nuestros huesos según las partes del cuerpo que representan. Por ejemplo, los términos "Brazo izquierdo" y "Antebrazo derecho" hacen evidente lo que regulan estos huesos.

Mapeo de estrategia

Si el modelo no produce una coincidencia adecuada, podemos usar un procedimiento similar al que emplea Unity:

- Para borrar cualquier asignación que intentó Unity, seleccione Borrar en la opción Asignación en la parte inferior de la ventana de Avatar.
- Para imitar la postura de modelado original del modelo, seleccione Posición de enlace de muestra en el menú Pose en la parte inferior de la ventana Avatar.
- Construir un mapeo óseo a partir de una postura inicial, vaya a Mapeo > Automap.
- Seleccione Pose > Forzar T-Pose para devolver el modelo a la pose en T deseada.

Si el mapeo automático falla total o parcialmente, podemos asignar manualmente los huesos arrastrándolos desde las vistas de Escena o Jerarquía. Si Unity siente que un hueso es apropiado, se mostrará en verde en la pestaña Asignación de avatar; de lo contrario, aparecerá en rojo.

Cambiar la postura

La posición T es la postura predeterminada requerida por la animación de Unity y es la mejor postura para construir en cualquier software de modelado 3D. Si, por otro lado, no usó la pose T para construir nuestro personaje y la animación no funciona según lo planeado, podemos restablecer la animación seleccionando Restablecer en el menú desplegable Pose:

Si la asignación de huesos es precisa, pero el personaje no está en la postura adecuada, aparecerá el mensaje "El personaje no está en la postura T". Podemos intentar corregir esto seleccionando Forzar T-Pose en la opción Pose. Si la posición sigue siendo incorrecta, gire manualmente los otros huesos en una posición en T.

Cómo hacer una máscara de avatar

El enmascaramiento nos permite eliminar algunos de los datos de animación de un clip, lo que le permite animar partes específicas de un objeto o figura en lugar de todo. Por ejemplo, podemos tener una animación de caminar convencional que incluya acción de brazos y piernas, pero si un personaje lleva un objeto enorme con ambas manos, no querríamos que sus brazos se balancearan hacia los lados mientras camina. Sin embargo, podríamos utilizar la animación habitual de caminar mientras transportamos el objeto enmascarando el elemento de la parte superior del cuerpo de la animación de transporte y reproduciéndolo sobre la parte superior del movimiento de caminar.

El enmascaramiento se puede aplicar a los clips de animación durante la importación o el tiempo de ejecución. El enmascaramiento durante el tiempo de importación es deseable, ya que permite que los datos de animación rechazados se eliminen de nuestra compilación, lo que reduce el tamaño de los archivos y, por lo tanto, consume menos RAM. También acelera el procesamiento porque es necesario combinar menos datos de animación en tiempo de ejecución. El enmascaramiento de importación puede no ser apropiado para sus necesidades en algunas circunstancias. En tal situación, podemos aplicar una máscara en tiempo de ejecución generando un activo de máscara de avatar y utilizándolo en la configuración de capa de nuestro controlador de animación.

Para crear un activo de máscara de avatar vacío, puede hacer una de estas dos cosas:

1. En el menú Activos, seleccione Crear > Máscara de avatar.

2. En la vista Proyecto, elija el objeto Modelo para el que desea definir la máscara, luego haga clic con el botón derecho y seleccione Crear > Máscara de avatar.

AGREGAR NO HUMANOIDE (GENÉRICO) ANIMACIONES A UN MODELO

Esta sección describe cómo importar un modelo para usarlo con el sistema de animación de Unity. Consulte Creación de modelos para animación para obtener información sobre cómo crear un modelo para usar con el sistema de animación.

El sistema de animación admite dos tipos de modelos:

1. Un modelo humanoide es una estructura especializada que contiene al menos 15 huesos que están estructurados para adherirse libremente a un esqueleto humano real. La importación de un modelo con animaciones humanoides incluye instrucciones sobre cómo hacerlo.

2. Todo lo demás es un modelo genérico. Esto puede variar desde una tetera hasta un dragón. Esta página ofrece información sobre cómo importar este tipo de modelo.

Esquema

Cuando importamos un modelo genérico a Unity, debemos especificar qué hueso es el nodo raíz. Esto determina efectivamente el centro de masa del modelo.

Las configuraciones genéricas no usan la ventana Humanoid Avatar ya que solo hay un hueso para mapear. En consecuencia, la preparación para importar un archivo de modelo no humanoide a Unity requiere menos pasos que la preparación de un modelo humanoide.

- Configurar nuestro Rig para que sea Genérico.
- Al diseñar una máscara de avatar, puede limitar la animación que se importa en ciertos huesos.
- Habilite la opción Importar animación en el menú Animación y luego configure los demás parámetros específicos del activo.

- Si el archivo tiene muchas animaciones o acciones, podemos usar clips de animación para definir rangos de cuadros específicos.
- Podemos hacer lo siguiente para cada clip de animación descrito en el archivo:
  - Configure la postura y la transformación raíz.
  - Se debe optimizar el bucle.
  - Para animar los tiempos de otras cosas, agregue curvas al clip.

- Agregue eventos al clip para activar acciones específicas en sincronía con la animación.
- Deseche una parte de la animación de la misma manera que lo haría una máscara de avatar en tiempo de ejecución, pero esta vez en el momento de la importación.
- Para impulsar la acción, seleccione un Root Motion diferente Nodo.
- Leer cualquier mensaje de Unity con respecto a la importación del clip.
- Ver un adelanto del clip de animación.

- Haga clic en el botón Aplicar en la parte inferior del cuadro Configuración de importación para conservar nuestros cambios, o Revertir para deshacerlos.

Configuración de la plataforma

Establezca el tipo de Avatar (animación) en Genérico en la pestaña Plataforma de la ventana Inspector. La propiedad Definición de avatar está establecida en Crear a partir de este modelo de forma predeterminada y la opción del nodo Raíz está establecida en Ninguno.

En algunas circunstancias, podemos cambiar la opción Definición de avatar a Copiar de otro avatar para utilizar un Avatar que ya creamos para otro archivo de modelo cambiando la opción Definición de avatar a Copiar de otro avatar. Por ejemplo, suponga que construimos una malla (piel) en su aplicación de modelado 3D con varias animaciones individuales. En ese caso, podemos exportar la Malla a un solo archivo FBX y cada animación a un archivo FBX por separado.

Al importar estos archivos a Unity, solo necesitamos crear un avatar para el primer archivo (generalmente la malla). Podemos reutilizar ese Avatar para el resto de los archivos, siempre que todos utilicen la misma estructura ósea.

Si mantenemos seleccionada la opción Crear a partir de este modelo, necesitaremos elegir un hueso de la propiedad Nodo raíz.

Si elegimos Copiar de otro avatar como la opción Definición de avatar, debemos indicar qué Avatar desea utilizar seleccionando el atributo Origen.

También podemos modificar el número máximo de huesos que impactan en un vértice específico con la propiedad Skin Weights.
Este parámetro restringe el impacto a cuatro huesos por defecto, pero podemos seleccionar más o menos.

Unity produce un avatar genérico y agrega un subactivo de avatar bajo el activo del modelo en la vista del proyecto al hacer clic en el botón Aplicar.

Cabe señalar que el Avatar genérico no es lo mismo que el Avatar humanoide, aunque se muestra en la vista Proyecto y tiene el mapeo del nodo Raíz. Sin embargo, cuando hacemos clic en el icono de Avatar en la vista Proyecto para exponer sus propiedades en el Inspector, solo se muestra su nombre y no aparece el botón Configurar Avatar.

Cómo hacer una máscara de avatar

El enmascaramiento se puede aplicar a los clips de animación durante la importación o en tiempo de ejecución. El enmascaramiento durante el tiempo de importación es deseable, ya que permite que los datos de animación rechazados se eliminen de nuestra compilación, lo que reduce el tamaño de los archivos y, por lo tanto, consume menos RAM. También acelera el procesamiento porque es necesario combinar menos datos de animación en tiempo de ejecuci

El enmascaramiento de importación puede no ser apropiado para nuestras necesidades en algunas circunstancias. En tal situación, podemos aplicar una máscara en tiempo de ejecución generando un activo de máscara de avatar y utilizándolo en la configuración de capa de nuestro controlador de animación.

Para hacer un activo de máscara de avatar vacío, podemos hacer una de dos cosas:

1. En el menú Activos, seleccione Crear > Máscara de avatar.

2. En la vista Proyecto, elija el objeto Modelo para el que desea definir la máscara, luego haga clic con el botón derecho y seleccione Crear > Máscara de avatar.

Ahora podemos seleccionar qué huesos incluir u omitir de una jerarquía de transformación antes de agregar la máscara a una capa de animación o agregar una referencia dentro de la parte Máscara de la pestaña Animación.

Cuadro de diálogo Configuración de importación de modelo

Tenga en cuenta que estas opciones son solo para importar modelos y animaciones realizadas en la mayoría de los programas de modelado 3D. Los modelos construidos en SketchUp y SpeedTree, por otro lado, hacen uso de parámetros particulares. Consulte Configuración de SketchUp y Configuración de importación de SpeedTree para obtener más detalles.

Cuando colocamos archivos de modelo en la carpeta de activos de nuestro proyecto de Unity, Unity los importa y guarda automáticamente como activos de Unity. Seleccione el archivo en la ventana Proyecto para abrir la ventana Inspector y examine la configuración de importación.
Establezca la configuración en las cuatro pestañas de esta ventana para modificar cómo Unity importa el archivo especificado:

Un modelo 3D puede representar una persona, una estructura o un mueble. Unity genera numerosos activos a partir de un único archivo de modelo en determinadas circunstancias. El principal elemento importado en la ventana Proyecto es un modelo Prefabricado. Por lo general, el modelo Prefab también hará referencia a varios objetos de malla.

Un Rig (también conocido como esqueleto) es una colección de deformadores organizados en una jerarquía que animan una Malla (también conocida como máscara) en uno o más modelos construidos en una aplicación de modelado 3D como Autodesk® 3ds Max® o Autodesk® Maya ®. Unity genera un avatar para modelos humanoides y genéricos (no humanoides) para reconciliar el Rig importado con el GameObject de Unity.

Como un clip de animación, podemos describir cualquier cantidad de posturas que ocurren a lo largo de un conjunto de fotogramas, como caminar, correr o simplemente estar inactivo (moverse de un pie al otro). Podemos reutilizar clips para cualquier modelo con el mismo equipo. Un solo archivo puede incluir numerosas acciones distintas, cada una de las cuales puede definirse como un clip de animación independiente.

Los materiales y texturas se pueden extraer o dejar incluidos en el modelo También puede cambiar cómo se representa el material en el modelo.

LA PESTAÑA MODELO

Cuando elegimos un modelo, la configuración de importación para ese modelo se muestra en la pestaña Modelo de la ventana Inspector. Estas opciones tienen un impacto en los numerosos componentes y atributos del Modelo. Unity importa cada Activo usando estos parámetros; por lo tanto, podemos cambiar cualquier configuración para aplicarla a varios Activos en nuestro Proyecto.

Esta sección contiene información sobre cada una de las secciones de la pestaña Modelo:

A. Configuración a nivel de escena, como si importar luces y cámaras y qué factor de escala aplicar.

B. Propiedades específicas de las mallas.

C. Funciones relacionadas con la geometría, como topología, UV y normales.

Escena

Propiedad	**Función**
Factor de escala	Cuando la escala del archivo original (del archivo Modelo) no cumple con la escala especificada en su Proyecto, configure este valor para aplicar una escala global al Modelo importado, el sistema de física en Unity espera que 1 metro en el mundo del juego sea igual a 1 unidad en el archivo importado.
Convertir Unidades	Habilitar esta opción hace que la escala del modelo definida en el archivo del modelo se convierta a Escala de la unidad.
eje de horneado Conversión	Cuando importamos un modelo que utiliza un sistema de ejes diferente al de Unity, habilite esta función para hornear los resultados de la conversión de ejes directamente en los datos de activos de nuestra aplicación (por ejemplo, datos de vértice o animación). Deshabilite este parámetro para ajustar el componente Transform de GameObject raíz en tiempo de ejecución para imitar la conversión del eje.
Importar mezclaformas	Permita que Unity importe formas combinadas con nuestra malla habilitando esta función. Para obtener más información, consulte Importación de formas de fusión.
Importar visibilidad	Importe los parámetros de FBX que determinan si los componentes de MeshRenderer están habilitados o no (visibles).

(Continuado)

	Función
Se pueden Importar cámaras	importar cámaras de **propiedad** de nuestros archivos .FBX.
Importar luces	Se deben importar las luces de nuestro archivo .FBX.
Preservar la jerarquía	Incluso si este modelo solo tiene una raíz, genere siempre una raíz prefabricada explícita. Como táctica de optimización, el importador de FBX generalmente elimina los nodos raíz vacíos del modelo. Si tenemos varios archivos FBX con diferentes partes de la misma jerarquía, podemos usar esta opción para mantener la estructura original. File1.fbx, por ejemplo, tiene un equipo y una malla, pero file2.fbx contiene el mismo equipo pero simplemente la animación para ese equipo. Si importamos file2.fbx sin marcar esta casilla, Unity elimina el nodo raíz, las jerarquías no coinciden y la animación se rompe.
Ordenar jerarquía por Nombre	Habilite esta función para clasificar los GameObjects dentro de la jerarquía alfabéticamente. Para mantener el orden jerárquico definido en el archivo FBX, deshabilite este parámetro.

Importación de formas de fusión

Unity admite formas combinadas (morphing) y puede importar formas combinadas desde archivos FBX y DAE de programas de modelado 3D. También se pueden importar archivos FBX animados. Las formas combinadas en vértices, normales y tangentes se pueden animar en el nivel de vértice en Unity.

Las mallas pueden verse afectadas tanto por pieles como por formas de fusión al mismo tiempo. Cuando Unity importa mallas con formas combinadas, utiliza el componente SkinnedMeshRenderer (en lugar del componente MeshRenderer), sin importar si la malla tiene apariencia o no.

Unity importa la animación de formas combinadas como parte de la animación estándar: anima los pesos de las formas combinadas en SkinnedMeshRenderers.

Para importar formas de mezcla usando normales, use uno de los dos métodos siguientes:

1. Establezca Blend Shape Normals para atribuir a Import para que Unity pueda usar las normales del archivo FBX.

2. Establezca la opción Mezclar normales de forma en Calcular y Unity usará la misma lógica para calcular las normales en una malla y combinar formas.

Importación de visibilidad

La función Importar visibilidad en Unity puede leer las propiedades de visibilidad de los archivos FBX. Ajustando el Renderer.
Las propiedades habilitadas, los valores y las curvas de animación pueden activar o desactivar los componentes de MeshRenderer.

La herencia de visibilidad está habilitada de forma predeterminada, aunque puede estar deshabilitada. Por ejemplo, si la visibilidad de una malla principal se establece en 0, todos los renderizadores secundarios se deshabilitan de manera similar. En este escenario, se construye una curva de animación para cada parámetro Renderer.enabled del elemento secundario.

Algunos software de modelado 3D no admiten o tienen restricciones en los atributos de visibilidad. Se puede encontrar más información en:

- La importación de modelos de Autodesk® Maya® a Unity tiene varias limitaciones.

- Importar modelos de Blender a Unity tiene ciertas limitaciones.

Importación de cámaras

Al importar cámaras desde un archivo .FBX, Unity admite las siguientes propiedades:

Propiedad	Función
Modo de proyección	Perspectiva u ortográfica. La animación no es compatible.
Campo de visión	La animación es compatible.
Todas las características de la cámara física	Cuando importamos una cámara con propiedades físicas (por ejemplo, de Maya), Unity crea una cámara con el atributo de cámara física habilitado y los valores del archivo FBX para distancia focal, tipo de sensor, tamaño del sensor, cambio de lente y ajuste de puerta.
Recorte cercano y lejano Distancia del avión	En esta configuración, Unity no importa ninguna animación. Habilite la configuración Recortar manualmente al exportar desde 3ds Max; de lo contrario, la configuración predeterminada se utiliza en la importación.
Cámaras objetivo	Al importar una cámara de destino, Unity construye una cámara con una restricción LookAt que utiliza la fuente como objeto de destino.

Importación ligera

Se admiten los siguientes tipos de luz:

- Lugar.
- Omni.
- Área.
- Direccional.

Se admiten características de luz como las que se enumeran a continuación:

Propiedad	Función
Rango	Si UseFarAttenuation está habilitado, se utiliza FarAttenuationEndValue. La animación no es compatible con FarAttenuationEndValue.
Color	La animación es compatible.
Intensidad	La animación es compatible.
Ángulo de punto	La animación es compatible. Los focos son los únicos que tienen esta opción.

Restricciones

El escalado de las características de la luz se utiliza en varias aplicaciones de modelado 3D. Por ejemplo, podemos cambiar el cono de luz escalando un punto de luz según su jerarquía. Debido a que Unity no realiza esto, la iluminación puede parecer diferente en el juego.

El ancho y la altura de las luces de área no están definidos en el formato FBX. Algunos programas de modelado 3D carecen de esta función y deben depender del escalado para determinar la región rectangular. Como resultado, cuando se importan, las luces de área siempre tienen un tamaño de 1.

Las animaciones de luz dirigidas no son compatibles a menos que estén horneadas.

LA PESTAÑA DEL EQUIPO

Las opciones de la pestaña Rig controlan cómo Unity asigna los deformadores a la malla en el modelo importado para ser animado. Esto implica dar o establecer un Avatar para personajes humanoides. Esto implica ubicar un hueso raíz en el esqueleto para personajes no humanoides (genéricos).

Cuando elegimos un modelo en la vista Proyecto, Unity calcula automáticamente qué tipo de animación se adapta mejor al modelo y lo presenta en la pestaña Rig. Si el archivo nunca se ha importado a Unity, el Tipo de animación se establece en Ninguna.

Propiedad	**Función**
Tipo de animación	Especifique el estilo de animación.
Ninguna	No hay animación.
Legado	Se debe utilizar el sistema de animación heredado. Importe y utilice animaciones de la misma manera que lo hicimos en Unity 3.x y antes.
Genérico	Si nuestro rig no es humanoide, utilice el Sistema de Animación Genérico (cuadrúpedo o cualquier entidad a animar). Unity elige un nodo raíz por nosotros, pero podemos especificar otro hueso para que sirva como nodo raíz en su lugar.
Humanoide	Si nuestro rig es humanoide, usa el Sistema de Animación Humanoide (dos piernas, dos brazos y una cabeza). Unity normalmente reconoce el esqueleto y lo transfiere con precisión al Avatar. En algunas circunstancias, es posible que necesitemos actualizar la definición de avatar y configurar manualmente el mapeo.

Tipos de animación genérica

Los avatares no se utilizan en las animaciones genéricas, como sí lo hacen en las animaciones humanoides. Debemos indicar qué hueso es el nodo raíz ya que el esqueleto puede ser arbitrario. El nodo Raíz permite que Unity mantenga la consistencia entre los clips de animación para un modelo genérico y se mezcle adecuadamente entre las animaciones que no se crearon "en el lugar" (es decir, donde todo el modelo mueve su posición mundial mientras se anima).

Especificar el nodo raíz ayuda a Unity a distinguir entre el movimiento de los huesos entre sí y el movimiento del nodo raíz en el entorno (controlado desde OnAnimatorMove).

Propiedad	Función
Definición de avatar	Seleccionar donde queremos adquirir la definición de Avatar.
Crear a partir de este modelo	Crea un Avatar con este modelo.
Copiar de otro Avatar	Apunte a un Avatar que se ha configurado en otro modelo.
Nodo raíz	Elige un hueso para que sirva como nodo raíz del Avatar. Solo se puede acceder a esta opción si la Definición de avatar está establecida en Crear a partir de este modelo.
Fuente	Importe clips de animación de otro Avatar con la misma configuración. Si la definición de avatar está establecida en Copiar de otro avatar, esta función solo está disponible.
Pesos de la piel	Establezca un número máximo de huesos que pueden influir en un solo vértice.
Estándar (4 huesos)	Utilice un máximo de cuatro huesos para ejercer el máximo influencia. Esta es la opción predeterminada y es la preferida para obtener los mejores resultados.
Disfraz	Establecer nuestro límite para el número de huesos. Los atributos Max Bones/Vertex y Max Bone Weight aparecen cuando seleccionamos esta opción.
Max Huesos/ Vértice	Establecer el número máximo de huesos que un vértice específico puede influir. Podemos especificar de uno a 32 huesos por vértice; sin embargo, cuantos más huesos utilicemos para impactar un vértice, mayor será la penalización de rendimiento. Solo se puede acceder a esta opción si el parámetro Pesos de la piel está establecido en Personalizado.

(Continuado)

Propiedad	**Función**
Peso óseo máx.	Establezca el límite inferior para tener en cuenta el peso de los huesos. La fórmula de ponderación ignora cualquier valor inferior a este número, y Unity aumenta los pesos óseos superiores a este valor hasta un total de 1,0. Solo si el atributo Pesos de la piel está establecido en Personalizado, esta opción está disponible.
Optimizar juego Objeto	Elimine y guarde la jerarquía GameObject Transform del personaje importado en el componente Avatar y Animator. Cuando está habilitado, los SkinnedMeshRenderers del personaje utilizan el esqueleto interno del sistema de animación de Unity, lo que aumenta el rendimiento de las figuras animadas. Si la Definición de avatar está establecida en Crear a partir de este modelo, esta opción solo está disponible.
Transformaciones adicionales para exponer	Cuando Optimize Game Object está habilitado, podemos especificar qué rutas de transformación debe ignorar Unity. Consulte Incluir transformaciones adicionales para obtener más detalles. Esta sección solo se muestra si Optimize Game Object está habilitado.

ASIGNACIÓN DE AVATAR DE PESTAÑAS

Cuando Unity Editor está en el modo de configuración de avatar, aparece la pestaña Asignación de avatar.

Para ingresar al modo de configuración de avatar, realice una de las siguientes acciones:

1. En la ventana del proyecto, elija el recurso de avatar y luego haga clic en "Configurar avatar" en el Inspector, o
2. Seleccione el activo del modelo en la ventana del proyecto, vaya a la pestaña "Equipo" del Inspector y haga clic en "Configurar..." en el menú Definición de avatar.

Cuando estamos en el modo de Configuración de Avatar, el Inspector muestra la pestaña Asignación de avatar, que muestra Mapeo óseo de Unity:

A. Alterne entre las pestañas Mapeo y Músculos y ajustes usando estos botones. Antes de pasar de una pestaña a otra, debemos Aplicar o Revertir cualquier cambio que hayamos realizado.

B. Botones para cambiar entre las secciones del Avatar: Cuerpo, Cabeza, Mano Izquierda y Mano Derecha.

C. Menús con numerosas herramientas de Mapeo y Pose para ayudarnos a mapear la estructura ósea del Avatar.

D. Botones para aceptar modificaciones (Aceptar), revertir cambios (Revertir) y salir de la ventana Avatar (Listo). Antes de salir de la ventana de Avatar, debemos Aplicar o Revertir cualquier cambio que hayamos hecho.

Guardar y reutilizar datos de avatar

La asignación de los huesos de nuestro esqueleto al Avatar se puede guardar en un disco como un archivo de plantilla humana (extensión *.ht).
Esta asignación se puede utilizar para cualquier carácter. Por ejemplo, supongamos que queremos enviar el mapeo de Avatar al control de código fuente y preferimos archivos basados en texto; o supongamos que queremos analizar el archivo con nuestra herramienta personalizada.

Elija Almacenar en la opción desplegable Mapeo en la parte inferior de la ventana de Avatar para guardar los datos de Avatar en un archivo de plantilla humana.

Unity muestra una ventana de diálogo en la que podemos especificar el nombre y la ubicación del archivo guardado.

Para cargar un archivo de plantilla humana previamente preparado, vaya a Mapeo > Cargar y seleccione el archivo.

Hacer uso de máscaras de avatar

Puede ser ventajoso limitar la animación a determinadas partes del cuerpo a veces. Por ejemplo, en un movimiento de paseo, la figura puede balancear los brazos, pero si toma una antorcha, debe sostenerla para iluminar. Se puede usar una máscara corporal de avatar para determinar a qué áreas de la animación de un personaje debe limitarse.

EL MÚSCULO DE AVATAR Y LA PESTAÑA DE CONFIGURACIÓN

Usando Muscles, podemos controlar el rango de movimiento de varios huesos en el sistema de animación de Unity.

Cuando el Avatar está correctamente configurado, el sistema de animación “entiende” la estructura ósea y nos permite usar la pestaña Músculos y ajustes del Inspector de Avatar. Utilice la página Músculos y configuraciones para ajustar el rango de movimiento del personaje y verifique que el personaje se deforme de manera convincente, sin artefactos visuales ni superposiciones.

La pestaña Muscle & Settings tiene las siguientes secciones:

A. Cambia entre las pestañas Mapeo y Músculos y ajustes. Antes de pasar de una pestaña a otra, debemos Aplicar o Revertir cualquier cambio que hayamos realizado.

B. Manipule el personaje utilizando deformaciones predeterminadas en el cuadro Vista previa del grupo muscular. Estos tienen un impacto en muchos huesos al mismo tiempo.

C. Ajuste huesos particulares del cuerpo utilizando la sección Configuración por músculo. Podemos ajustar las limitaciones de rango de cualquier opción ampliando los ajustes de los músculos. Por ejemplo, Head-Nod y Head-Tilt de Unity

Los parámetros tienen un rango predeterminado de -40 a 40 grados, pero puede disminuir estos rangos aún más para endurecer estos movimientos.

D. Ajuste efectos particulares en el cuerpo utilizando la configuración adicional.

E. El menú Músculos tiene un botón Restablecer que restablece todos parámetros musculares a sus niveles preestablecidos.

F. Botones para aceptar modificaciones (Aceptar), revertir cambios (Revertir) y salir de la ventana Avatar (Listo). Antes de salir de la ventana de Avatar, debemos Aplicar o Revertir cualquier cambio que hayamos hecho.

Cambios en vista previa

Es posible que veamos los cambios en las secciones Vista previa del grupo muscular y Configuración por músculo directamente en la vista Escena. Podemos ver el rango de movilidad para cada configuración aplicada a nuestro personaje arrastrando los controles deslizantes hacia la izquierda y hacia la derecha.

A través de la Malla, podemos ver los huesos del esqueleto.

Grado de libertad Traducir

Para habilitar las animaciones de traducción para el humanoide, active la opción Traducir DoF en la Configuración adicional. Cuando esta opción está desactivada, Unity solo usa rotaciones para animar los huesos. La traducción DoF es accesible para los siguientes músculos: Chest, UpperChest, Neck, LeftUpperLeg, RightUpperLeg, LeftShoulder y RightShoulder.

Habilitar Translate DoF puede aumentar los requisitos de rendimiento, ya que el sistema de animación debe ejecutar un paso adicional para volver a orientar la animación humanoide. Como resultado,

solo deberíamos usar esta opción si sabemos que nuestra animación incorpora traducciones animadas de algunos de los huesos de nuestro personaje.

LA VENTANA PARA LA MÁSCARA DE AVATAR

Hay dos métodos para especificar qué elementos de nuestra animación deben enmascararse:

1. Eligiendo un mapa corporal humanoide.
2. Especificando qué huesos deben incluirse o excluirse de una jerarquía de transformación.

Elegir un cuerpo humanoide

Si nuestra animación incluye un avatar humanoide, podemos usar el diagrama de cuerpo humanoide simplificado para determinar dónde ocultar la animación.

El diagrama del cuerpo divide el cuerpo en los siguientes secciones:

- Cabeza.
- Brazo derecho.
- Brazo izquierdo.
- Mano derecha.
- Mano izquierda.
- Pierna derecha.
- Pierna izquierda.
- Raíz.

Para agregar movimiento a una de estas partes del cuerpo, haga clic en el diagrama de Avatar para esa parte hasta que se vuelva verde. Para desactivar la animación, haga clic en la parte del cuerpo hasta que se vuelva roja. Haga doble clic en el espacio vacío alrededor del Avatar para incluir u omitir todo.

Alterne Inverse Kinematics__ (IK)__ para manos y pies, que regula si las curvas IK se incluyen o no en la mezcla de animación.

Selección de una transformación

Si nuestra animación no emplea un avatar humanoide, o si queremos tener más control sobre qué huesos específicos están enmascarados, podemos seleccionar o anular la selección de elementos de la jerarquía del modelo:

- Asignar una referencia al Avatar cuya transformación queremos ocultar.
- Seleccione la opción Importar esqueleto. En el inspector, podemos ver la jerarquía del avatar.
- Podemos verificar cada hueso en la jerarquía para ver cuál queremos usar como nuestra máscara.

Los activos de máscara se pueden utilizar en los controladores de animador al definir capas de animación o en la configuración de importación de nuestros archivos de animación para aplicar máscaras durante la importación. animación.

Las máscaras tienen la ventaja de reducir la sobrecarga de memoria, ya que las partes del cuerpo que no están activas no requieren sus correspondientes curvas de animación. Además, no es necesario calcular las curvas no utilizadas durante la reproducción, lo que reduce el costo de CPU de la animación.

VENTANA PLANTILLA HUMANA

Un archivo de plantilla humana (*.ht) es un archivo YAML que contiene un mapeo de huesos humanoides para un modelo que guardamos en la ventana de Avatar.

El contenido de los archivos de plantilla humana se muestra como campos de texto ordinarios de Unity en la ventana Plantilla humana.

Cada agrupación corresponde a una entrada de archivo YAML, con el nombre del objetivo de mapeo óseo etiquetado como Primero y el nombre del hueso en el archivo Modelo etiquetado como Segundo.

Este atributo nos permite alterar la mayoría de los datos en este archivo; sin embargo, Unity actualiza instantáneamente cualquier modificación que hagamos al archivo. Sin embargo, podemos deshacer cualquier cambio realizado mientras esta ventana esté abierta.

INSTRUCCIONES DE LA VENTANA DE ANIMACIÓN

La ventana de animación en Unity nos permite crear y editar clips de animación desde dentro del programa. Está destinado a ser una alternativa fuerte y simple a las aplicaciones de animación 3D externas. Además de animar el movimiento, el editor nos permite animar variables de componentes y materiales y complementar nuestros clips de animación con eventos de animación, que son funciones que se invocan en momentos específicos a lo largo de la línea de tiempo.

HACIENDO USO DE LA VISTA DE ANIMACIÓN

En Unity, la vista Animación se usa para examinar y modificar Clips de animación para GameObjects animados. En Unity, vaya a Ventana > Animación para obtener la vista Animación.

Ver animaciones en un GameObject

La ventana Jerarquía, la ventana Proyecto, la vista Escena y la ventana Inspector están todas relacionadas con la ventana Animación. La ventana Animación, al igual que el Inspector, muestra la línea de tiempo y los fotogramas clave de la Animación para el GameObject o el Activo de Clip de Animación seleccionado actualmente. Podemos seleccionar un GameObject desde la ventana Jerarquía o la Vista de escena, o podemos seleccionar un Activo de clip de animación desde la Ventana del proyecto.

La lista de propiedades animadas

La vista Animación (izquierda) muestra la Animación utilizada por el GameObject elegido actualmente y cualquier GameObject secundario controlado por esta Animación. Las vistas Escena y Jerarquía están a la derecha, lo que indica que la vista Animación muestra las Animaciones asociadas con el GameObject seleccionado actualmente.

Se puede ver una lista de atributos animados en el lado izquierdo de la vista Animación. Esta lista está vacía en un clip recién producido cuando aún no se ha capturado ninguna animación.

Cuando empecemos a animar atributos específicos en este clip, las propiedades animadas se mostrarán aquí.

Si la animación controla muchos objetos secundarios, la lista también incluirá sublistas jerárquicas de los objetos animados. atributos de cada elemento secundario. Diferentes secciones de la

La jerarquía GameObject de Robot Arm está animada dentro del mismo clip de animación del ejemplo anterior.

Al animar una jerarquía de GameObjects dentro de un solo clip, asegúrese de que la animación se cree en el GameObject raíz de la jerarquía.

Cada propiedad se puede plegar y desplegar para mostrar los valores precisos capturados en cada fotograma clave. Los campos de valor muestran el valor interpolado si el cabezal de reproducción (la línea blanca) está entre fotogramas clave. Podemos modificar directamente estos campos.

Cuando se realizan ajustes mientras el cabezal de reproducción está sobre un fotograma clave, los valores de ese fotograma clave se actualizan si se realizan modificaciones cuando el cabezal de reproducción está entre fotogramas clave, se produce un nuevo fotograma clave con el nuevo valor que ingresamos en ese punto.

Se ha desplegado una propiedad en la vista de animación, permitiendo que el valor del fotograma clave se introduzca directamente.

Cronología de la animación

La línea de tiempo del clip actual se muestra en el lado derecho de la Vista de animación. Esta línea de tiempo muestra los fotogramas clave de cada propiedad de animación. La vista de línea de tiempo está disponible en dos modos: Dopesheet y Curves. Vaya a la parte inferior del área de la lista de propiedades animadas y haga clic en Dopesheet o Curve para cambiar entre estos modos.

Estos proporcionan dos perspectivas diferentes sobre la línea de tiempo de la animación y los datos de fotogramas clave.

Modo de línea de tiempo en Dopesheet

El modo Dopesheet proporciona una vista más condensada, permitiéndonos ver la secuencia de fotogramas clave de cada propiedad en su pista horizontal. Esto proporciona un resumen rápido de la sincronización de fotogramas clave para varias propiedades o GameObjects.

Modo de línea de tiempo para curvas

El modo de curvas muestra un gráfico de tamaño variable que muestra cómo los valores de cada atributo de animación varían con el tiempo. Dentro de la misma vista de gráfico, todos los atributos elegidos muestran la pila.

Este modo nos da un control completo sobre cómo se muestran y editan los valores y cómo se interpolan entre ellos.

Adaptando nuestra elección a la ventana

Cuando vea nuestra Animación en el modo Curvas, recuerde que los rangos variados para cada parámetro pueden variar sustancialmente a veces. Considere un clip de animación introductorio de un cubo que gira y rebota. El valor de la posición Y de rebote puede oscilar entre 0 y 2 (lo que significa que el cubo rebota dos unidades de altura durante la animación); sin embargo, el valor de rotación puede oscilar entre 0 y 360 (que representan sus grados de rotación). Al ver estas dos curvas al mismo tiempo, será difícil discernir las curvas de animación para los valores de posición, ya que la vista se alejará para adaptarse al rango de 0–360 de valores de rotación dentro de la ventana.

Se eligen las curvas de posición y rotación de un cubo giratorio que rebota, pero debido a que la pantalla se aleja para cumplir con el rango de 0–360 de la curva de rotación, es difícil distinguir la curva de posición Y que rebota.

Para acercar los fotogramas clave seleccionados actualmente, presione F en el teclado. Esto es excelente para enfocar y redimensionar rápidamente la ventana en una sección de nuestra línea de tiempo de Animación para una mejor edición.

Haga clic en propiedades específicas en la lista y presione F en el teclado para volver a escalar la pantalla para que se ajuste al rango de valores. También podemos ampliar manualmente la ventana Curvas arrastrando los controladores en cualquiera de los extremos de los controles deslizantes de la barra de desplazamiento de la vista. La ventana de animación se amplía para mostrar la animación de la posición Y que rebota. El comienzo de

la curva giratoria amarilla todavía es visible, pero ahora se ha extendido mucho más allá de la parte superior.

Para ajustar y volver a escalar la ventana para mostrar todos los fotogramas clave en el clip, presione A en el teclado. Si deseamos ver la línea de tiempo completa manteniendo su selección actual, intente esto.

Controles para reproducción y navegación por fotogramas

Utilice los controles de reproducción en la parte superior izquierda de la vista de animación para controlar la reproducción del clip de animación.

Estos son los controles, de izquierda a derecha:

- Activar/desactivar el modo de vista previa.
- Modo de grabación (activar/desactivar) Si el modo de grabación está activado, el modo de vista previa siempre está activado.
- Ajuste el cabezal de reproducción al inicio del clip.
- Vuelva a colocar el cabezal de reproducción en el fotograma clave anterior.
- La animación debe reproducirse.
- Navegue el cabezal de reproducción al siguiente fotograma clave.
- Coloque el cabezal de reproducción al final del clip.

También podemos controlar el cabezal de reproducción usando los atajos de teclado que se enumeran a continuación:

- Para volver al cuadro anterior, presione la coma (,).
- Para pasar al cuadro siguiente, utilice la tecla Punto (.).
- Para volver al fotograma clave anterior, mantenga presionada la tecla Alt y presione Coma (,).
- Para ir al siguiente fotograma clave, mantenga presionada la tecla Alt y presione Punto (.).

Bloqueo de ventana

Podemos evitar que la ventana del editor de animación cambie automáticamente para reflejar el GameObject elegido actualmente en la jerarquía o la escena bloqueándola.
Bloquear la ventana es esencial si queremos concentrarnos en la Animación de un GameObject mientras seguimos seleccionando y manipulando otros GameObjects en la Escena.

HACER UN NUEVO CLIP DE ANIMACIÓN

Seleccione un GameObject en nuestra escena y abra la ventana de animación para crear un nuevo clip de animación (menú superior :) Se puede acceder a la animación a través de Ventana > Animación > Animación.

Si no se han agregado clips de animación al GameObject, el botón "Crear" se muestra en el área de la línea de tiempo de la ventana de animación.

Seleccione la opción Crear. Unity nos invita a guardar nuestro clip de animación vacío recién creado en nuestra carpeta de activos.

Unity realiza las siguientes acciones cuando guarda este nuevo clip de animación vacío:

- Esta función genera un nuevo Animator Controller Activo.
- El nuevo clip se agrega como estado predeterminado al Animator Controller.
- Agrega un Componente Animador al GameObject al que se aplica la animación.
- El nuevo Animator Controller se asigna al Animator Component.

Incluyendo otro clip de animación

El botón "Crear" no se muestra si GameObject ya ha asignado uno o más clips de animación. En la ventana Animación, en cambio, se muestra uno de los clips existentes. Para cambiar entre clips de animación, utilice el menú ubicado en la esquina superior izquierda de la ventana de animación, junto a los controles de reproducción.

Seleccione Crear nuevo clip en este menú para agregar un nuevo clip de animación a las animaciones de un GameObject existente.
Antes de que podamos comenzar con su nuevo clip de animación vacío, Unity nos invita a guardarlo.

Cómo funciona todo junto

Los pasos anteriores crean automáticamente los componentes y referencias necesarios. Sin embargo, es beneficioso comprender cómo encajan los componentes.

- Se requiere un componente Animator para un GameObject.
- Se debe adjuntar un recurso de controlador de animador a el componente Animador.
- Se deben adjuntar uno o más clips de animación a el activo del controlador de animador.

Tras la creación de un nuevo clip de animación, ahora podremos ver:

- La ventana de animación (arriba a la izquierda) muestra una línea de tiempo con un título de reproducción blanco, lista para grabar nuevos fotogramas clave. El nombre del clip se puede ver en el menú de clips, directamente debajo de los controles de reproducción.

- El Inspector (centro) revela que el “Cubo” GameObject contiene un componente Animator. El campo Controlador del componente revela que está asignado a un Activo de controlador de Animator denominado Cubo.

- La Ventana de Proyecto (abajo a la derecha) muestra dos nuevos Activos: un Activo de Controlador de Animador llamado Cubo y un Activo de Clip de Animación llamado Clip de Animación de Cubo.

- La ventana de Animator (abajo a la izquierda) muestra el contenido del controlador de Animator. Hay un clip de animación de cubo en el controlador y está en el estado predeterminado (como lo indica el color naranja).
  Los clips subsiguientes agregados al controlador aparecen en gris, lo que indica que no se encuentran en la condición predeterminada.

AGREGAR ANIMACIÓN A UN GAMEOBJECT

Una vez que hayamos guardado el nuevo recurso de clip de animación, podemos comenzar a agregar fotogramas clave al clip.

En la ventana Animación, tenemos dos opciones para animar GameObjects: Modo de grabación y Modo de vista previa.

1. **Modo de grabación:** cuando movemos, giramos o ajustamos cualquier propiedad animable en nuestro GameObject animado en modo de grabación, Unity produce automáticamente fotogramas clave en el cabezal de reproducción. Para habilitar el modo de grabación, presione el botón con el círculo rojo.
   Cuando está en modo de grabación, la línea de tiempo de la ventana de animación está sombreada en rojo.

2. **Modo de vista previa:** la modificación de nuestro GameObject animado en el modo de vista previa no produce fotogramas clave de inmediato. Cada vez que cambiamos el estado de su

GameObject, debemos agregar fotogramas clave manualmente (por ejemplo, moverlo o rotarlo). Para habilitar el modo de vista previa, haga clic en el botón Vista previa. Cuando está en modo de vista previa, la línea de tiempo de la ventana de animación está sombreada en azul.

Grabación de fotogramas clave

Haga clic en el botón Grabar animación para comenzar a grabar fotogramas clave para el GameObject seleccionado. Esto activa el modo de grabación de animación, que registra los cambios del GameObject en el clip de animación.

Una vez en el modo de grabación, podemos agregar fotogramas clave arrastrando el cabezal de reproducción blanco al tiempo deseado en la línea de tiempo de animación y luego modificando nuestro GameObject al estado correcto en ese momento.

Los cambios en GameObject se guardan como fotogramas clave actualmente, que se muestran con la línea blanca en la ventana de animación.

Cualquier modificación a una propiedad animable (como su posición o rotación) dará como resultado la aparición de un cuadro clave para esa propiedad en la ventana de Animación.

Seleccionar o arrastrar en la barra de la línea de tiempo cambia el cabezal de reproducción y muestra el estado de la animación en el momento actual del cabezal de reproducción.

La ventana de Animación se ve en modo de grabación. La barra de la línea de tiempo tiene un tinte rojo, lo que indica que está en modo de grabación, y las propiedades de la animación tienen un fondo rojo en el inspector.

Pulsando de nuevo el botón de Grabar, podemos salir del Modo de Grabación en cualquier momento. Cuando salimos del modo de grabación, la ventana de animación cambia al modo de vista previa, lo que nos permite ver el GameObject en su posición actual a lo largo de la línea de tiempo de la animación.

Podemos animar cualquier propiedad de GameObject modificándola mientras está en el modo de grabación de animación. Mover, rotar o escalar el GameObject agrega fotogramas clave al clip de animación para esos atributos.

En el modo de grabación, al ajustar los valores directamente en el inspector de GameObject, se insertan fotogramas clave. Esto es cierto para cada propiedad que se puede animar en el inspector, incluidos los valores numéricos, las casillas de verificación, los colores y la mayoría de los demás valores.

Todos los atributos de GameObject que ahora están animados se enumeran en el lado izquierdo de la ventana de animación. Esta ventana no muestra propiedades que no estén animadas. Todas las propiedades nuevas que anime, incluidas las de los objetos secundarios, se agregan al área de la lista de propiedades tan pronto como se activen.
están activos.

Los atributos de transformación son únicos en el sentido de que las propiedades .x, .y y .z están conectadas, lo que permite insertar curvas para los tres simultáneamente.

Al presionar el botón Agregar propiedad, también puede agregar propiedades animables al GameObject actual (y sus descendientes). Cuando hacemos clic en este botón, aparece una lista de las características animables de GameObject en una ventana emergente.

Estos corresponden a las propiedades especificadas en el inspector.

La línea vertical blanca en el modo Vista previa o Grabar indica qué fotograma del clip de animación se está viendo en ese momento. El GameObject es visible en el Inspector y en la Vista de escena en ese cuadro del Clip de animación. Los valores de las propiedades animadas en ese cuadro también se muestran en una columna a la derecha de los nombres de las propiedades.

línea de tiempo

Podemos cambiar el cabezal de reproducción a cualquier cuadro en la línea de tiempo de la ventana de animación haciendo clic en cualquier lugar y luego previsualizando o ajustando ese cuadro en el clip de animación. Los números en la línea de tiempo se muestran en segundos y cuadros; por lo tanto, 1:30 es igual a un segundo y treinta fotogramas.

En el modo de vista previa, puede crear fotogramas clave

Además de usar el modo de grabación para producir fotogramas clave automáticamente cuando modifica un GameObject, podemos crear fotogramas clave en el modo de vista previa modificando una propiedad de GameObject y luego eligiendo manualmente crear un fotograma clave para esa propiedad.

En el modo de vista previa, las propiedades de la animación en la ventana del Inspector están sombreadas en azul. Cuando notamos este tinte azul, implica que estos valores están siendo controlados por los fotogramas clave del clip de animación que ahora se muestran en la ventana de animación.

Si cambiamos cualquiera de estas propiedades teñidas de azul durante la vista previa, el GameObject entra en un estado de animación modificado. Esto se indica mediante un cambio de tonalidad rosa en el tono del campo de inspección. Debido a que no estamos en modo de grabación, su cambio aún no se ha almacenado como un fotograma clave.

Crear fotogramas clave de forma manual

Cuando hemos editado un GameObject en modo de vista previa, hay tres formas de generar un fotograma clave manualmente.

Podemos agregar un fotograma clave haciendo clic con el botón derecho en la etiqueta de propiedad de la propiedad que hemos actualizado y seleccionado

"Agregar un fotograma clave solo para esa propiedad" o "Agregar un fotograma clave para todas las propiedades animadas":

Cuando agregamos un fotograma clave, el nuevo fotograma clave aparece como un símbolo de diamante en la ventana del animador. El campo de propiedad vuelve a tener un tinte azul, lo que indica que nuestro cambio se almacenó como un fotograma clave y que ahora estamos viendo un valor impulsado por los fotogramas clave de la animación.

En la ventana Animación, también podemos crear un cuadro clave haciendo clic en el botón Agregar cuadro clave:

Alternativamente, podemos insertar un fotograma clave usando el botón caliente teclas K o Shift-K, como se indica a continuación:

- **Atajos de teclado:**
  - **K:** Tecla toda animada. Agrega un fotograma clave para todas las propiedades animadas en la ventana de animación en la ubicación actual del cabezal de reproducción.
  - **Shift-K:** Modificar todas las teclas. Solo agrega un fotograma clave para las propiedades animadas que se han cambiado en la ubicación actual del cabezal de reproducción en la ventana de animación.

CONTROLADORES PARA ANIMADORES

Un controlador de animador se usa para crear y administrar una colección de animaciones para un personaje u otro objeto de juego animado.

El controlador contiene referencias a los clips de animación utilizados en su interior. Controla los muchos estados de animación y las transiciones entre ellos mediante una máquina de estados, que puede considerarse como un diagrama de flujo o un programa básico escrito en el lenguaje de programación visual de Unity.

SISTEMA DE NAVEGACIÓN DE UNITY

Podemos diseñar personajes que puedan viajar por el entorno del juego utilizando el sistema de navegación. Permite que nuestros personajes comprendan que deben usar escaleras para llegar al segundo piso o saltar para cruzar una zanja. El sistema Unity NavMesh está compuesto por los siguientes componentes:

- NavMesh (abreviatura de Navigation Mesh) es una estructura de datos que define las superficies transitables del mundo del juego y nos permite identificar un camino de un área transitable a otra. La estructura de datos se genera automáticamente en función de la geometría de su nivel.
- El componente NavMesh Agent nos permite diseñar personajes que se evitan unos a otros a medida que avanzan hacia su objetivo. Los agentes usan NavMesh para razonar sobre el mundo de los juegos y saben cómo evitarse unos a otros y cómo mover obstáculos.
- El componente Vínculo fuera de la malla nos permite proporcionar accesos directos de navegación que una superficie transitable no puede representar. Los enlaces fuera de la malla incluyen cosas como saltar una zanja o una cerca o abrir una puerta antes de atravesarla.
- Podemos usar el componente NavMesh Obstacle para definir impedimentos móviles que los agentes evitan mientras navegan por el mundo. Un obstáculo es algo así como un barril o un contenedor que está regulado por el sistema físico. Los agentes hacen todo lo posible para evitar el obstáculo mientras se mueve. Aún así, una vez que se detiene,

cortará un agujero en NavMesh para que los agentes puedan modificar sus rumbos para sortearlo, o si la obstrucción estacionaria está bloqueando el camino principal, los agentes pueden elegir otra ruta.

Diseño de interfaces de usuario (UI)

Unity ofrece tres sistemas de interfaz de usuario para crear una interfaz de usuario para Unity Editor y aplicaciones creadas en Unity Editor:

1. El paquete de interfaz de usuario de Unity.
2. Kit de herramientas de interfaz de usuario.
3. IMGUI.

Kit de herramientas para interfaces de usuario

El kit de herramientas de interfaz de usuario es el sistema de interfaz de usuario más nuevo de Unity. Se basa en tecnologías web estándar y está destinado a mejorar el rendimiento en todas las plataformas. Cuando instalamos el paquete UI Toolkit, podemos usarlo para desarrollar extensiones para el editor de Unity, así como la interfaz de usuario en tiempo de ejecución para juegos y aplicaciones.

El kit de herramientas de interfaz de usuario comprende lo siguiente:

- Un sistema de interfaz de usuario en modo retenido que incluye las funciones y capacidades esenciales necesarias para construir interfaces de usuario se incluye en el kit de herramientas de interfaz de usuario.
- Los tipos de activos de la interfaz de usuario están influenciados por formatos web estándar como HTML, XML y CSS. Úselos para organizar y diseñar interfaces de usuario.
- Herramientas y recursos para aprender a usar UI Toolkit, así como desarrollar y depurar interfaces.

Unity quiere que UI Toolkit sea el sistema de UI predeterminado para nuevos proyectos de desarrollo de UI, aunque carece de varias funcionalidades presentes en Unity UI (uGUI) e IMGUI.

El paquete de interfaz de usuario de Unity

El paquete Unity UI (Unity UI) (también conocido como uGUI) es un marco de interfaz de usuario más antiguo basado en GameObject para desarrollar una interfaz de usuario en tiempo de ejecución para juegos y aplicaciones. Utiliza componentes y la vista del juego para estructurar, posicionar y diseñar la interfaz de usuario en Unity UI. Tiene potentes funciones de texto y representación.

IU gráfica de modo inmediato

La interfaz de usuario gráfica de modo inmediato es un kit de herramientas de interfaz de usuario basado en código que dibuja y administra las interfaces de usuario mediante la función OnGUI y los scripts que la implementan. IMGUI se puede usar para construir inspectores personalizados para componentes de secuencias de comandos, extensiones de Unity Editor y pantallas de depuración en el juego. No se recomienda para crear interfaces de usuario en tiempo de ejecución.

Elegir un sistema de interfaz de usuario para nuestro proyecto

Unity quiere que UI Toolkit sea el sistema de UI predeterminado para nuevos proyectos de desarrollo de UI, aunque carece de varias funcionalidades presentes en Unity UI (uGUI) e IMGUI.
Estas tecnologías más antiguas son superiores en situaciones de uso específicas y deben ser respaldadas para sustentar los proyectos heredados.

El tipo de sistema de interfaz de usuario que seleccione para un proyecto específico está determinado por el tipo de interfaz de usuario que pretende crear y las funciones que necesitamos soporte.

Audio

Sonido espacial 3D completo, mezcla y masterización en tiempo real, jerarquías de mezcladores, instantáneas, efectos preconfigurados y muchas más capacidades están disponibles en Unity Audio. Esto también incluye los sonidos del juego.

Dicho esto, centraremos nuestra atención en la optimización del rendimiento en el próximo capítulo.

Capítulo 5

Rendimiento de la escena Mejoramiento

Ahora que hemos creado y manejado escenas, es hora de centrar nuestra atención en la optimización del rendimiento real.

SOPORTE DE LA INTERFAZ DE PROGRAMACIÓN DE APLICACIONES (API) PARA GRÁFICOS

Unity es compatible con las API de gráficos DirectX, Metal, OpenGL y Vulkan, según la disponibilidad de la API en una plataforma determinada. Unity emplea un conjunto preinstalado de API de gráficos o las API de gráficos que especifique en el Editor.

Para utilizar las API de gráficos predeterminadas de Unity, siga estos pasos:

- Vaya a la configuración del reproductor (Editar > Configuración del proyecto, luego elija la categoría de jugador).
- Navegue a Otras configuraciones y marque Gráficos automáticos cuadro API.

DOI: 10.1201/9781003214755-5

Cuando se selecciona la casilla Auto Graphics API para una plataforma, la compilación del reproductor contiene un conjunto de gráficos integrados API y utiliza la mejor en tiempo de ejecución para ofrecer lo mejor de los casos.

Cuando la API de gráficos automáticos para una plataforma no está seleccionada, el editor emplea la primera API de la lista.
Para observar cómo funciona nuestro programa en OpenGL en el Editor, por ejemplo, arrastre OpenGLCore a la parte superior de la lista y el Editor cambia a la representación de OpenGL.

Desmarque la API de gráficos automáticos aplicable, haga clic en el botón de adición (+), luego seleccione la API de gráficos en el cuadro desplegable para anular las API de gráficos predeterminadas para el editor y el reproductor.

La API predeterminada es la API de gráficos en la parte superior de la lista de API de Auto Graphics. Si la plataforma no es compatible con la API predeterminada, Unity recurre a la siguiente API en la lista de API de Auto Graphics.

Consulte Diferencias de representación específicas de la plataforma para obtener información sobre cómo la representación de gráficos difiere entre los sistemas y la semántica del lenguaje Shader. Solo una fracción de las API de gráficos permite sombreadores de geometría y teselado. El nivel de destino de la compilación de sombreadores gobierna esto.

DirectX

Vaya a la configuración del reproductor (menú: Editar > Configuración del proyecto, luego elija la categoría del reproductor) y seleccione DirectX11 como nuestra API de gráficos elegida en el editor o el reproductor independiente. Deshabilite la configuración de Auto Graphics API para Windows y seleccione DirectX11 en el menú desplegable.

Shaders para la superficie

Debido a que algunas secciones de la canalización de compilación de Surface Shader no comprenden la sintaxis HLSL específica de DX11,

debemos envolverlo en una macro de preprocesador solo DX11 si usamos funciones HLSL como StructuredBuffers, RWTextures y otros vocabularios que no son DX9.

Sombreadores de geometría y teselación

Los sombreadores de superficie admiten teselado y desplazamiento simples.

Podemos utilizar toda la gama de capacidades del modelo 5.0 de DX11 Shader cuando construimos manualmente programas Shader, incluidos Geometry, Hull y Domain Shaders.

Solo una fracción de las API de gráficos permite sombreadores de geometría y teselado. El nivel de destino de la compilación de sombreadores gobierna esto.

Sombreadores calculados

Compute Shaders son programas basados en tarjetas gráficas que pueden acelerar el renderizado.

Metal

Metal es la API de gráficos estándar de la industria de Apple. Unity funciona con Metal en iOS, tvOS y macOS (Independiente y Editor).

En los sistemas Apple, Metal proporciona más funciones que OpenGL ES.

El metal tiene las siguientes ventajas:

- Reducir la sobrecarga de la CPU asociada con las solicitudes de la API de gráficos.
- La capa de validación de la API.
- En sistemas de unidades de procesamiento de múltiples gráficos (GPU), mejor control de la GPU.

- En iOS/tvOS, se admiten destinos de procesamiento sin memoria.
- Apple ha establecido una nueva norma.
- Shaders para ordenadores.
- Shaders para teselado.

Los inconvenientes de usar Metal:

- No hay soporte para dispositivos de gama baja.
- Los sombreadores de geometría no son compatibles.

Restricciones y Requisitos

- La compatibilidad con Metal está disponible en iOS y tvOS para Apple A7 o SoC más nuevos.
- La compatibilidad con Metal está disponible en macOS para Intel HD e Iris Graphics de la serie HD 4000 o posterior, GPU basadas en AMD GCN y GPU basadas en Nvidia Kepler o posterior.
- El objetivo mínimo de compilación de shaders es 3.5.
- Metal no admite sombreadores de geometría.

Habilitación de metales

Para configurar Metal como la API de gráficos predeterminada para Unity Editor y Standalone Player, realice una de las siguientes acciones:

- Vaya al menú Edición > Configuración del proyecto en el Editor, elija la categoría Reproductor y active la compatibilidad con el Editor de metal.

- Alternativamente, si usamos macOS, inicie Terminal y use el parámetro de línea de comando -force-metal.
- En los reproductores independientes de iOS, tvOS y macOS, Metal está habilitado de forma predeterminada.

Validación de API de metal

Xcode proporciona la validación de API de metal y se puede usar para rastrear errores oscuros. Para habilitar la validación de la API de Metal en Xcode, siga estos pasos:

- Crear un proyecto de iOS en Unity. Esto produce un proyecto Xcode.
- En Xcode, abra el proyecto Xcode producido y seleccione Editar esquema.

Núcleo OpenGL

OpenGL Core es un backend que puede manejar las capacidades más recientes de OpenGL en Windows, macOS X y Linux. Dependiendo de la compatibilidad del controlador OpenGL, esto varía de OpenGL 3.2 a OpenGL 4.5.

Activando el núcleo de OpenGL

Vaya a la configuración del reproductor (menú: Editar > Configuración del proyecto, luego elija la categoría del reproductor) y navegue a Otras configuraciones para designar OpenGL Core como su API de gráficos preferida en el Editor o el Reproductor independiente. Deshabilite la propiedad Auto Graphics API para Windows y seleccione OpenGLCore en el menú desplegable.

Especificaciones de OpenGL

Los siguientes son los requisitos mínimos para OpenGL
Centro:

- Mac OS X 10.8 (OpenGL 3.2), Mac OS X 10.9.
  (OpenGL 3.2 a 4.1).
- Windows ha usado GPU NVIDIA desde 2006 (GeForce 8), GPU
  AMD desde 2006 (Radeon HD 2000) y GPU Intel desde 2012 (HD
  4000/IvyBridge)
  (OpenGL 3.2 a OpenGL 4.5).
- GNU/Linux (OpenGL 3.2 a OpenGL 4.5).

Limitaciones del controlador OpenGL de macOS

Las capacidades de OpenGL 3.x y 4.x, como la teselación y los sombreadores de geometría, son compatibles con el backend de macOS OpenGL para el Editor y Standalone.

Sin embargo, Apple limita la versión de OpenGL en el escritorio de Mac OS X a 4.1 como máximo. No es compatible con todas las capacidades de DirectX 11 (como vistas de acceso desordenado o sombreadores de cómputo). Esto implica que cualquier sombreador configurado para apuntar a Shader Level 5.0 (por #pragma target 50) no se cargará en OS X.

Como resultado, se agrega un nuevo nivel de destino de sombreador: #pragma target gl4.1. Este nivel objetivo requiere al menos OpenGL 4.1 o DirectX 11.0 Shader Level 5 en una computadora de escritorio o OpenGL ES 3.1 + Android Extension Pack en un dispositivo móvil.

Características de OpenGL Core

El nuevo back-end de OpenGL agrega una gran cantidad de nuevas funciones (anteriormente limitadas a DX11/GLES3):

- Shaders que calculan (junto con ComputeBuffers y texturas de renderizado de "escritura aleatoria").
- Shaders para teselado y geometría.
- Dibujo indirecto (Graphics.DrawProcedural. Graphics.DrawProceduralIndirect).
- Modos de la mezcla avanzada.

Parámetros de línea de comandos para el perfil principal de OpenGL

Las siguientes opciones de línea de comandos se pueden usar para iniciar el editor o el reproductor usando OpenGL:

- **-force-opengl:** fuerza el uso del OpenGL heredado back-end
- **-force-glcore:** Fuerza el uso del back-end OpenGL más reciente. Con este parámetro, Unity identificará todas las funciones que admite la plataforma para ejecutarse con la mejor versión de OpenGL y todos los OpenGL accesibles. extensiones
- **-force-glcoreXY:** el valor de XY puede ser 32, 33, 40, 41, 42, 43, 44 o 45, y cada número representa una versión diferente de OpenGL. Si la plataforma no es compatible con una versión determinada de OpenGL, Unity recurrirá a una versión compatible.
- **-force-clamped:** solicita que Unity no emplee extensiones OpenGL, lo que garantiza que se ejecute la misma ruta de código en diferentes sistemas. Este es un método para determinar si un problema es específico de la plataforma (un error de controlador, por ejemplo).

Parámetros nativos de la línea de comandos
de OpenGL ES en el escritorio

Se puede acceder a la API de gráficos OpenGL ES en computadoras Windows con GPU Intel o NVIDIA con controladores OpenGL ES.

- **-force-gles:** Obliga a que el nuevo backend de OpenGL se use en el modo OpenGL ES. Con este parámetro, Unity identificará todas las funciones que admite la plataforma para ejecutarse con la mejor versión de OpenGL ES y todas las extensiones de OpenGL ES accesibles.
- **-force-glesXY:** el valor de XY puede ser 20, 30, 31, 31aep o 3.2, donde cada número significa una versión diferente de OpenGL ES. Si la plataforma no es compatible con una versión determinada de OpenGL ES, Unity recurrirá a una versión compatible. Unity utilizará una API de gráficos alternativa si la plataforma no es compatible con OpenGL ES.
- **-force-clamped:** solicita que Unity no emplee extensiones OpenGL, lo que garantiza que se ejecute la misma ruta de código en diferentes sistemas. Este es un método para determinar si un problema es específico de la plataforma (un error de controlador, por ejemplo).

OPTIMIZACIÓN DEL RENDIMIENTO DE GRÁFICOS

Muchos juegos dependen de un buen desempeño para tener éxito. Aquí hay algunos consejos esenciales para aumentar la velocidad de renderizado de nuestro juego.

Encuentre gráficos de alto impacto

Los elementos gráficos de nuestro juego pueden tener una influencia significativa en dos sistemas informáticos: la GPU y

la CPU Debido a que las tácticas para optimizar GPU versus CPU son generalmente diferentes (e incluso pueden ser contrarias; por ejemplo, es bastante común hacer que la GPU trabaje más cuando se optimiza para CPU y viceversa), la primera regla de cualquier optimización es descubrir dónde está el problema de rendimiento.

Cuellos de botella comunes y cómo detectarlos:

- El rendimiento de la GPU suele verse limitado por el relleno velocidad o ancho de banda de la memoria.
  - Reduzca la resolución de la pantalla e inicie el juego. Si reducir la resolución de la pantalla hace que el juego se ejecute más rápido, es posible que esté limitado por la velocidad de llenado de la GPU.
- La CPU está frecuentemente restringida por el número de lotes que deben ser renderizados.
  - En el panel Estadísticas de procesamiento, seleccione "lotes". Cuanto mayor sea el número de lotes procesados, mayor será el costo para la CPU.

Los cuellos de botella menos comunes incluyen:

- Hay demasiados vértices para procesar en la GPU.
  La GPU y la complejidad de los sombreadores de vértices determinan el número de vértices adecuados para un rendimiento excelente. En general, esfuércese por no más de 100,000 vértices en dispositivos móviles. Aunque una PC puede manejar varios millones de vértices, es mejor mantener este número lo más bajo posible a través de la optimización.

- Hay demasiados vértices para que los maneje la CPU. Las mallas con piel, la simulación de telas, las partículas y otros elementos y mallas del juego pueden beneficiarse de esto. Como se indicó anteriormente, la mejor práctica suele ser mantener este número lo más bajo posible sin sacrificar la calidad del juego.
- Si el renderizado no es un problema en la GPU o la CPU, puede haber una falla en alguna parte, como en nuestro script o en la física. Para identificar el problema, use Unity Profiler.

Mejora de la CPU

Para representar cosas en la pantalla, la CPU debe realizar un trabajo de procesamiento considerable, como detectar si las luces afectan a ese objeto, establecer los parámetros del sombreador y del sombreador, emitir órdenes de dibujo al controlador de gráficos y preparar los comandos para la transmisión de la tarjeta gráfica.

Toda esta utilización de CPU "por objeto" consume muchos recursos y puede acumularse si tiene muchos elementos visibles. Por ejemplo, si tenemos mil triángulos, es mucho más fácil para la CPU, y todos están en una malla en lugar de una malla por triángulo. El costo de ambos casos en la GPU es más o menos comparable, pero el esfuerzo de la CPU requerido para renderizar mil elementos es mucho mayor.

Reducir el número de objetos visibles. Para minimizar la cantidad de trabajo que debe realizar la CPU:

- Combine elementos cercanos de forma manual o con la ayuda del procesamiento por lotes de llamadas de sorteo de Unity.
- Al combinar texturas individuales en un atlas de texturas más destacado, puede usar menos materiales en sus modelos.

- Reduzca la cantidad de elementos que hacen que los objetos se muestren varias veces (como reflejos, sombras y luces por píxel).

Combine elementos de modo que cada malla tenga al menos unos cientos de triángulos, y solo se use un Material para toda la malla. Vale la pena señalar que unir dos cosas que no comparten una sustancia produce un aumento en el rendimiento. La causa más común para requerir múltiples materiales es que dos modelos no comparten las mismas texturas, asegúrese de que cualquier objeto que combine tenga las mismas texturas para mejorar la eficiencia de la CPU.

La combinación de objetos puede no tener sentido al utilizar muchas luces de píxeles en el proceso de renderizado Adelante.

OnDemandRendering Optimización de CPU

OnDemandRendering puede ayudarlo a mejorar el rendimiento de la CPU al permitirle cambiar el ritmo de procesamiento de su aplicación.

En los siguientes casos, es posible que queramos reducir el marco Velocidad:

- **Menús, como el iniciador de la aplicación o un menú de pausa: los** menús suelen ser secuencias primarias que no requieren renderización a toda velocidad. Para ahorrar energía y evitar que la temperatura del dispositivo suba hasta el punto en que se estrangule la frecuencia de la CPU, podemos dibujar menús a una velocidad de fotogramas más baja.

- **Ajedrez y otros juegos por turnos:** los jugadores esperan a que otros usuarios se muevan o piensan en su movimiento. Durante los momentos de baja actividad, podemos reducir

la velocidad de fotogramas para evitar desperdiciar energía y prolongar la duración de la batería.

- Aplicaciones con material mayormente estático, como la interfaz de usuario (UI) automotriz.

Ajustar la velocidad de renderizado le permite regular el consumo de energía y la temperatura del dispositivo para maximizar la duración de la batería y evitar la aceleración de la CPU. Es especialmente efectivo cuando se usa con el paquete Adaptive Performance. Incluso cuando los marcos se muestran con menos frecuencia, el programa continúa entregando eventos a los scripts a una velocidad estándar (por ejemplo, puede aceptar entradas durante un marco no procesado). Podemos usar OnDemandRendering para evitar la latencia de entrada. render FrameInterval = 1 para la longitud de entrada para mantener la capacidad de respuesta de los movimientos, botones, etc.

Esta API no es útil para situaciones que requieren mucha programación, física, animación, pero no renderizado.
Las imágenes en nuestro programa pueden detenerse sin afectar la utilización de energía.

Las aplicaciones de realidad virtual no admiten el renderizado bajo demanda. Cuando no se procesan todos los fotogramas, los gráficos se desincronizan con el movimiento de la cabeza, lo que puede aumentar el riesgo de mareos.

GPU: optimización de la geometría del modelo
Hay dos reglas principales para maximizar la geometría de un modelo:

1. No use más triángulos de los necesarios.
2. Reduzca la cantidad de costuras de mapeo ultravioleta (UV) y bordes ásperos.

Es crucial tener en cuenta que el número real de vértices que debe procesar el hardware de gráficos no siempre es el mismo que el número presentado por una aplicación tridimensional (3D). El software de modelado a menudo muestra el número de puntos de esquina diferentes que componen un modelo (conocido como el número de vértices geométricos). Sin embargo, algunos vértices geométricos deben dividirse en dos o más vértices lógicos por razones de representación en una tarjeta gráfica.

Si un vértice tiene varias normales, coordenadas UV o colores de vértice, debe dividirse. Como resultado, el recuento de vértices en Unity suele ser mayor que el recuento proporcionado por la aplicación 3D.

Si bien la cantidad de geometría en los modelos es más significativa para la GPU, varios Unity también cuentan con modelos de proceso en la CPU (por ejemplo, desollado de malla).

Eficiencia de iluminación

El método más rápido es producir iluminación que no requiera ningún cálculo. Para hacer esto, utilice Lightmapping para "hornear" la iluminación estática una vez en lugar de calcularla por cuadro. El proceso de creación de un entorno de mapa de luz en Unity lleva solo un poco más de tiempo que simplemente agregar una luz en el área, pero:

- Es significativamente más rápido (dos o tres veces más rápido para dos luces por píxel).
- Se ve mucho mejor ahora que puede hornear la iluminación global y usar el mapeador de luz para suavizar los resultados.

En muchas circunstancias, se pueden usar enfoques simples en lugar de instalar varias luces más. en lugar de agregar

una luz que brilla directamente en la cámara para crear un efecto de Rim Lighting, incluye un cálculo especializado de Rim Lighting directamente en nuestros shaders (ver Ejemplos de Surface Shader para aprender cómo hacer esto).

Dibujo delantero de luces

La iluminación dinámica por píxel aumenta la cantidad de esfuerzo de renderizado requerido para cada píxel afectado, lo que da como resultado que los objetos se dibujen en numerosas pasadas. En dispositivos menos potentes, como GPU móviles o de PC de gama baja, evite tener más de un Pixel Light iluminando cualquier elemento y, en su lugar, utilice mapas de luz para iluminar objetos estáticos en lugar de calcular su iluminación en cada fotograma. Debido a que la iluminación dinámica por vértice puede agregar una gran cantidad de esfuerzo a las transformaciones de vértice, evite los casos en los que varias luces iluminan un solo objeto.

Combinar modelos que están lo suficientemente lejos como para verse influenciados por varios conjuntos de iluminación de píxeles no es una buena idea. Cada malla debe renderizarse tantas veces como el número de luces de píxeles que la iluminan cuando se usa la iluminación de píxeles. Cuando se unen dos mallas que están muy separadas, el tamaño efectivo del objeto resultante crece. Dado que todas las luces de píxeles que iluminan cualquier área de este objeto compuesto se tienen en cuenta durante la renderización, es posible que aumente el número de pases de renderización necesarios. En general, el número de pasadas necesarias para representar el elemento combinado es igual al número total de pasadas necesarias para dibujar cada objeto componente; por lo tanto, fusionar mallas no produce ningún beneficio.

Durante el renderizado, Unity detecta todas las luces en las inmediaciones de una malla y determina cuál de esas luces es más

influenciarlo. Los parámetros de la ventana Calidad controlan cuántas luces son luces de píxeles y cuántas son luces de vértice.
Cada luz calcula su valor dependiendo de qué tan lejos esté de la malla y qué tan brillante sea su iluminación—
y ciertas luces son más esenciales que otras simplemente según el escenario del juego. Como resultado, cada luz tiene una configuración de modo de procesamiento que se puede configurar como importante o no importante; las luces identificadas como No importante tienen una sobrecarga de representación más pequeña.

Considere un juego de conducción en el que el automóvil del jugador conduce en la oscuridad con los faros encendidos.
Debido a que los faros son la fuente de luz más perceptible visualmente en el juego, su Modo de procesamiento debe establecerse en Importante. Otras luces en el juego, como las luces traseras de otros autos o farolas distantes, pueden ser menos relevantes y pueden no aumentar mucho la impresión visual al convertirse en luces de píxeles. Para evitar desperdiciar poder de renderizado en regiones donde no es efectivo, establezca el Modo de renderizado para dichas luces en No importante.

La optimización de iluminación por píxel ahorra trabajo de CPU y GPU: la CPU tiene menos llamadas de dibujo y la GPU tiene menos vértices para calcular y píxeles para rasterizar para todas las representaciones de objetos adicionales.

Compresión de texturas y Mipmaps en la GPU

Las texturas comprimidas se pueden utilizar para reducir el tamaño de sus texturas. Esto puede conducir a tiempos de carga más rápidos, una huella de memoria reducida y una velocidad de renderizado muy mejorada. Las texturas RGBA de 32 bits sin comprimir necesitan una fracción del ancho de banda de memoria que requieren las texturas comprimidas.

Mapas MIP para Texturas

Active siempre Crear mipmaps para texturas en un entorno 3D. Para triángulos pequeños, una textura mipmap permite que la GPU use una textura de menor resolución. Esto es análogo a cómo la compresión de texturas puede ayudar a limitar la cantidad de datos de textura transmitidos por la GPU al renderizar.

La única excepción es cuando se sabe que un texel (píxel de textura) se asigna 1:1 a un píxel de pantalla visible, como en los componentes de la interfaz de usuario o en un juego bidimensional (2D).

LOD y distancias de eliminación por capa

Sacrificar elementos implica hacerlos invisibles. Este es un método eficiente para reducir las cargas de CPU y GPU.

En muchos juegos, sacrificar elementos pequeños de manera más agresiva que los grandes es una técnica rápida y efectiva sin sacrificar la experiencia del usuario. Los pequeños guijarros y la basura, por ejemplo, pueden volverse invisibles desde una gran distancia, mientras que los edificios masivos permanecen visibles.

Hay varias técnicas que podríamos usar para hacer esto:

- Hacer uso del mecanismo de Nivel de Detalle.
- Establezca manualmente las distancias de eliminación selectiva por capa de la cámara.
- Coloque elementos pequeños en una capa separada y use la función de secuencia de comandos Camera.layerCullDistances para crear distancias de eliminación por capa.

Sombras en tiempo real

Las sombras en tiempo real son encantadoras, pero pueden afectar significativamente la velocidad, tanto en términos de más llamadas de extracción de CPU como de procesamiento adicional de GPU.

GPU: pautas para crear sombreadores de alto rendimiento

Las capacidades de rendimiento varían significativamente entre plataformas; una GPU de PC de gama alta puede manejar muchos más gráficos y sombreadores que una GPU móvil de gama baja. Una GPU rápida es decenas de veces más rápida que una GPU mal integrada, incluso en la misma plataforma.

Es casi seguro que la eficiencia de GPU en plataformas móviles y PC de gama baja será significativamente menor que en su computadora de desarrollo. Para obtener un rendimiento decente en los dispositivos de GPU de gama baja, se recomienda ajustar manualmente los sombreadores para reducir los cálculos y las lecturas de texturas. Algunos sombreadores integrados de Unity, por ejemplo, tienen versiones "móviles" que son sustancialmente más rápidas pero tienen algunos límites o aproximaciones.

Operaciones matemáticas complejas Las funciones matemáticas trascendentales (como pow, exp, log, cos, sin, tan) consumen muchos recursos, por lo que los utilizan con moderación. Si corresponde, considere emplear texturas de búsqueda como una alternativa a las operaciones matemáticas complejas.

Evitar el desarrollo de nuestros negocios (como normalizar, punto, inversesqrt). Las características integradas en Unity aseguran que el controlador genere un código significativamente mejor. Tenga en cuenta que la acción Alpha Test (descartar) con frecuencia ralentiza nuestro sombreador de fragmentos.

La precisión de los puntos flotantes Si bien la precisión de las variables de punto flotante (flotante frente a medio frente a fijo) a menudo se pasa por alto en las GPU de escritorio, es fundamental para el rendimiento de la GPU móvil.

Una lista de verificación simple para ayudarnos a mejorar
La velocidad de nuestro juego

- Al crear para PC, mantenga el conteo de vértices alrededor de 200K y el conteo de fotogramas por debajo de 3M. (dependiendo de la GPU de destino).
- Elija shaders de la categoría Mobile o Unlit si estamos utilizando shaders integrados. También son compatibles con sistemas no móviles; sin embargo, son contrapartes reducidas y aproximadas de los sombreadores más complicados.
- Mantenga la menor cantidad posible de materiales distintos por escena y comparta tantos materiales como sea posible entre varios elementos.
- Para habilitar optimizaciones internas, como el procesamiento por lotes estático, configure el atributo Estático en un objeto que no se mueva.
- En lugar de múltiples, use una sola luz de píxel (idealmente dirigida) para afectar nuestra geometría.
- En lugar de utilizar iluminación dinámica, utilice iluminación para hornear.
- Cuando sea factible, elija formatos de textura comprimida y texturas de 16 bits en lugar de texturas de 32 bits.
- Cuando sea posible, evite emplear niebla.
- En escenas estáticas complicadas con muchas oclusiones, use Eliminación de oclusiones para limitar la cantidad de geometría visible y llamadas de dibujo. Considere eliminar la oclusión mientras crea nuestros niveles.
- Los skyboxes se pueden usar para “falsificar” geometría distante.

- En lugar de una técnica de varias pasadas, utilice sombreadores de píxeles o combinadores de texturas para combinar varias texturas.
- Cuando sea factible, utilice variables de precisión media.
- Reducir el uso de operaciones matemáticas complejas en los sombreadores de píxeles como pow, sen y cos.
- Reducir el número de texturas por fragmento.

Lotes de Sorteos

El motor debe enviar una llamada de dibujo a la API de gráficos para dibujar un GameObject en la pantalla (como OpenGL o Direct3D). Las llamadas de sorteo suelen consumir muchos recursos, y la API de gráficos realiza un trabajo sustancial para cada solicitud de sorteo, lo que genera una sobrecarga en el rendimiento de la CPU. Esto se debe principalmente a los cambios de estado realizados entre llamadas de sorteo (como cambiar a un nuevo material), que necesitan procesos de traducción y validación que consumen muchos recursos en el controlador de gráficos.

Para remediar esto, Unity emplea dos métodos:

1. Procesamiento por **lotes dinámico:** para mallas lo suficientemente pequeñas, esto cambia los vértices en la CPU, combina varios vértices similares y los dibuja todos a la vez.
2. **Dosificación estática:** agregados estáticos (sin movimiento) GameObjects en mallas grandes, que luego se procesan más rápidamente.

A diferencia de la fusión manual de GameObjects, el procesamiento por lotes integrado tiene varias ventajas; lo más importante es que los GameObjects aún pueden seleccionarse por separado.

Sin embargo, tiene ciertos inconvenientes; el procesamiento por lotes estático incurre en gastos de memoria y almacenamiento, mientras que el procesamiento por lotes dinámico incurre en cierta sobrecarga de la CPU.

Preparación de materiales para dosificación

Solo se pueden agrupar GameObjects con el mismo material. Como resultado, si desea lograr un procesamiento por lotes eficaz, intente distribuir los materiales entre tantos GameObjects distintos como sea posible.

Si tiene dos materiales similares que solo difieren en la textura, puede fusionar sus texturas en una sola textura grande. Un atlas de textura es un término usado para describir este procedimiento. Una vez que las Texturas están en el mismo atlas, se puede usar un solo Material en su lugar.

Solo se pueden agrupar GameObjects con el mismo material. Como resultado, si queremos lograr un procesamiento por lotes efectivo, intente distribuir los Materiales entre tantos GameObjects distintos como sea posible.

Si tenemos dos Materiales similares que solo difieren en la textura, podemos fusionar sus Texturas en una única Textura grande. Un atlas de textura es un término usado para describir este procedimiento. Una vez que las Texturas están en el mismo atlas, se puede usar un solo Material en su lugar.

Si necesitamos acceder a las propiedades del Material compartido desde los scripts, tenga en cuenta que cambiar Renderer.material produce un duplicado del Material. Para mantener los materiales compartidos, use Renderer.sharedMaterial en su lugar.

Aunque sus materiales son diferentes, los creadores de sombras normalmente se pueden agrupar durante el renderizado. En Unity, los creadores de sombras pueden emplear el procesamiento por lotes dinámico con varios materiales, siempre que los valores en los materiales

requeridas por el pase de sombra son las mismas. Muchas cajas, por ejemplo, pueden emplear Materiales con diferentes Texturas, pero las texturas son irrelevantes para dibujar sombras; por lo tanto, se pueden agrupar en lotes en este escenario.

Dosificación dinámica

Si dos GameObjects tienen el mismo Material y cumplen requisitos adicionales, Unity puede moverlos automáticamente por lotes a la misma llamada de sorteo. El procesamiento por lotes dinámico ocurre automáticamente y no requiere más esfuerzo de nuestra parte.

- Debido a que el procesamiento por lotes de GameObjects dinámicos genera algunos gastos por vértice, se limita a mallas con no más de 900 atributos de vértice y no más de 300 vértices.
    - Podemos agrupar hasta 300 verts si nuestro Shader usa Posición de vértice, Normal y UV simple, pero solo 180 verts si nuestro Shader usa Posición de vértice, Normal, UV0, UV1 y Tangente.
- Los GameObjects no se pueden agrupar si la transformación contiene duplicación (por ejemplo, GameObject A con escala +1 y GameObject B con escala -1 no se pueden agrupar juntos).
- El uso de instancias de Material separadas evita que los GameObjects se agrupen, incluso si son idénticos. La representación del lanzador de sombras es una excepción.
- Los GameObjects habilitados para mapas de luz tienen dos nuevos parámetros de renderizado: índice de mapa de luz y desplazamiento/escala

en el mapa de luz. En general, todos los GameObjects con mapas de luz dinámicos deben apuntar a la ubicación exacta del mapa de luz que se va a agrupar.

• Los sombreadores de varias pasadas interrumpen el procesamiento por lotes.

- Casi todos los Unity Shaders permiten múltiples luces en el renderizado de reenvío, esencialmente realizando pases adicionales para ellos. Las “luces extra por píxel” solicitadas en el sorteo no se agrupan.
- Debido a que debe dibujar GameObjects dos veces, el enfoque de renderizado heredado diferido (paso previo ligero) deshabilita el procesamiento por lotes dinámico.

Debido a que el procesamiento por lotes dinámico funciona al traducir todos los vértices de GameObject al espacio mundial en la CPU, solo es útil si el esfuerzo es menor que el de una llamada de sorteo. Las necesidades de recursos de una llamada de sorteo están determinadas por varios factores, el más importante de los cuales es la API de gráficos utilizada. Por ejemplo, en las consolas o las API contemporáneas como Apple Metal, el costo de la llamada de sorteo suele ser sustancialmente menor y el procesamiento por lotes dinámico suele ser ineficaz.

Lotes dinámicos (sistemas de partículas, renderizadores de líneas, renderizadores de senderos)

El procesamiento por lotes dinámico funciona de manera diferente para los componentes con geometría generada dinámicamente por Unity que para las mallas.

- Unity fusiona todo el material procesable por lotes para cada tipo de renderizador adecuado en un solo gran búfer de vértices.
- El renderizador crea el estado Material del lote.

- El búfer de vértices está vinculado al dispositivo gráfico por Unidad.
- Unity cambia el desplazamiento en el búfer de vértices para cada Renderer en el lote antes de enviar una nueva solicitud de dibujo.

Al calcular el costo de las llamadas del dispositivo gráfico, la configuración del estado del material es el componente más lento del dibujo de un componente. En comparación, enviar llamadas de sorteo en múltiples compensaciones en un búfer de vértices estándar es extremadamente rápido.

Este método es bastante similar a cómo Unity envía llamadas de sorteo cuando se usa el procesamiento por lotes estático.

Dosificación que es estática

El procesamiento por lotes estático permite que el motor disminuya las solicitudes de dibujo para la geometría de cualquier tamaño, siempre que esté hecha del mismo material y no se mueva. Con frecuencia es más eficiente que el procesamiento por lotes dinámico (ya que no convierte vértices en la CPU), pero consume más memoria.

Para usar el procesamiento por lotes estático, debemos definir explícitamente que los GameObjects particulares son estáticos y no se mueven, rotan ni escalan en el juego. Para ello, utilice la casilla de verificación Estático del Inspector para marcar los GameObjects como estáticos.

El procesamiento por lotes estático requiere el uso de memoria adicional para almacenar la geometría combinada. Si varios GameObjects compartían la misma geometría antes del procesamiento por lotes estático, se crea un duplicado de esa geometría para cada GameObject, ya sea en el Editor o en tiempo de ejecución. Esto no siempre es una idea inteligente; para preservar una huella de memoria más baja, es posible que tengamos que cambiar el rendimiento de representación eliminando el procesamiento por lotes estático para

algunos GameObjects. Por ejemplo, etiquetar árboles como estáticos en un nivel de bosque denso podría tener una influencia significativa en la memoria.

El procesamiento por lotes estático funciona internamente al convertir GameObjects estáticos en espacio mundial y crear un único vértice compartido y un búfer de índice para ellos. Si activamos Optimized Mesh__ Data__ (en la configuración del reproductor), Unity elimina los elementos de vértice que no son utilizados por ninguna variación de shader mientras se construye el búfer de vértice. Para hacer esto, se utilizan ciertas verificaciones de palabras clave específicas; por ejemplo, si Unity no detecta la palabra clave LIGHTMAP ON, elimina los UV de mapas de luz de un lote. Luego, para los GameObjects visibles en el mismo lote, Unity ejecuta una serie de llamadas de dibujo básicas esencialmente sin cambios de estado en el medio.

Técnicamente, Unity no conserva las llamadas de dibujo de la API, sino los cambios de estado entre ellas (que es la parte que consume muchos recursos). En la mayoría de los sistemas, las limitaciones de lote son 64k vértices y 64k índices (48k índices en OpenGLES, 32k índices en macOS).

Instanciación de GPU

Con un número limitado de llamadas de dibujo, use instancias de GPU para dibujar (o renderizar) varias copias de la misma malla simultáneamente. Esto es útil para esbozar elementos que se repiten regularmente en una escena, como casas, árboles, césped u otras cosas.

Cada llamada de sorteo simplemente produce mallas idénticas, pero cada instancia puede tener varios parámetros (por ejemplo, color o tamaño) para agregar diversidad y disminuir la percepción de repetición.

La creación de instancias de GPU tiene el potencial de minimizar la cantidad de llamadas de sorteo utilizadas por escena. Esto aumenta drásticamente el rendimiento de renderizado de nuestro proyecto.

Incluir instancias en nuestros materiales

Para habilitar instancias de GPU en materiales, elija nuestro material en la ventana del proyecto y haga clic en la opción Habilitar instancias en el Inspector.

Esta casilla de verificación aparece en Unity solo si Material Shader admite instancias de GPU. Esto cubre todos los Shaders de superficie, así como Standard, StandardSpecular y StandardSpecular.

La creación de instancias de GPU está habilitada, aunque no está en la imagen inferior. Tome nota de las diferencias en FPS, lotes y tiempo ahorrado por lotes.

Las siguientes restricciones se aplican al usar instancias de GPU:

- Unity selecciona automáticamente los componentes MeshRenderer y Graphics. La creación de instancias es requerida por DrawMesh. Cabe señalar que Desollado MeshRenderer no es compatible.
- En una sola llamada de sorteo de creación de instancias de GPU, Unity solo agrupa GameObjects con la misma malla y material. Para mejorar la eficiencia de creación de instancias, use una cantidad limitada de mallas y materiales. Modifique nuestros scripts de sombreado para incluir datos por instancia para generar variaciones.

Visualización de la ventana de estadísticas

La vista del juego incluye un cuadro de estadísticas que muestra información de representación en tiempo real sobre su programa durante el modo de reproducción. Seleccione el botón Estadísticas en la esquina superior derecha para abrir esta ventana. La ventana aparece como una superposición en la esquina superior derecha de la vista del juego. Sus estadísticas son esenciales para optimizar el rendimiento. Las estadísticas específicas accesibles dependen del objetivo de compilación.

Estadísticas

Estadísticas	Descripción
FPS	Actualizaciones al cuadro por segundo, realiza Unity.
UPC	La cantidad total de tiempo necesario para procesar un fotograma. Esto incluye el tiempo que Unity tardó en ejecutar la actualización de fotogramas de nuestra aplicación y el tiempo que Unity pasó en el Editor actualizando la vista de escena, otras ventanas del Editor u otras operaciones exclusivas del Editor. Tiempo de renderizado: el número de veces que se tarda en renderizar un cuadro. Esta cifra incluye el tiempo que Unity tardó en renderizar la Vista del juego, pero no el tiempo que Unity usó en el Editor para renderizar la Vista de escena o crear el Inspector.
lotes	El número total de lotes procesados por Unity durante un marco. Esta figura contiene lotes estáticos y dinámicos, así como, por ejemplo, lotes.
Guardado por lotes	El número total de lotes creados por Unity. Comparta materiales entre distintos GameObjects con la frecuencia que sea posible para garantizar un procesamiento por lotes óptimo. Cambiar el estado de representación divide los lotes en grupos que tienen el mismo estado.
tris	La cantidad de triángulos procesados por Unity durante un cuadro. Esto es especialmente cierto cuando se optimiza para hardware de gama baja.
Verduras	El número de vértices procesados por Unity durante un cuadro. Esto es especialmente cierto cuando se optimiza para hardware de gama baja.
Pantalla	La resolución de la pantalla, así como la cantidad de memoria RAM que emplea.

(Continuado)

Estadísticas	**Descripción**
Establecer Pase	La cantidad de veces que Unity cambia entre pases de sombreador al renderizar GameObjects durante un cuadro. Un sombreador puede tener varios pases de sombreado, cada uno de los cuales representa GameObjects en la escena de manera diferente. Cada pase requiere la vinculación de un nuevo sombreador, lo que puede resultar en un costo de CPU.
Lanzadores de sombras	El número de GameObjects en el cuadro que arrojar sombras.
Mallas de piel visible	Unity representó la cantidad de renderizadores de malla con piel en el cuadro.
animaciones	La cantidad de animaciones que están activas. en todo el marco.

Depurador para marcos

El Frame Debugger nos permite detener un juego en ejecución en un marco específico e inspeccionar las llamadas de sorteo individuales utilizadas para crear ese marco. Además de identificar las llamadas de sorteo, el depurador nos permite recorrerlas una a la vez, lo que nos permite examinar con gran detalle cómo se construye la Escena a partir de sus piezas gráficas.

Hacer uso del depurador de marcos

ventana de Frame Debugger (menú: Ventana > Análisis > Frame Debugger) muestra información de llamada de dibujo y nos permite cambiar la "reproducción" del marco a medida que se crea.

La lista principal muestra la serie de llamadas de dibujo (y otros eventos como la limpieza del búfer de fotogramas) en una jerarquía que especifica dónde se originaron. Se muestra más información sobre la llamada de dibujo en el panel a la derecha de la lista, incluidos los datos de geometría y el sombreador utilizado para renderizar.

Cuando elegimos un elemento de la lista, la escena, incluida la llamada al sorteo, se presenta en su totalidad. La izquierda y la derecha

Los botones de flecha en la barra de herramientas y las teclas de flecha avanzan y retroceden en la lista en un solo paso. Además, el control deslizante en la parte superior de la ventana nos permite "barrer" rápidamente a través de las llamadas de sorteo para encontrar un artículo de interés. Cuando una llamada de sorteo corresponde a la geometría de un GameObject, el elemento se resalta en la ventana principal de Jerarquía para facilitar la identificación.

Si la llamada de dibujo seleccionada se representa en una RenderTexture, el contenido de esa RenderTexture se muestra en la vista del juego. Esto es útil para estudiar cómo se construyen los diferentes objetivos de representación fuera de la pantalla, como el búfer G difuso en el sombreado diferido:

El depurador de Remote Frames

Para usar Frame Debugger de forma remota, el reproductor debe ser compatible con la representación multiproceso (por ejemplo, WebGL no lo admite; por lo tanto, el depurador de marcos no puede operar en él), la mayoría de las plataformas de Unity lo habilitan y debe elegir la opción "Compilación de desarrollo" mientras creando

Nota para plataformas de escritorio: marque la opción "Ejecutar en segundo plano" antes de construir; de lo contrario, cuando conectamos Frame Debugger al reproductor, no reflejará ningún cambio de renderizado hasta que tenga el foco; suponiendo que esté ejecutando el Editor y el reproductor en la misma máquina, cuando controle Frame Debugger en el Editor, quitaremos el foco del reproductor.

Comience rápidamente:

- Compile el proyecto desde el editor hasta la plataforma de destino (seleccione el reproductor de desarrollo).
- El reproductor debe ejecutarse.

- Volver al Editor.
- Inicie la ventana del depurador de marcos.
- Representación de perfil activo, animación o en nuestro juego lógica.
- Cuando hace clic en Habilitar, el depurador de cuadros debe estar habilitado en el reproductor.

Opciones para visualización de destino de renderizado

Una barra de herramientas en la parte superior del panel de información nos permite aislar los canales rojo, verde, azul y alfa para el estado actual de la vista del juego. De manera similar, usando el control deslizante Niveles a la derecha de estos botones de canal, podemos separar secciones de la pantalla según los niveles de brillo. Estas opciones solo están disponibles cuando se renderiza en una RenderTexture.

Al renderizar en varios objetivos de renderizado simultáneamente, podemos elegir cuál mostrar en la vista del juego. Las memorias intermedias de iluminación difusa, especular, normal y de emisión/indirecta se muestran en el modo de sombreado diferido 5.0.

También podemos ver el contenido del búfer de profundidad seleccionando "Profundidad" en el menú desplegable.

Al aislar el canal alfa de la textura de renderizado, podemos observar la oclusión (almacenada en RT0 alfa) y la suavidad (almacenada en RT1 alfa) del búfer G retrasado.

La emisión de esta Escena y la iluminación ambiental/indirecta son bastante oscuras; podemos hacerlos más evidentes usando el control deslizante Niveles.

Streaming de Mapas Mip

Podemos controlar qué niveles de mipmap carga Unity en la memoria usando el mecanismo Mip Map Streaming. La unidad debe

dibuja la posición actual de la cámara en una escena en lugar de cargarla de forma predeterminada, ya que solo carga los mipmaps; esta técnica reduce la cantidad total de memoria que Unity requiere para las texturas. Sacrifica una pequeña cantidad de recursos de CPU por la posibilidad de ahorrar una cantidad sustancial de memoria GPU.

También podemos establecer un límite de memoria total para todas las Texturas en un Proyecto usando el Presupuesto de Memoria. Para mantenerse por debajo de este límite, el mecanismo Mip Map Streaming corta automáticamente los niveles del mapa mip.

La API de transmisión de mapas Mip se puede usar para solicitar niveles de mapas Mip particulares para texturas específicas. Unity incluye código C# de ejemplo que replica la lógica del motor para la selección del mapa mip, que podemos usar para modificar la lógica del motor en nuestros proyectos.

Mip Map Streaming ahorra entre un 25 % y un 30 % de RAM de textura en el proyecto de muestra Viking Village de Unity, según la ubicación de la cámara.

Para empezar

Para habilitar la transmisión de mapas Mip, navegue por la configuración de calidad en Unity (Editar > Configuración del proyecto > Calidad) y marque la casilla de verificación Transmisión de texturas. Esto muestra las opciones del sistema Mip Map Streaming.

Luego, para cada textura, active Mip Map Streaming para permitir que el sistema Mip Map Streaming transmita los mapas mip de cada textura desde el disco a la memoria. Para hacerlo, elija la textura a la que desea aplicar Mip Map Streaming, luego vaya a la ventana Inspector y observe la configuración de Importación de textura. Habilite la opción Streaming Mip Maps en la configuración avanzada.

Si estamos trabajando en Android, también debe ingresar a la configuración de compilación y cambiar el método de compresión a LZ4 o LZ4HC. Para la carga asíncrona de texturas, Unity requiere uno de estos algoritmos de compresión, en los que se basa el sistema Mip Map Streaming.

Unity carga mapas mip con la mejor resolución posible mientras se adhiere al presupuesto de memoria de textura. Utilice la API de C# para proporcionar niveles de mapa mip para cada textura para un control más preciso o ajustar la salida automatizada del sistema Mip Map Streaming.

Restricciones

Se puede indicar a Mipmap Streaming que calcule los niveles de mipmap requeridos utilizando una de las formas que se enumeran a continuación:

- Cada textura se asigna a un material, que posteriormente se asigna a un renderizador de Unity.
- Texture2D.requestedMipmapLevel se utiliza para solicitar niveles mip manualmente.

Unity no puede determinar qué nivel de mip utilizar si no le indicamos a Mipmap Streaming que genere niveles de mipmap utilizando una de estas formas. Como resultado, Unity carga la textura con mips de baja calidad que parecen borrosos.

Los sistemas enumerados a continuación no utilizan renderizadores convencionales. Esto implica que debemos configurar manualmente los mipmaps deseados para estos sistemas; de lo contrario, Unity utilizará texturas de baja resolución:

- Texturas para proyectores de calcas.
    - **Texturas del medidor de reflexión:** los mipmaps de menor resolución sirven como una tabla de búsqueda de rugosidad. Como un

Como resultado, si Unity elige un nivel de mipmap más bajo, renderiza los materiales con una rugosidad incorrecta.

- **Texturas en el sistema Terrain de Unity:**
  Mipmap Streaming en Terrain Textures no es compatible con Unity. Esto se debe a que las texturas del terreno deben estar disponibles con la máxima calidad en todo momento para que Unity coloque mosaicos y mezcle las texturas.

- Shaders que se comportan de manera diferente a los shaders integrados de Unity cuando se trata de coordenadas UV de textura.

  Unity siempre espera que las texturas se muestreen usando UV0 y se guarden en el Msh. Se ignoran todos los cambios basados en sombreadores en las coordenadas de la textura, con la excepción de la escala y la traducción, o el uso de UV1.

Cuando se ejecuta un renderizador, la malla que utiliza requiere métricas de distribución de UV precisas para determinar el nivel de mipmap requerido. Como parte del procedimiento de importación de mallas, Unity genera automáticamente métricas de dispersión. Esto también se puede calcular en un script usando Mesh.
Obtener Métrica de Distribución UV.

Cuando Unity muestra una textura de transmisión a través de una API (como Graphics.DrawMeshNow), el sistema carece de los límites del renderizador y otra información necesaria para determinar el nivel de mip; por lo tanto, debemos definir manualmente el nivel mip de la textura (o deshabilitar Mipmap Streaming en esta textura).

Solución de problemas de transmisión de Mipmap

Unity tiene un modo de vista de depuración de Mipmap Streaming.
Para acceder a él, seleccione Streaming de texturas en la vista Escena.

menú desplegable de control. Dependiendo de su estado en el sistema Mipmap Streaming, este modo de vista tiñe los GameObjects con los siguientes colores:

- Verde para texturas con mipmaps reducidos como resultado de la tecnología Mipmap Streaming.
- Las texturas con menos mipmaps se muestran en rojo porque el sistema Mipmap Streaming no tiene los recursos para cargarlas.
- Las texturas que no están configuradas para transmitir, o si no hay un renderizador que calcule los niveles de mip, se muestran en azul.

Con la API de depuración, también podemos crear nuestras propias herramientas y visualizaciones de depuración.

Capítulo 6

Completando el juego

A partir de ahora, hemos cubierto todos los conceptos básicos relacionados con Unity, desde la instalación y configuración hasta la gestión de escenas, la optimización y la física mundial.

Sin embargo, a veces nuestro código puede encontrarse con ciertos problemas. lemas Aquí es donde entra en juego la depuración.

DEPURACIÓN DE CÓDIGO C# EN UNITY

Podemos ver nuestro código fuente mientras nuestra aplicación o juego se está ejecutando mediante el uso de un depurador. Los siguientes editores de código están disponibles para depurar programas C# en Unity:

- Visual Studio (junto con el complemento Visual Studio Tools para Unity).

- Visual Studio para Mac es una herramienta de desarrollo de software.

DOI: 10.1201/9781003214755-6

- Piloto de Jetbrains.
- Código en Visual Studio.

Aunque las capacidades de depuración admitidas por estos editores de código difieren ampliamente, todos permiten funciones esenciales como puntos de interrupción, paso único y examen variable.

A excepción de WebGL, la depuración de código administrado en Unity funciona en todas las plataformas. Es compatible con los backends de secuencias de comandos Mono e IL2CPP.

Configuración del editor de código

- **En Visual Studio (Windows):** el instalador de Unity Editor ofrece la opción de instalar Visual Studio junto con el complemento de Visual Studio Tools para Unity. Este es el método recomendado para configurar Visual Studio para la depuración con Unity.

  Si ya tenemos Visual Studio instalado en nuestra computadora, vaya al menú Herramientas > Obtener herramientas y características… para encontrar e instalar el complemento de Visual Studio Tools para Unity.

- **En Visual Studio para Mac:** la instalación de Unity Editor ofrece la opción de instalar Visual Studio para Mac. Este es el método recomendado para instalar Visual Studio para Mac con el fin de depurar con Unity.

  Si ya tenemos Visual Studio para Mac instalado en nuestra computadora, utilice su Administrador de extensiones para buscar e instalar el complemento de Visual Studio Tools para Unity.

- **JetBrains Rider:** la instalación de JetBrains Rider de forma predeterminada puede depurar código en Unity en Windows o Mac. Para instalarlo, vaya al sitio web de JetBrains.

- **VS Code:** Para depurar código en Unity, debemos instalar la extensión correspondiente de VS Code. Por ejemplo, el Depurador de Unity: https://marketplace.visualstu dio.com/items?itemName=Unity.unity-debug o la extensión Unity Tools: https://marketplace.visualstu dio.com/items?itemName=Tobiah.unity-tools.

Elegir un editor de secuencias de comandos externo en Unity

Después de instalar un editor de código, vaya a Preferencias > Herramientas externas y cambie el Editor de secuencias de comandos externo a nuestro editor de código preferido.

Depuración del editor

Podemos depurar el código C# que se ejecuta en el editor de Unity mientras está en modo de reproducción.

Para depurar en el editor, cambie el modo de optimización de código del editor a modo de depuración, luego adjunte un editor de código con capacidades de depuración.

Para cambiar el modo de optimización de código, haga clic en el botón Depurar en la parte inferior derecha de la barra de estado del editor de Unity. esquina.

La opción de optimización de código en Unity Ofrece dos opciones

1. El modo de depuración, que nos permite adjuntar un software de depuración externo, reduce el rendimiento de C# cuando ejecutamos nuestro proyecto en el editor en modo de reproducción.

2. Cuando ejecutamos nuestro proyecto en modo de reproducción en el editor, obtenemos un rendimiento de C# más rápido, pero no podemos adjuntar ningún depurador externo.

Cuando hacemos clic en el botón Depurar en la barra de estado, aparece una pequeña ventana emergente con un botón de cambio de modo. También muestra información sobre el modo actual y explica lo que ocurre cuando cambiamos de modo.

Para modificar el modo en que se inicia Unity Editor, vaya a Edición > Preferencias > General > Optimización de código al inicio.

Utilice la siguiente API para controlar esta configuración desde un script: Compilation.Compilation, ManagedDebugger Compilation. CodeOptimization y Pipeline-codeOptimization.

También podemos alterar el modo de inicio del Editor o deshabilitar el depurador para escuchar el socket. Para lograr esto, ejecute el Editor con los siguientes argumentos de línea de comando:

- **-releaseCodeOptimization:** Habilita la optimización del código de publicación en el Editor.
- **-debugCodeOptimization:** Habilita la optimización del código de depuración en el Editor.
- **-disableManagedDebugger:** inicia el Editor sin que los depuradores escuchen el socket.

Adjuntar al editor y establecer puntos de interrupción

Establezca un punto de interrupción en nuestro editor de código externo en una línea de código de secuencia de comandos donde debe detenerse el depurador. Por ejemplo, en Visual Studio, haga clic en la columna a la izquierda de nuestro código en la línea donde queremos detener el depurador. Aparece una línea roja, resaltando la línea.

```
Clase pública ExampleScript: MonoBehaviour
{
Usar para la inicialización
```

```
Comienzo vacío ()
{
Debug.Log("Mensaje de depuración");
}
}
```

A continuación, conecte el editor de código al Editor de Unity. Esta opción varía según el editor de código y, con frecuencia, es distinta del procedimiento de depuración estándar del editor de código.

Algunos editores de código pueden permitirnos elegir una instancia de Unity para depurar. En Visual Studio, por ejemplo, la opción Depurar > Adjuntar Unity Debugger ofrece esta posibilidad.

Vuelva al Editor de Unity y seleccione Modo de reproducción después de adjuntar el editor de código. Cuando se ejecuta el código en el punto de interrupción, el depurador se detendrá, como se muestra a continuación:

```
Clase pública ExampleScript: MonoBehaviour
{
Usar para la inicialización
Comienzo vacío ()
{
Debug.Log("Mensaje de depuración");
}
}
```

Podemos inspeccionar el contenido de las variables cuando el editor de código se encuentra en un punto de interrupción. El editor de Unity estará inactivo hasta que seleccionemos la opción continuar del depurador o salgamos del modo de depuración.

Depuración en el reproductor

Para analizar el código de secuencia de comandos que se ejecuta en Unity Player, primero seleccione las opciones "Compilación de desarrollo" y "Depuración de secuencias de comandos".

antes de construir el reproductor (estas opciones se pueden encontrar en Archivo > Configuración de construcción). Habilitar la opción "Esperar al depurador administrado" hace que el reproductor espere a que se adjunte un depurador antes de ejecutar cualquier código de secuencia de comandos.

Seleccione la dirección IP y el puerto de nuestro Unity Player para adjuntar el editor de código. El menú desplegable en la opción "Adjuntar a Unity" de Visual Studio.

Depuración de dispositivos Android e iOS

- **Android:** al depurar un reproductor en un dispositivo Android, conéctelo a través de USB o TCP. Para conectarse a un dispositivo Android, por ejemplo, en Visual Studio (Windows), seleccione Depurar > Adjuntar el depurador de Unity.

- **Chrome OS con Android:** Unity no puede detectar dispositivos Chrome OS automáticamente. Para iniciar una conexión, conéctese al dispositivo usando su dirección IP usando Android Debug Bridge (adb), y luego ingrese manualmente la dirección IP en la ventana de depuración.

- **iOS:** conéctese al dispositivo usando TCP mientras depura un reproductor que se ejecuta en un dispositivo iOS. Para conectarse a un dispositivo iOS, por ejemplo, en Visual Studio (Mac), seleccione Depurar > Adjuntar el depurador de Unity.

Uso del depurador para solucionar problemas

La mayoría de los problemas del depurador ocurren porque el editor de código no puede identificar el reproductor o el editor de Unity. Esto significa que no puede conectar el depurador correctamente. La red genera con frecuencia problemas de conexión ya que el depurador requiere una conexión TCP al Editor o Player.

Estas son algunas medidas que podemos tomar para solucionar problemas comunes de conectividad.

Asegúrese de que el depurador esté conectado a la instancia de Unity correcta

Podemos conectar el editor de código a cualquier Unity Editor o Unity Player en nuestra red local que admita la depuración.
Al adjuntar el depurador, asegúrese de que lo estamos adjuntando a la instancia correcta. Si conocemos la dirección IP del dispositivo o el nombre de la máquina donde estamos ejecutando Unity Player, podemos usarlo para encontrar la instancia relevante.

Verifique nuestra conexión de red a la instancia de Unity

Los editores de código usan la misma lógica que Unity Profiler para encontrar una instancia de Unity para depurar. Si el editor de código no puede ubicar la instancia de Unity que necesita, intente adjuntar Unity Profiler a esa instancia. Si Unity Profiler no puede descubrirlo, es posible que haya un firewall en el sistema donde ejecutamos el editor de código o en la máquina donde ejecutamos la instancia de Unity (o posiblemente en ambos).

Asegúrese de que el dispositivo solo tenga una interfaz de red activa

Una gran cantidad de dispositivos tienen varias interfaces de red.
Un teléfono móvil, por ejemplo, puede tener tanto una conexión celular activa como una conexión Wi-Fi activa. Para conectar correctamente el depurador para TCP, el IDE debe establecer una conexión de red con la interfaz adecuada del dispositivo. Si pretendemos depurar a través de Wi-Fi, por ejemplo, ponga el dispositivo en modo avión para desactivar todas las demás interfaces antes de habilitar Wi-Fi.

En el Registro del jugador, podemos ver la dirección IP del Unity Player instruir al IDE para que lo use.

Examine la configuración del cortafuegos

Una conexión TCP conecta la instancia de Unity con el editor de código. Esta conexión TCP tiene lugar en un puerto arbitrario en la mayoría de los sistemas Unity. Por lo general, no deberíamos necesitar conocer este puerto porque el editor de código lo detectará por nosotros. Si no funciona, intente usar una herramienta de análisis de red para establecer qué puertos podrían estar restringidos, ya sea en el sistema donde se ejecuta el editor de código o en la máquina o dispositivo donde se ejecuta la instancia de Unity. Cuando hayamos encontrado los puertos, asegúrese de que nuestro firewall permita el acceso tanto al puerto en el sistema del editor de código como al puerto en la máquina de la instancia de Unity.

Prueba para ver si la información de depuración administrada está disponible

Si el depurador se adjunta, pero no se cargan puntos de interrupción, es posible que el depurador no localice la información de depuración controlada del código. La información de depuración del código administrado se guarda en el disco en archivos .pdb junto con el ensamblado administrado (archivo .dll).

Cuando la configuración adecuada y los parámetros de compilación están habilitados. Unity generará automáticamente esta información de depuración. Por otro lado, Unity no puede crear esta información de depuración para complementos controlados en el Proyecto. La depuración del código de los complementos administrados es factible si los archivos .pdb asociados están en el proyecto de Unity en el disco junto a los complementos administrados.

Evitar que el dispositivo se bloquee

Si el dispositivo que usamos para depurar la aplicación tiene un bloqueo de pantalla, asegúrese de que esté apagado. Los bloqueos de pantalla desconectan el depurador y evitan que se vuelva a conectar. Al depurar programas controlados, es mejor evitar bloquear la pantalla.
Si la pantalla se bloquea, debemos reiniciar el programa en el dispositivo antes de volver a conectar con el depurador.

PRUEBA DE UNIDADES

A medida que nuestro proyecto crece en tamaño y crece la cantidad de scripts, clases y métodos, puede ser un desafío verificar que un cambio en una sección de nuestro código no rompa cosas en otra.

Las pruebas automatizadas nos permiten asegurarnos de que todos los elementos de nuestro código funcionan correctamente. Ahorra tiempo al identificar dónde y cuándo surgen los problemas tan pronto como se presentan durante el desarrollo en lugar de depender de las pruebas manuales o, peor aún, de los informes de errores de nuestros usuarios finales.

El paquete Unity Test Framework es una herramienta que nos permite probar nuestro código en los modos de edición y reproducción y en las plataformas de destino, incluidas las independientes, Android e iOS.

Evaluación

Unity es un motor de juegos multipropósito que es popular para la creación de juegos en todos los paradigmas. Admite gráficos bidimensionales (2D) y bidimensionales (3D), así como secuencias de comandos en C# y capacidad de arrastrar y soltar.
Es un motor de juego multiplataforma creado por Unity Technologies. La gente lo usó para crear simulaciones y videojuegos para consolas, PC y dispositivos móviles en sus primeros días.

La primera presentación formal de Unity ocurrió en 2005 en la Conferencia Mundial de Desarrolladores de Apple. Solo era compatible con OS X en ese momento. Ahora se ha desarrollado y ampliado para apuntar a hasta 27 plataformas. A pesar de su amplia gama de usos, Unity es el más popular para la creación de juegos móviles.

Por lo tanto, una gran parte de su atención también se dirige hacia las plataformas móviles.

¿QUÉ TIENE UNITY PARA LOS DESARROLLADORES?

Unity es un poderoso motor de juego que ofrece a sus creadores una gran cantidad de capacidades funcionales integradas. La representación 3D, la física y la detección de colisiones son ejemplos de esto.

DOI: 10.1201/9781003214755-7

Desde el punto de vista de un desarrollador, esto elimina efectivamente la necesidad de reinventar la rueda. Les ahorra el desarrollo de un nuevo motor de física y la definición de las propiedades y atributos intrínsecos de todos los materiales componentes desde el principio. La inclusión de un Visual Studio incorporado y su interfaz de programación de aplicaciones (API) de secuencias de comandos C# también funciona a su favor.

La disponibilidad de una próspera "Tienda de activos" es lo que hace que Unity se gane el cariño de sus creadores. Asset Store permite a los desarrolladores cargar sus trabajos y compartirlos con el resto de la comunidad.

¿QUÉ ES EXACTAMENTE EL IDE DE UNITY?

Unity también se conoce como un entorno de desarrollo integrado (IDE), además de ser solo un motor de juego.
Esto implica que Unity brinda a los desarrolladores una interfaz a través de la cual pueden acceder a todas las herramientas necesarias en un solo lugar.

Además, el programa Unity tiene un editor visual que permite a los desarrolladores alterar los atributos de varios objetos y construir sus escenas usando la herramienta de arrastrar y soltar.

Aparte de eso, Unity proporciona a sus usuarios una gran cantidad de herramientas y capacidades útiles adicionales. Estos incluyen el uso de una herramienta de línea de tiempo para crear animaciones y navegar entre varios directorios en un proyecto. Unity le brinda la opción de cambiar a un editor diferente de su elección para satisfacer nuestras necesidades de codificación.

¿CUÁL ES EL LENGUAJE UTILIZADO POR UNITY?

El motor de juego de Unity emplea C# junto con varias otras clases y API relacionadas para manejar el código y la lógica.

Lo mejor de usar Unity es que nos permite realizar varios trabajos sin necesidad de administrar o comprender una gran cantidad de código. Sin embargo, si dominamos la codificación, haremos mucho más en la plataforma que el usuario común. Dado lo adaptable que es Unity en los ajustes y alteraciones, tener una sólida comprensión de la codificación sin duda nos ofrecerá una ventaja.

C# es un lenguaje de programación excepcionalmente fácil de usar para principiantes. Esta es la razón fundamental por la que Unity se ha convertido en la plataforma de producción de juegos elegida, especialmente por los creadores jóvenes e inexpertos.

¿QUÉ ESTÁ INCLUIDO EN LA INTERFAZ DE UNITY?

Se divide en las cinco secciones que se enumeran a continuación:

1. **Vista de escena:** esta es la parte en la que el desarrollador diseña los muchos niveles para sus proyectos 3D, juegos y otros escenarios. Esta categoría contiene todos nuestros componentes de diseño y artículos de juego. Somos libres de modificarlos para satisfacer sus necesidades.
2. **Vista del juego:** esta es el área donde podemos ver nuestros resultados. En esencia, Game View proporciona una buena descripción del escenario o nivel que tenemos en mente. Sin embargo, para ver este efecto, una cámara debe estar presente en la escena. Como resultado, esta parte también se denomina Sección de vista de cámara.
3. **Jerarquía:** En la sección Jerarquía veremos todos los objetos del juego que hemos puesto directamente en nuestro escenario o nivel. En pocas palabras, debemos registrar todo lo que muestra Game View. Esto se aplica tanto a los elementos del juego visibles como a los no visuales.

4. **Proyecto:** El trabajo de la Ventana de Proyecto es mostrar el contenido de la carpeta de Activos en nuestro disco. Esta parte brinda acceso a varios aspectos, incluidos scripts, carpetas, texturas, audio, modelos, video y objetos de juego.
5. **Inspector:** El panel Inspector nos permite ver las muchas cualidades y propiedades de cada Game Object seleccionado. La selección del usuario determina los componentes y características que se presentan.

LOS NUEVOS USUARIOS DE UNITY DEBEN SEGUIR ESTE CONSEJO

Haz algo

La mayoría de los consejos que recibimos para los aspirantes a desarrolladores, estudiantes o novatos de Unity se centran en una directriz básica que se extiende más allá de este motor específico: crea algo, incluso si es solo un juego sencillo. Completar incluso un pequeño esfuerzo desde la concepción hasta la finalización tiene un valor excelente.

“Si recién estamos comenzando en la creación de juegos, simplemente debemos hacer cosas y hacer que funcionen de la forma que podamos”, aconseja Williams. “Entonces, si logramos eso, deberíamos ampliar sus horizontes e intentar comprender temas fuera de la creación de juegos, para ir más allá y descubrir qué más podemos lograr que la manera estándar de Unity”.

Sin embargo, Price de Playtonic siente que sin un marco trabajo, no debemos entrar.

“Fijémonos algunos objetivos y pruebas comparativas y pruébalo”, sugiere. “No basemos nuestra decisión solo en nuestras opiniones y conjeturas”.

Descubre cómo fabricar nuestras herramientas

Después de completar algunas tareas simples, debemos avanzar más allá de los fundamentos e investigar otras funcionalidades.

“Aprende a crear nuestras herramientas con Unity”, sugiere Foster. “Todo está escrito en C#. Descubra qué hace nuestro código cuando genera basura. El recolector de basura es un monstruo que ralentizará nuestro juego cuando intentemos colocar cosas en dispositivos móviles, Switch, Xbox y PS4. Y si construimos un código libre de basura, nuestro código fallará y no tendremos que preocuparnos por eso nuevamente”.

Haga un esfuerzo para llegar a la comunidad

Si estamos pensando en usar Unity como nuestro motor de juego, la buena noticia es que su comunidad es lo suficientemente grande como para que la mayoría de nuestros problemas estén a solo una búsqueda de Google de encontrar una solución. Hay docenas de tutoriales disponibles para sea cual sea el grado de desarrollo, desde principiantes hasta optimizaciones pesadas, debemos apoyarnos en esa comunidad.

Gerges describe los juegos como "monstruos interesantes" y "aún no ha trabajado en uno que no produzca momentos adecuados para rascarse la cabeza". Y estamos hablando de alguien que lleva 15 años trabajando en la industria de los juegos.

Continúa: "Nos hemos beneficiado de la comunidad y del apoyo directo de Unity en múltiples ocasiones para ayudarnos a superar obstáculos y circunstancias desafiantes".

El precio termina resaltando. El historial establecido de Unity crea juegos asombrosos en una amplia gama de plataformas y géneros que ya están disponibles.

“Como desarrollador, independientemente del motor que elijamos, debemos ser innovadores en la forma de utilizarlo para lograr un objetivo”.

él añade. “Unity puede construir cualquier juego que nuestra imaginación pueda soñar”.

¿HAY UNA GRAN NECESIDAD DE DESARROLLADORES DE UNITY?

Aparte del negocio de los juegos, un número cada vez mayor de sectores y empresas perciben potencial en el empleo de Unity. La posibilidad de lo que puede lograr este motor brinda una oportunidad increíble para los desarrolladores de Unity, quienes pueden anticipar múltiples perspectivas de carrera en el camino.

Como desarrollador de Unity, debemos estar atentos a las nuevas funciones que publica Unity y mantenernos actualizados para atraer a nuestro futuro empleador. Repasar las habilidades requeridas en áreas como la médica o la automotriz si queremos trabajar en estas profesiones.

Los mejores desarrolladores de Unity aprenden continuamente y se mantienen actualizados sobre los estándares de juegos emergentes y la tecnología de desarrollo.

DESARROLLADOR DE UNITY: HABILIDADES REQUERIDAS

El desarrollo web de Unity requiere habilidades y experiencia específicas en videojuegos.

Un desarrollador de Unity a veces se centrará únicamente en el diseño del juego y en aspectos más creativos, mientras que otras veces se centrará exclusivamente en el código. Una excelente idea sería encontrar un término medio feliz.

Para construir proyectos sofisticados, se requieren habilidades de codificación absolutas (C#, UnityScript, Boo). Los desarrolladores de Unity deben mantenerse actualizados sobre las técnicas de codificación actuales en la industria de los juegos.

Además, tener un gran sentido de la vista es una ventaja y ser capaz de crear magníficas imágenes interactivas es una habilidad crucial para tener como desarrollador de Unity. Si queremos mejorar nuestras habilidades, la Tienda de activos de Unity tiene una gran cantidad de materiales que pueden ayudar a los creadores de juegos con una experiencia mínima en codificación. Las secuencias de comandos visuales, por ejemplo, están disponibles con PlayMaker o Bolt.

Como desarrollador de Unity, debemos tener la
Habilidades siguientes

- Excelente comprensión de Unity, incluidos los scripts, las texturas, la animación, los estilos de GUI y la gestión de sesiones de usuario.

- Los scripts requieren experiencia en programación C#.
- Experiencia en diseño y planificación de niveles.
- Comprensión de la física del juego y los sistemas de partículas.
- Experiencia en el desarrollo de juegos móviles y de consola.
- Capacidad para minimizar la utilización de la memoria y el espacio para soporte de hardware obsoleto.
- Amplios conocimientos en desarrollo 3D y 2D.
- Experimenta con Realidad Virtual o Realidad Aumentada.
- Comprensión excepcional de Programación Orientada a Objetos (OOP) y Programación Orientada a Datos (DOOP).

- Amplio conocimiento del Sistema de Componentes de la Entidad (SEC).

- Conocimientos de diseño y arquitectura contemporáneos patrones.
- Un talento para desarrollar código que sea claro, legible y facil de mantener.
- Amplia experiencia con herramientas de pruebas automatizadas y pruebas unitarias.
- Comprensión de herramientas de versionado de código (Git).

LAS RESPONSABILIDADES DEL DESARROLLADOR DE UNITY

Un desarrollador de Unity en el negocio de los juegos está creando juegos para numerosas plataformas de destino utilizando el marco de trabajo de Unity.

Su responsabilidad clave será traducir ideas, conceptos y requisitos de diseño en un juego práctico y emocionante. Se requiere dedicación a la resolución colaborativa de problemas, diseño inteligente y un resultado de alta calidad.

¿Qué responsabilidades tiene un desarrollador de Unity?

- Implementar características del juego siguiendo el diseño establecido.
- Convertir los requisitos de diseño en un juego jugable.
- Implementar funcionalidades de manera rápida y ágil.
- Comuníquese con otros miembros del equipo para crear una canalización eficiente e incorporar activos de medios.
- Crear, diseñar y mantener código que sea eficiente, reutilizable y confiable.

- Asegúrese de que las aplicaciones tengan el rendimiento, la calidad y la capacidad de respuesta más excelentes posibles.
- Identifique cuellos de botella y fallas y proponga estrategias para resolver y minimizar estos problemas.
- Integrar la entrada del jugador para mejorar los aspectos del juego.
- Ayudar a mantener la calidad del código, la estructura y automatización.
- Agregar código a repositorios remotos como Git.

Índice

j

k

L

METRO

R

S

T

EN

EN

En

Y