

UT 03 - Utilización de librerías multimedia integradas

Conceptos sobre aplicaciones multimedia.

Arquitectura del API utilizado - Unity

Fuentes de datos multimedia.

Datos basados en el tiempo - Unity.

Procesamiento de objetos multimedia.

Herramientas y técnicas de pruebas de videojuegos.

Pruebas Videojuegos vs Pruebas de software Clásicas.

Clasificación de errores en videojuegos.

Detección y corrección automática de errores

Unity – Debug.Log

Unity - Coverage

Unity - Profiler

Unity - Device Simulator

Documentación del guion.

CFGS Desarrollo de Aplicaciones Multiplataforma – Curso 23/24 – IES AGL

Programación Multimedia y Dispositivos Móviles.

Conceptos sobre aplicaciones multimedia.

La multimedia es muy usada en la industria del entretenimiento, para desarrollar especialmente efectos especiales en películas y la animación para los personajes de caricaturas.

Los juegos de la multimedia son un pasatiempo popular y son programas del software como CD-ROMs o disponibles en línea. Algunos juegos de vídeo también utilizan características de la multimedia. Los usos de la multimedia permiten que los usuarios participen activamente en vez de estar sentados llamados recipientes pasivos de la información, la multimedia es interactiva.

- Tipos de información multimedia:
- **Texto:** sin formatear, formateado, lineal e hipertexto.
- **Gráficos:** utilizados para representar esquemas, planos, dibujos lineales, etc.
- **Imágenes:** son documentos formados por píxeles. Pueden generarse por copia del entorno (escaneado, fotografía digital) y tienden a ser ficheros muy voluminosos.
- **Animación:** presentación de un número de gráficos por segundo que genera en el observador la sensación de movimiento.
- **Vídeo:** Presentación de un número de imágenes por segundo, que crean en el observador la sensación de movimiento. Pueden ser sintetizadas o captadas.
- **Sonido:** puede ser habla, música u otros sonidos

Conceptos sobre aplicaciones multimedia.

El trabajo multimedia está actualmente a la orden del día y un buen profesional debe seguir unos determinados pasos para elaborar el producto.

- **Definir el mensaje clave.** Saber qué se quiere decir. Para eso es necesario conocer al cliente y pensar en su mensaje comunicacional. Es el propio cliente el primer agente de esta fase comunicacional.
- **Conocer al público.** Buscar qué le puede gustar al público para que interactúe con el mensaje. Aquí hay que formular una estrategia de ataque fuerte. Se trabaja con el cliente, pero es la agencia de comunicación la que tiene el protagonismo. En esta fase se crea un documento que los profesionales del multimedia denominan "ficha técnica", "concepto" o "ficha de producto". Este documento se basa en 5 ítems: necesidad, objetivo de la comunicación, público, concepto y tratamiento.
- **Desarrollo o guion.** Es el momento de la definición de la Game-play: funcionalidades, herramientas para llegar a ese concepto. En esta etapa solo interviene la agencia que es la especialista.
- **Creación de un prototipo.** En multimedia es muy importante la creación de un prototipo que no es sino una pequeña parte o una selección para testear la aplicación. De esta manera el cliente ve, ojea, interactúa... Tiene que contener las principales opciones de navegación.

Conceptos sobre aplicaciones multimedia.

Ahora ya se está trabajando con digital, un desarrollo que permite la interactividad. Es en este momento cuando el cliente, si está conforme, da a la empresa el dinero para continuar con el proyecto.

En relación con el funcionamiento de la propia empresa, está puede presuponer el presupuesto que va a ser necesario, la gente que va a trabajar en el proyecto (lista de colaboradores). En definitiva, estructura la empresa. **El prototipo es un elemento muy importante en la creación y siempre va a ser testado (público objetivo y encargados de comprobar que todo funciona)**

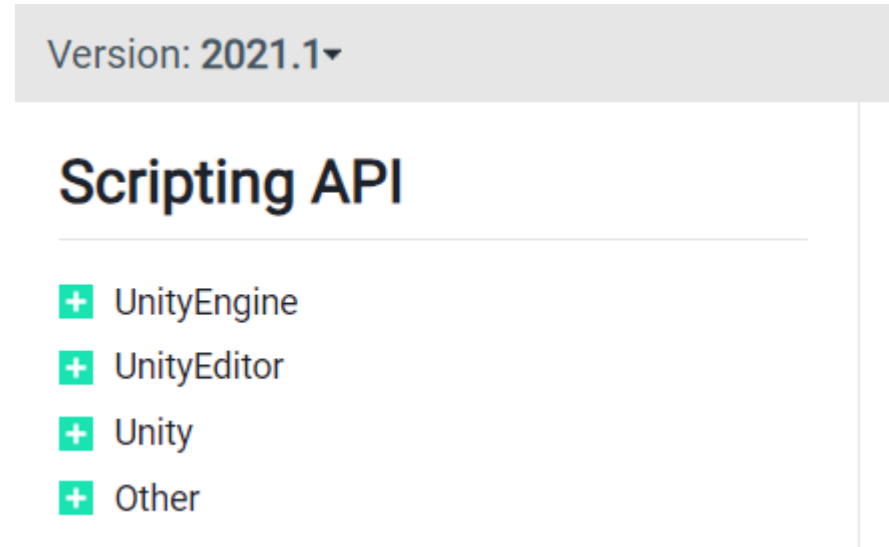
Creación del producto: En función de los resultados del testeo del prototipo, se hace una redefinición y se crea el producto definitivo, **el esquema del multimedia.**

Conceptos sobre aplicaciones multimedia.

Los diferentes tipos de multimedia se pueden clasificar de acuerdo a la finalidad de la información, o también, al medio en el cual serán publicadas.

- **Multimedia educativa:** Es importante recalcar que la multimedia educativa es previa a que el computador apareciera, se puede considerar como un proceso no lineal esto hace que el **estudiante lleve su propio orden en su modelo educativo (a distancia, presencial etc.)**. Se fundamenta en un desarrollo navegable que permite cierta libertad de moverse sobre la aplicación.
- **Multimedia publicitaria:** Es el uso de diferentes medios enfocado a una campaña publicitaria, esto ha generado nuevos espacios en este sector, se viene presentando un cambio de los medios tradicionales a los digitales con un abanico enorme de nuevas posibilidades, tabletas, móviles, desarrollo web, TDT (Televisión Digital Terrestre), hipertexto y el correo, y como elemento destacado las redes sociales como herramienta de difusión viral.
- **Multimedia comercial:** En este tipo de multimedia encontramos una gran variedad de productos, tales como: **Bases de datos (DB), promociones, catálogos, videojuegos, simuladores, páginas web, publicidad entre otros**, todo este material se presenta en forma digital, interactivo y su funcionalidad principal es la de convencer a un posible comprador o cliente de adquirir un servicio o producto. De alguna forma este tipo de multimedia está directamente relacionada con el aprendizaje electrónico (e-learning).
- **Multimedia informativa:** Está relacionada con los elementos multimediales que brindan información, tales como: **noticias, prensa, revistas, televisión y diarios**, esta información se presenta en la mayoría de los casos en forma masiva (entorno mundial) y se mantiene actualizada al momento de los hechos, su valor informativo es primordial para conocer hechos antes que los medios de comunicación tradicionales.

Arquitectura del API utilizado - Unity



Como se puede observar Unity dispone de 4 niveles en los que expone el API que se utiliza a nivel de desarrollador. A continuación, detallaremos los aspectos de cada uno. Más información en <https://docs.unity3d.com/es/2021.1/ScriptReference/index.html>

Arquitectura del API utilizado - Unity

A continuación, se muestran los diferentes aspectos técnicos que se encuentran dentro de **Unity Engine**.

Version: 2021.1▼

Scripting API

UnityEngine

- + UnityEngine.Accessibility
- + UnityEngine.AI
- + UnityEngine.Analytics
- + UnityEngine.Android
- + UnityEngine.Animations
- + UnityEngine.Apple
- + UnityEngine.Assertions
- + UnityEngine.Audio
- + UnityEngine.CrashReportHandler
- + UnityEngine.Diagnostics
- + UnityEngine.Events
- + UnityEngine.Experimental
- + UnityEngine.iOS
- + UnityEngine.Jobs
- + UnityEngine.LowLevel
- + UnityEngine.Lumin
- + UnityEngine.Networking

- + UnityEngine.ParticleSystemJobs
- + UnityEngine.Playables
- + UnityEngine.PlayerLoop
- + UnityEngine.Profiling
- + UnityEngine.Rendering
- + UnityEngine.SceneManagement
- + UnityEngine.Scripting
- + UnityEngine.SearchService
- + UnityEngine.Serialization
- + UnityEngine.SocialPlatforms
- + UnityEngine.Sprites
- + UnityEngine.SubsystemsImplementa
- + UnityEngine.TestTools
- + UnityEngine.TextCore
- + UnityEngine.Tilemaps
- + UnityEngine.tvOS
- + UnityEngine.U2D
- + UnityEngine.UIElements
- + UnityEngine.VFX
- + UnityEngine.Video
- + UnityEngine.Windows

- + UnityEngine.WSA
- + UnityEngine.XR
- + Classes
- + Interfaces
- + Enumerations
- + Attributes
- + Assemblies

Arquitectura del API utilizado - Unity

A continuación, se muestran los diferentes aspectos técnicos que se encuentran dentro de **Unity Editor**.

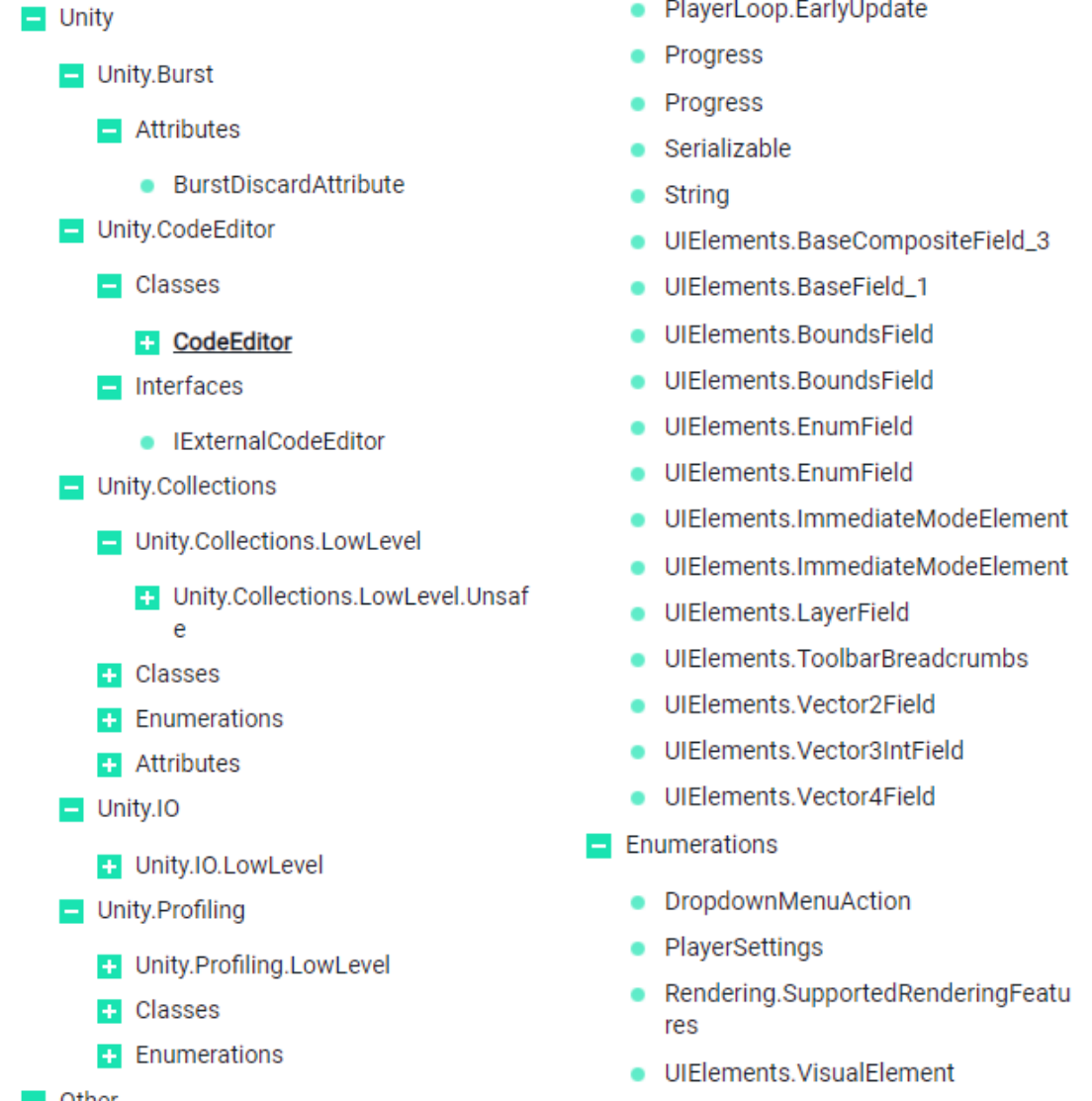
Información más específica orientada hacia la parte del desarrollo del videojuego/aplicación.

- UnityEditor

- + UnityEditor.AI
- + UnityEditor.AnimatedValues
- + UnityEditor.Animations
- + UnityEditor.AssetImporters
- + UnityEditor.Build
- + UnityEditor.Compilation
- + UnityEditor.CrashReporting
- + UnityEditor.EditorTools
- + UnityEditor.Events
- + UnityEditor.Experimental
- + UnityEditor.Il2Cpp
- + UnityEditor.IMGUI
- + UnityEditor.Localization
- + UnityEditor.Media
- + UnityEditor.MemoryProfiler
- + UnityEditor.MPE
- + UnityEditor.Networking
- + UnityEditor.PackageManager
- + UnityEditor.Playables
- + UnityEditor.Presets
- + UnityEditor.Profiling
- UnityEditor.ProjectWindowC
- + UnityEditor.Purchasing
- + UnityEditor.Rendering
- + UnityEditor.SceneManagem
- + UnityEditor.SceneTemplate
- + UnityEditor.Scripting
- + UnityEditor.Scripting
- + UnityEditor.SearchService
- + UnityEditor.ShortcutManagement
- + UnityEditor.Sprites
- + UnityEditor.U2D
- + UnityEditor.UIElements
- + UnityEditor.UnityLinker
- + UnityEditor.VersionControl
- + Classes
- + Interfaces
- + Enumerations
- + Attributes
- + Assemblies

Arquitectura del API utilizado - Unity

- A continuación, se muestran más a detalle los otros 2 niveles API de Unity tanto el denominado Unity como Other.
- Esta jerarquía es aplicada en las otras dos categorías, e imposible de replicároslo aquí, para que observéis el alcance de la información que se proporciona junto con los propios foros para problemas concretos con este Motor Gráfico.
- También se debe añadir que existen otras **API externas** adicionales como las asociadas a las diferentes gafas de Realidad Extendida como las MetaQuest.



Fuentes de datos multimedia.

A continuación, se enumeran diferentes plataformas en las cuales poder obtener de forma gratuita diferentes recursos multimedia.

- Modelos 3D: <https://free3d.com/es/>
<https://www.turbosquid.com/es/Search/3D-Models/free>
- Imágenes y videos: <https://pixabay.com/es/>
<https://www.pexels.com/es-es/>
- Sonidos: <https://pixabay.com/es/sound-effects/>
- Unity Asset Store: <https://assetstore.unity.com/> con una gran cantidad de recursos multimedia para el desarrollo de aplicaciones multiplataforma.

Datos basados en el tiempo - Unity.

Clase Time:

La clase Time de Unity proporciona propiedades básicas importantes que te permiten trabajar con valores relacionados con el tiempo en tu proyecto.

Esta página contiene explicaciones para algunos miembros de la clase Time más utilizados y cómo se relacionan entre sí. Puede leer descripciones individuales de cada miembro de la clase de tiempo en la página de referencia del script de tiempo.

La clase Time tiene algunas propiedades que le proporcionan valores numéricos que le permiten medir el tiempo transcurrido mientras se ejecuta su juego o aplicación. Por ejemplo:

- **Time.time**: devuelve la cantidad de tiempo en segundos desde que su proyecto comenzó a reproducirse.
- **Time.deltaTime**: devuelve la cantidad de tiempo en segundos transcurrido desde que se completó el último fotograma. Este valor varía dependiendo de los frames por segundo (FPS) a la que se ejecuta tu juego o aplicación.

Datos basados en el tiempo - Unity.

La clase Time también le proporciona propiedades que le permiten controlar y limitar cómo transcurre el tiempo, por ejemplo:

- **Time.timeScale** controla la velocidad a la que transcurre el tiempo. Puede leer este valor o configurarlo para controlar qué tan rápido pasa el tiempo, lo que le permite crear efectos de cámara lenta.
- **Time.fixedDeltaTime** controla el intervalo de Unity paso de tiempo fijo bucle (usado para física y si desea escribir código determinista basado en el tiempo).
- **Time.maximumDeltaTime** establece un límite superior en la cantidad de tiempo que el motor informará que ha pasado por las propiedades de "tiempo delta" anteriores.

Datos basados en el tiempo - Unity.

Pasos de tiempo variables y fijos

- Unity tiene **dos sistemas** que rastrean el tiempo, **uno con una cantidad de tiempo variable entre cada paso y otro con una cantidad de tiempo fija entre cada paso.**
- El sistema de **paso de tiempo variable** opera en el proceso repetido de dibujar un frame en la pantalla y **ejecutar el código de su aplicación o juego una vez por frame.**
- El sistema de **pasos de tiempo fijo** avanza en una cantidad predefinida en cada paso y no está vinculado a las actualizaciones del marco visual. Se asocia más comúnmente con el sistema de física, que se ejecuta a la velocidad especificada por el tamaño del paso de tiempo fijo, pero también puede ejecutar su propio código en cada paso de tiempo fijo si es necesario.

Datos basados en el tiempo - Unity.

Gestión de velocidad de cuadros variable

- La velocidad de cuadros de su juego o aplicación puede variar debido al tiempo que lleva mostrar y ejecutar el código para cada cuadro. Esto se ve afectado por las capacidades del dispositivo en el que se ejecuta, y también por la cantidad variable de complejidad de los gráficos mostrados y el cálculo requerido en cada cuadro. Por ejemplo, **tu juego puede ejecutarse a una velocidad de fotogramas más lenta cuando hay cien personajes activos y en pantalla, en comparación con cuando solo hay uno.** Esta velocidad variable a menudo se denomina “frames por segundo”, o **FPS**
- A menos que su configuración de calidad o el paquete Adaptive Performance lo limiten de otra manera, Unity intenta ejecutar su juego o aplicación a la velocidad de fotogramas más rápida posible. Puedes ver más detalles de lo que ocurre en cada frame en el diagrama de orden de ejecución, en la sección denominada “Lógica del juego”.
- Unity proporciona el **Update**, método como punto de entrada, para que usted ejecute su propio código en cada cuadro. Por ejemplo, en el Update, método del personaje de tu juego, puedes leer la entrada del usuario desde un joystick y mover el personaje hacia adelante una cierta cantidad. Es importante recordar que cuando se manejan acciones basadas en el tiempo como esta, la velocidad de fotogramas del juego puede variar y, por lo tanto, el período de tiempo entre las llamadas de actualización también varía.

Datos basados en el tiempo - Unity.

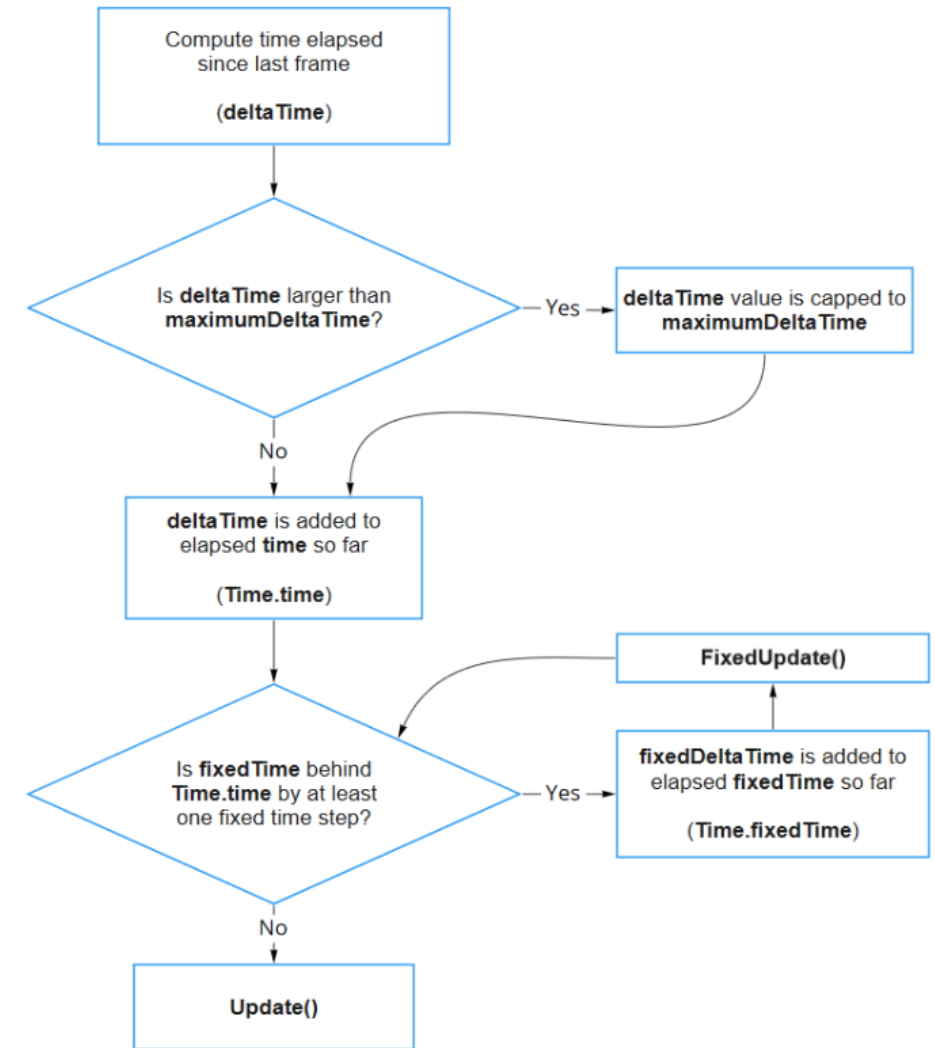
Paso de tiempo fijo

- A diferencia de la actualización del marco principal, el sistema de física de Unity funciona en un paso de tiempo fijo, lo cual es importante para la precisión y coherencia de la simulación. Al comienzo de cada cuadro, Unity realiza tantas actualizaciones fijas como sean necesarias para ponerse al día con la hora actual. Puedes ver más detalles de lo que ocurre durante el ciclo de actualización fijo en el diagrama de orden de ejecución , en la sección marcada como “Física”.
- También puedes ejecutar tu propio código sincronizado con el paso de tiempo fijo, si es necesario. Esto se usa más comúnmente para ejecutar su propio código relacionado con la física, como aplicar una fuerza a un Cuerpo rígido
- Unity proporciona el `FixedUpdate`, método como punto de entrada para que usted ejecute su propio código en cada paso de tiempo fijo.
- Puede leer o cambiar la duración del paso de tiempo fijo en la ventana Hora o desde un script utilizando la propiedad `Time.fixedDeltaTime`
- Nota: Un valor de paso de tiempo más bajo significa actualizaciones físicas más frecuentes y simulaciones más precisas, lo que genera una mayor carga de CPU.

Datos basados en el tiempo - Unity.

La lógica del tiempo de Unity

El siguiente diagrama de flujo ilustra la lógica que utiliza Unity para contar el tiempo en un solo fotograma y cómo las propiedades time, deltaTime, fixDeltaTime y MaximumDeltaTime se relacionan entre sí.



Datos basados en el tiempo - Unity.

- Más información de la anterior información y de los siguientes métodos en:

<https://docs.unity3d.com/Manual/TimeFrameManagement.html>

Métodos:

- [Time.time](#)
- [Time.unscaledTime](#)
- [Time.deltaTime](#)
- [Time.unscaledDeltaTime](#)
- [Time.smoothDeltaTime](#)
- [Time.timeScale](#)
- [Time.maximumDeltaTime](#)

Procesamiento de objetos multimedia - Unity.

Orden de ejecución de las funciones de evento

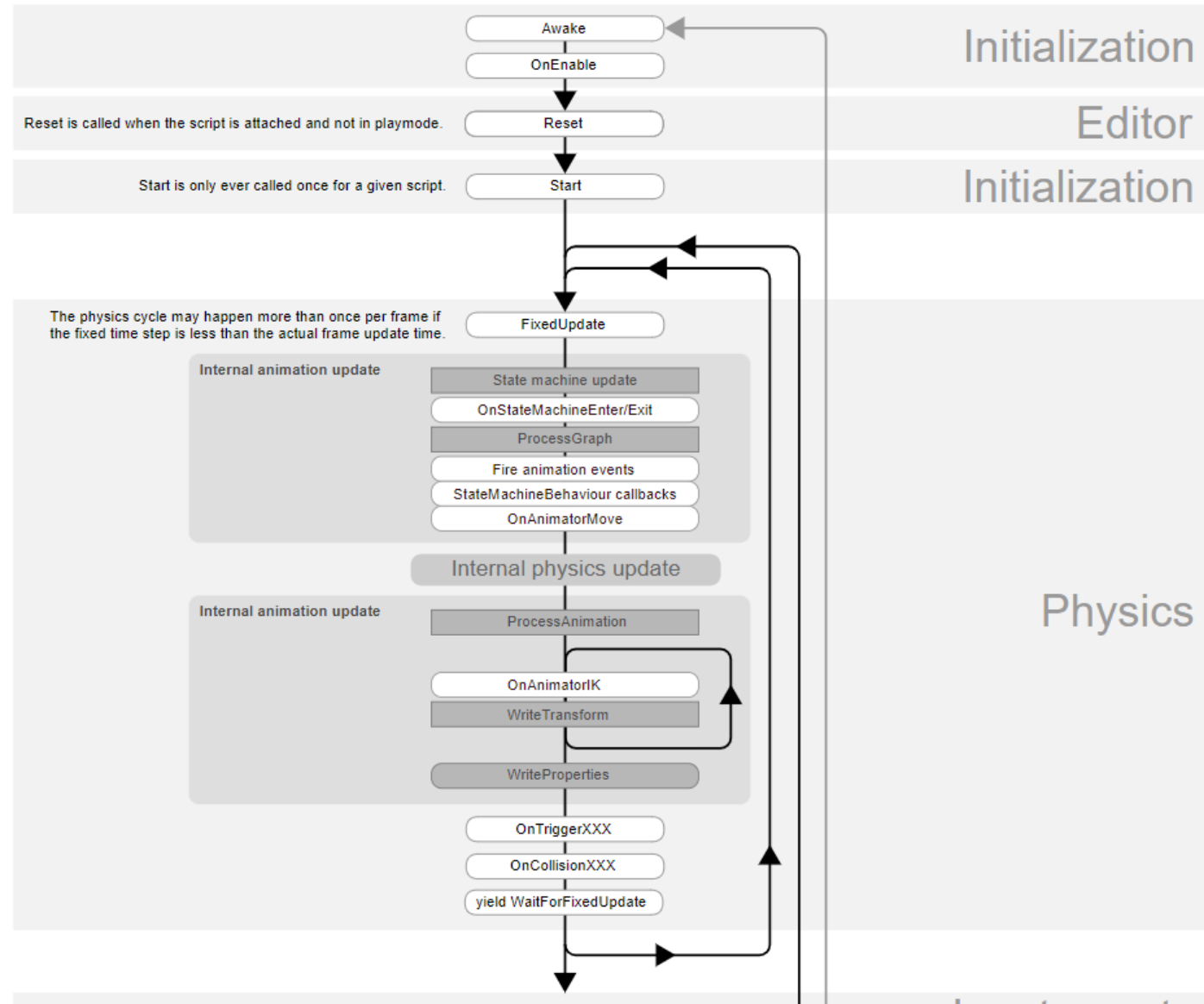
- La ejecución de un script de Unity ejecuta una serie de funciones de evento en un orden predeterminado. En esta página se describen esas funciones de evento y se explica cómo encajan en la secuencia de ejecución.

Información general sobre el ciclo de vida del script

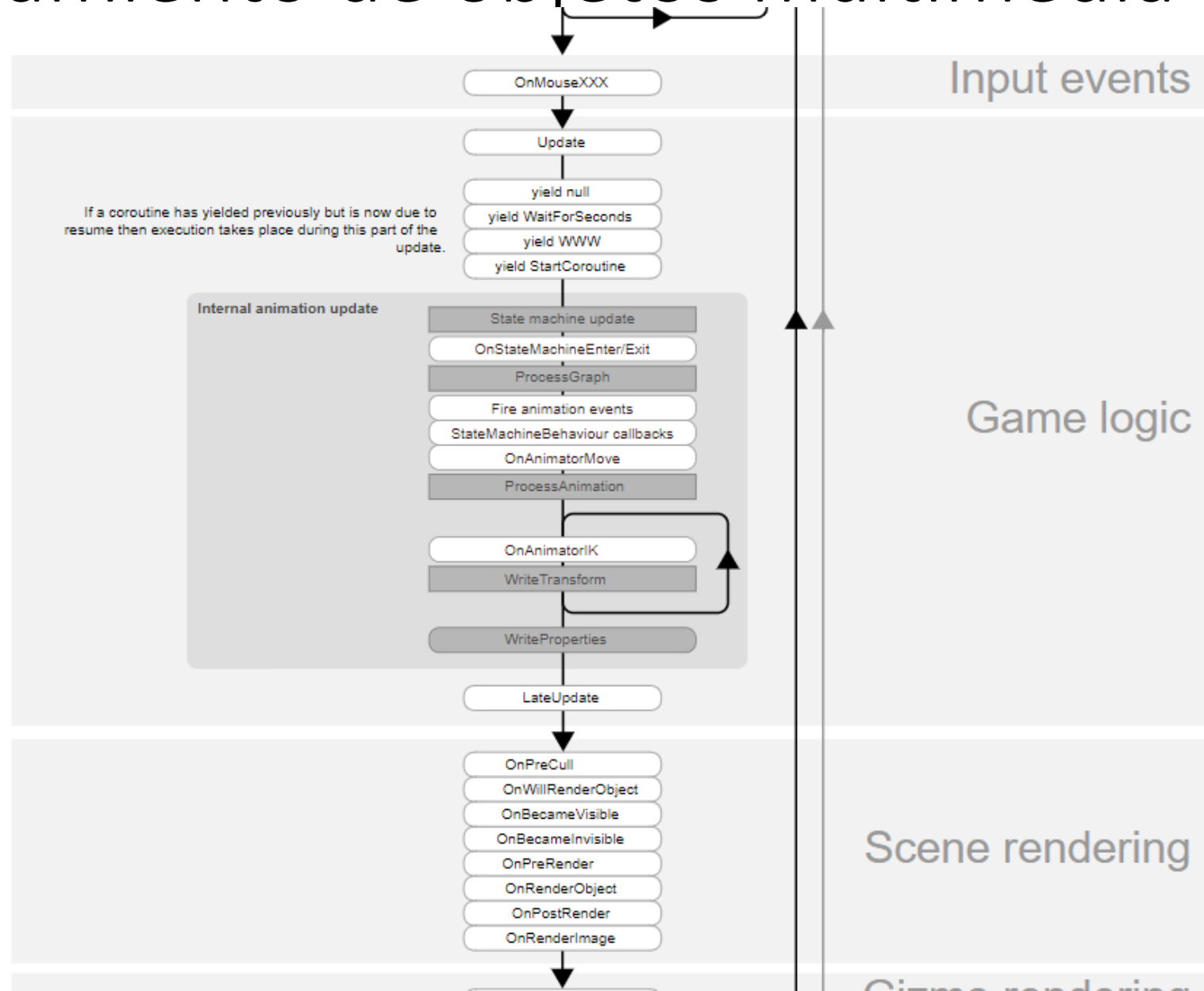
- El siguiente diagrama resume cómo Unity ordena y repite las funciones de eventos a lo largo de la vida útil de un script.
- Para obtener más información sobre las distintas funciones de eventos, consulte las siguientes secciones:
- [First Scene Load](#) [Editor](#) [Before the first frame update](#) [In between frames](#)
[Update order](#) [Animation update loop](#)
- [Rendering](#) [Coroutines](#) [When the object is destroyed](#) [When quitting](#)
- Más info en:

<https://docs.unity3d.com/Manual/ExecutionOrder.html>

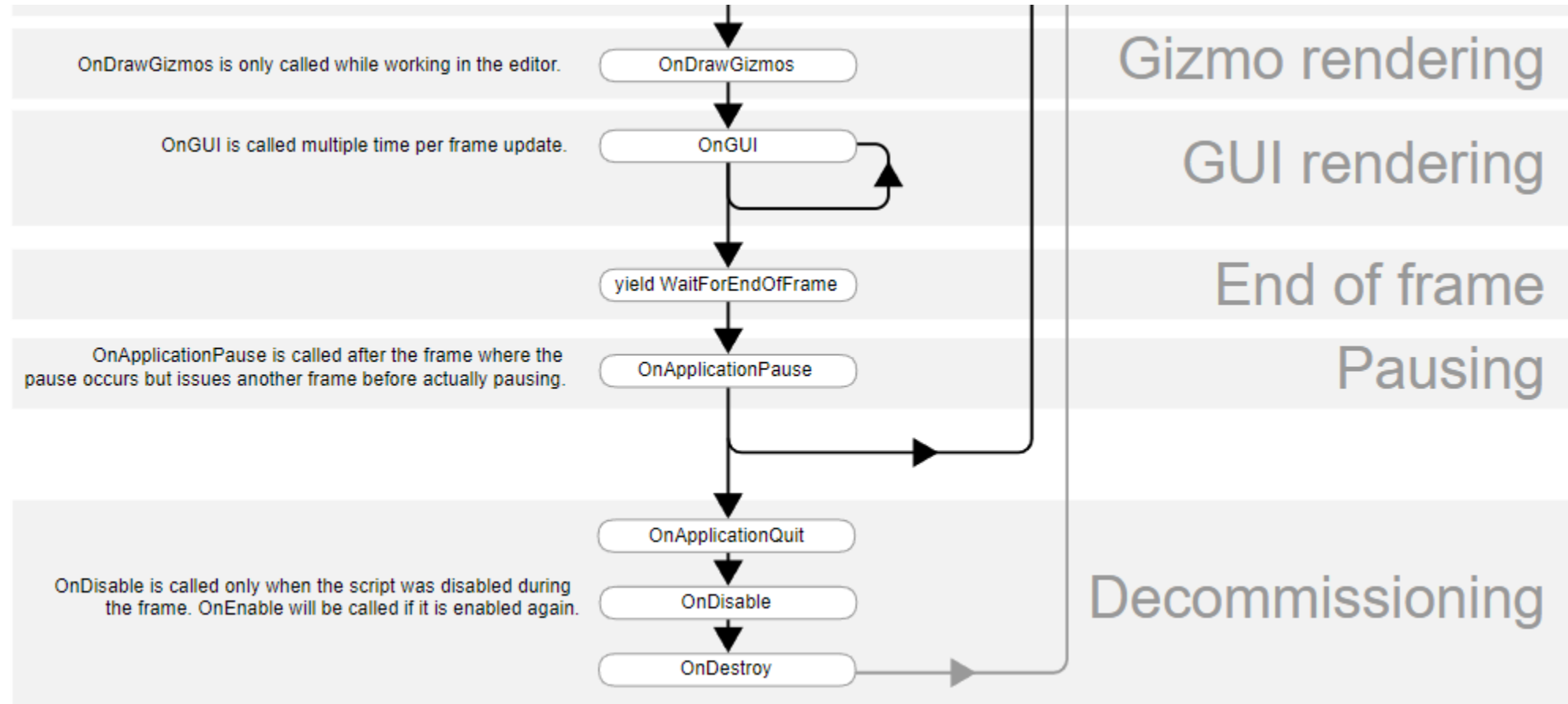
Procesamiento de objetos multimedia - Unity.



Procesamiento de objetos multimedia - Unity.



Procesamiento de objetos multimedia - Unity.



Herramientas y técnicas de pruebas de videojuegos.

- La **prueba** en el mundo de los videojuegos , especialmente el software de prueba de campo , es una práctica de evaluación de la funcionalidad de un videojuego.
- Esta prueba se puede realizar durante el desarrollo del juego para buscar fallos o mejoras necesarias (realizadas por probadores en versión alfa o en beta abierta o cerrada).
- Diferentes técnicas permiten identificar la presencia de errores para corregirlos.
- Los videojuegos también son probados **durante el marketing por periodistas para evaluar su calidad y contenido.**

Herramientas y técnicas de pruebas de videojuegos.

- **Las técnicas de prueba automática no funcionan en el campo de los videojuegos porque la mayoría de los videojuegos son parte de software con características emergentes, una categoría específica de software donde la información devuelta por el sistema no es predecible.**
- De hecho, **la clave del éxito de un videojuego es sorprender al jugador.**
- Como resultado, las pruebas son difíciles de configurar en los videojuegos porque requieren conocer todos los datos de entrada, así como las salidas correspondientes, pero estas salidas son por definición impredecibles.
- Incluso si se pudieran enumerar todas las entradas y salidas posibles, **la capacidad de los desarrolladores para realizar pruebas que cubran todos estos estados disminuye exponencialmente.**
- Los ingenieros son cada vez menos capaces de predecir los resultados de un sistema, y su capacidad para verificar y validar el software se verá considerablemente disminuida.

Herramientas y técnicas de pruebas de videojuegos.

- **También hay preguntas (especialmente a nivel de diseño) que no pueden corresponder a una prueba unitaria** (por ejemplo: “¿el 90% de los jugadores terminan el juego en menos de 5 minutos?”).
- La metodología actual de prueba de videojuegos requiere estructuralmente la **contratación de cientos de probadores humanos** que jugarán diferentes versiones del juego a lo largo del proceso de desarrollo para **detectar los diversos errores y permitir que los desarrolladores los corrijan**.

Pruebas Videojuegos vs Pruebas de software Clásicas

- **Los videojuegos son un software especial**, por lo que también deben respetar las limitaciones relativas a las funcionalidades solicitadas, el presupuesto y el tiempo, proporcionando una calidad aceptable.
- Sin embargo, **no tienen los mismos objetivos y prioridades que el software tradicional.**
- De hecho, **los desarrolladores de videojuegos buscan entretener y divertir al usuario.**
- También es un campo creativo que toca las emociones del jugador y busca brindarle satisfacción. **Debido a las diferencias observadas entre los videojuegos y el software tradicional, todas las pruebas serán, por tanto, diferentes.**

Pruebas Videojuegos vs Pruebas de software Clásicas

Podemos destacar algunas diferencias importantes:

- **El mantenimiento en los videojuegos parece menos importante debido a la corta vida útil de los juegos.** Por lo tanto, no es esencial que el equipo de desarrollo configure la arquitectura y las pruebas necesarias para facilitar el mantenimiento en el futuro, mientras que esto es una prioridad en el software tradicional.
- **Las pruebas en los videojuegos no se centran en las mismas partes que en el software.** De hecho, el videojuego preferirá la experiencia del usuario (el juego debe ser intuitivo, fácil de manejar, divertido) al aspecto técnico (estabilidad, seguridad).
- **Durante la fase de desarrollo de un videojuego, es normal ver cambios tardíos en el producto, por lo que es difícil automatizar las pruebas y predecir el resultado.**
- En general, **los videojuegos tienen fechas de lanzamiento inflexibles**, como durante las vacaciones. **Si la gestión del tiempo asignado es deficiente, los desarrolladores no podrán completar sus fases de prueba.**
- **En el software**, las pruebas de usuario permiten identificar y luego eliminar los obstáculos a la productividad del usuario. **En los videojuegos**, el obstáculo es necesario: la prueba de usuario (playtest) permitirá (entre otras cosas) adaptar el obstáculo a las habilidades del jugador, y no eliminarlo.

Por lo tanto, los videojuegos tendrán preferencia por las pruebas de usabilidad y las "pruebas exploratorias"

Clasificación de errores en videojuegos.

- **No existe una clasificación oficial y estándar** de los errores presentes en los videojuegos, pero **existe una taxonomía** que permite clasificar los errores en dos categorías:
 - **Errores atemporales**, que pueden aparecer en cualquier momento del juego.
 - **Errores temporales**, que requieren conocimiento, el estado anterior del juego para reconocer el error.

Clasificación de errores en videojuegos.

Errores atemporales

- **Posición inválida:** un objeto no debería poder estar donde está.
 - **Objeto fuera de límites para cualquier estado:** un objeto se encuentra en un lugar normalmente inaccesible (por ejemplo, un personaje camina sobre la superficie del agua).
 - **Objeto fuera de los límites para un estado específico:** un objeto está en una posición inaccesible pero que podría ser válida en otro contexto (por ejemplo: un NPC aparece en una escena cuando no debería).
- **Representación gráfica no válida:** la representación gráfica de ciertos objetos o aspectos del mundo es incorrecta (por ejemplo, un personaje comienza a nadar mientras está en tierra).
- **Cambio de valor no válido:** un problema con un contador (por ejemplo, recibir una bala no elimina ningún punto de vida).
- **Estupidez artificial:** la inteligencia artificial no cumple con las expectativas, es decir, los personajes no jugadores (NPC) rompen la ilusión de inteligencia con algunas de sus acciones (por ejemplo, el NPC camina contra una pared o bloquea el paso).
- **Errores de información:** problemas relacionados con la información conocida por el jugador.
 - **Falta de información:** el jugador debe conocer cierta información para comprender el juego, pero no la tiene (por ejemplo, una escena no funciona).
 - **Acceso a información no válida:** el jugador conoce más información de la esperada por accidente (por ejemplo, al ver a través de una pared).
 - **Información fuera de orden:** el jugador recibe información en el orden incorrecto (por ejemplo, cuando hay varias formas de obtener información, el jugador ve a su personaje descubrir lo mismo varias veces).
- **Errores de acción:** problemas relacionados con las acciones del juego.
 - **Acción imposible:** acciones que el juego debería permitir pero que son imposibles (por ejemplo: imposible recoger un objeto).
 - **Acción no autorizada:** acciones posibles cuando el juego está en pausa o durante una escena, o un guión que comienza en el momento equivocado (por ejemplo: la escena se activa antes de que llegue el personaje).

Clasificación de errores en videojuegos.

Errores de tiempo

- **Posición inválida en el tiempo:** movimientos imposibles (ej. ∴ teletransportación) o ausencia de movimientos esperados (ej. ∴ un NPC está atascado).
- **Estado de contexto inválido a lo largo del tiempo:** los objetos permanecen en el mismo estado durante demasiado tiempo (por ejemplo: inmovilidad).
- **Ocurrencia de eventos no válidos a lo largo del tiempo:** los eventos ocurren con demasiada frecuencia o muy raramente (p. Ej., Un evento considerado raro ocurre varias veces seguidas).
- **Eventos interrumpidos:** las acciones en el juego terminan abruptamente (por ejemplo, un personaje habla pero la banda sonora se detiene sin ningún motivo).
- **Problemas de respuesta de implementación :** problemas relacionados con el hardware (por ejemplo, latencias).

Detección y corrección automática de errores

Detección de errores

- Los estudios han demostrado la **posibilidad de detectar problemas en tiempo de ejecución**. De hecho, los programas de videojuegos tienen **similitudes: están orientados a objetos y tienen un bucle de juego**.
- Sabemos que **un giro de bucle cambia atómicamente el estado del juego**. En el lanzamiento, el bucle se inicia y espera la entrada del jugador. Con cada acción del usuario, se procesan los datos, se cambia el estado del juego y el resultado se devuelve al jugador. Cuando termina la fase de juego, el bucle se destruye.
- **Al monitorear el entorno de ejecución**, es posible detectar dinámicamente problemas durante una ronda de bucle a través de restricciones y condiciones.
- **Por lo tanto, se detectan los problemas que podrían haber pasado desapercibidos para un probador y luego se pueden corregir.**

Detección y corrección automática de errores

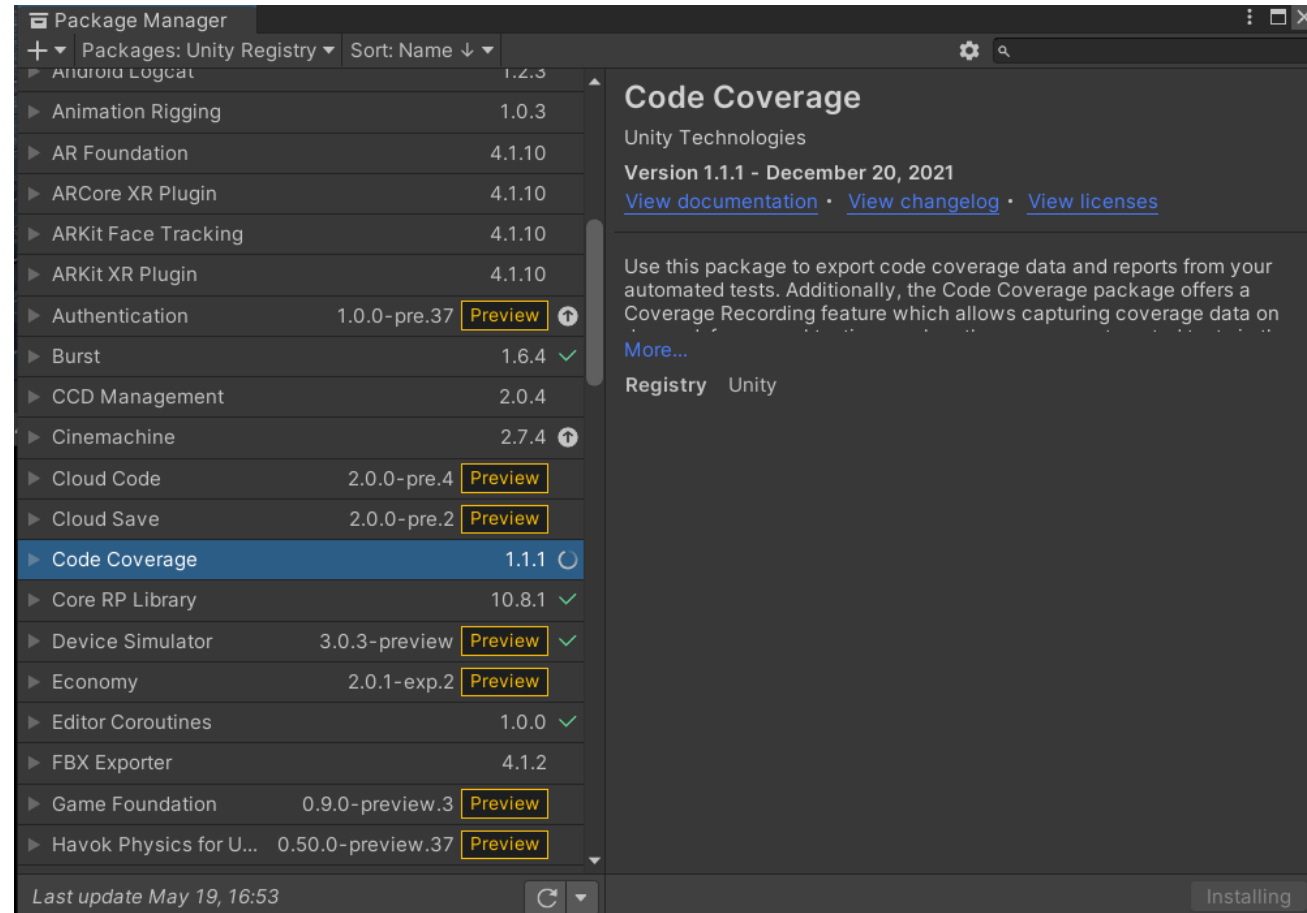
Arreglo del fallo

- También es posible, según algunos estudios, **corregir en tiempo de ejecución los problemas detectados.**
- **Cuando no se cumple una condición**, el valor que viola esta restricción puede modificarse dinámicamente para corregir el error.
- Por otro lado, nada indica que esta corrección no dará lugar a regresiones, es decir, a **desencadenar otras infracciones.**
- El peor caso posible es que cada reparación provoque un error que luego se corrija automáticamente pero cause un nuevo problema, y así sucesivamente, creando un ciclo de reparación infinito.

Unity – Debug.Log

- Para realizar comprobaciones de variables, posicionamiento, colisiones, entre otros muchos.
- <https://docs.unity3d.com/ScriptReference/Debug.Log.html>

Unity - Coverage



The screenshot shows the Unity Package Manager window. On the left, a list of packages is displayed, sorted by name. The 'Code Coverage' package is highlighted in blue. On the right, the details for the 'Code Coverage' package are shown, including its version (1.1.1) and a description of its functionality.

Package Name	Version	Status
Android Logcat	1.2.3	
Animation Rigging	1.0.3	
AR Foundation	4.1.10	
ARCore XR Plugin	4.1.10	
ARKit Face Tracking	4.1.10	
ARKit XR Plugin	4.1.10	
Authentication	1.0.0-pre.37	Preview
Burst	1.6.4	✓
CCD Management	2.0.4	
Cinemachine	2.7.4	ⓘ
Cloud Code	2.0.0-pre.4	Preview
Cloud Save	2.0.0-pre.2	Preview
Code Coverage	1.1.1	○
Core RP Library	10.8.1	✓
Device Simulator	3.0.3-preview	Preview
Economy	2.0.1-exp.2	Preview
Editor Coroutines	1.0.0	✓
FBX Exporter	4.1.2	
Game Foundation	0.9.0-preview.3	Preview
Havok Physics for U...	0.50.0-preview.37	Preview

Code Coverage
Unity Technologies
Version 1.1.1 - December 20, 2021
[View documentation](#) · [View changelog](#) · [View licenses](#)

Use this package to export code coverage data and reports from your automated tests. Additionally, the Code Coverage package offers a Coverage Recording feature which allows capturing coverage data on [More...](#)

Registry Unity

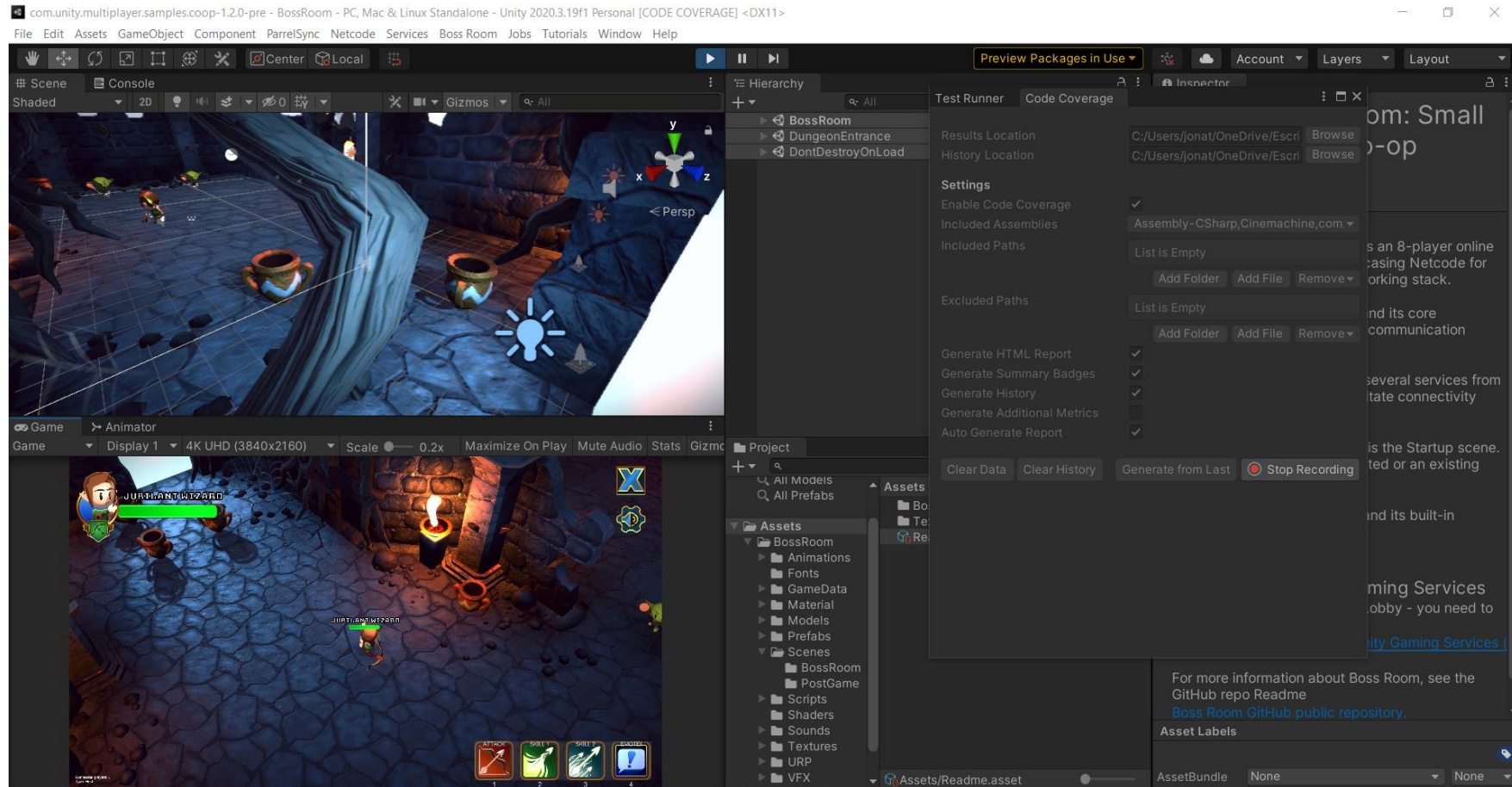
Last update May 19, 16:53

Installing

Unity - Coverage

- Una vez instalado se debe reiniciar Unity.
- Se encuentra en Windows -> Analysis -> Code Coverage.
- Al lanzarlo se indica que se debe cambiar al modo Debug.
- Al lanzar una aplicación desde el Play se debe dar a Start Recording y cuando se desee pausar el proceso a Stop Recording.
- Automáticamente se genera una carpeta con todos los scripts que se hayan querido incluir.
- En resumen, **Coverage** muestra hasta donde ha llegado en uno o varios script la ejecución de una aplicación y cuanta de código se ha ejecutado en concreto. Es una herramienta para modular código o para detectar problemas en ejecución en tiempo real.

Unity - Coverage



Unity - Coverage

Report

Compartir

Vista

« BossRoom » com.unity.multiplayer.samples.coop-1.2.0-pre » CodeCoverage » Report

	Nombre	Fecha de modificación	Tipo	Tamaño
	Cinemachine_CinemachineShotPlayable	19/05/2022 17:11	Microsoft Edge HT...	12 KB
js	Cinemachine_CinemachineSmoothPath	19/05/2022 17:11	Microsoft Edge HT...	117 KB
	Cinemachine_CinemachineStateDrivenCa...	19/05/2022 17:11	Microsoft Edge HT...	254 KB
sesion	Cinemachine_CinemachineStoryboard	19/05/2022 17:11	Microsoft Edge HT...	178 KB
	Cinemachine_CinemachineTargetGroup	19/05/2022 17:11	Microsoft Edge HT...	197 KB
	Cinemachine_CinemachineTouchInputMa...	19/05/2022 17:11	Microsoft Edge HT...	24 KB
	Cinemachine_CinemachineTrack	19/05/2022 17:11	Microsoft Edge HT...	26 KB
erson	Cinemachine_CinemachineTrackedDolly	19/05/2022 17:11	Microsoft Edge HT...	157 KB
	Cinemachine_CinemachineTransposer	19/05/2022 17:11	Microsoft Edge HT...	216 KB
	Cinemachine_CinemachineTriggerAction	19/05/2022 17:11	Microsoft Edge HT...	147 KB
	Cinemachine_CinemachineVirtualCamera	19/05/2022 17:11	Microsoft Edge HT...	281 KB
js	Cinemachine_CinemachineVirtualCamera...	19/05/2022 17:11	Microsoft Edge HT...	341 KB
	Cinemachine_CinemachineVolumeSettings	19/05/2022 17:11	Microsoft Edge HT...	159 KB
	Cinemachine_ConfinerOven	19/05/2022 17:11	Microsoft Edge HT...	273 KB
	Cinemachine_Damper	19/05/2022 17:11	Microsoft Edge HT...	126 KB
	Cinemachine_DocumentationSortingAttri...	19/05/2022 17:11	Microsoft Edge HT...	43 KB

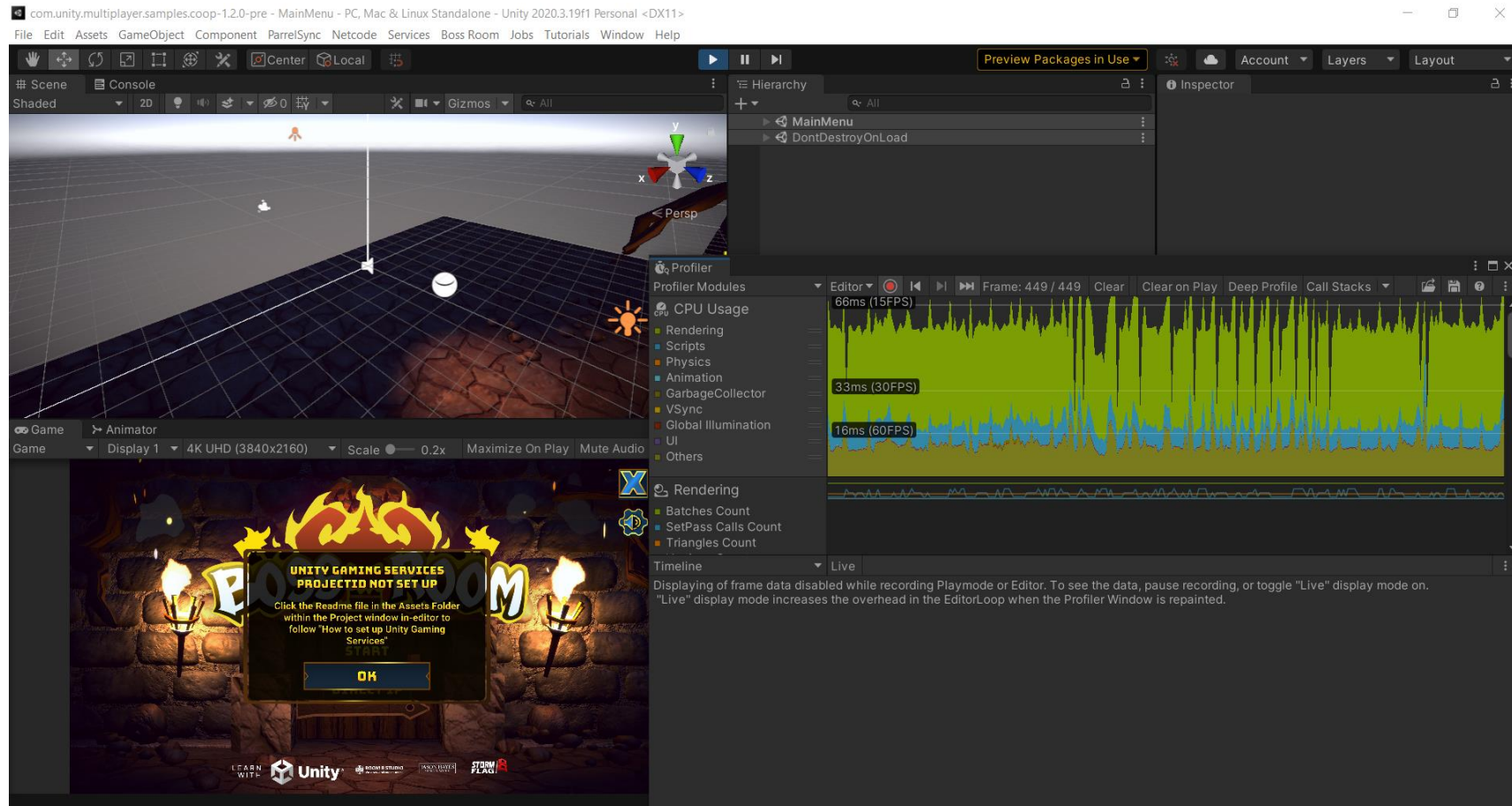
Unity - Coverage

< Summary	
Class:	UnityEngine.UI.InputField
Assembly:	UnityEngine.UI
File(s):	C:/Users/jonat/OneDrive/Escritorio/BossRoom/com.unity.multiplayer.samples.coop-1.2.0-pre/Library/PackageCache/com.unity.ugui@1.0.0/Runtime/UI/Core/InputField.cs
Covered lines:	42
Uncovered lines:	1507
Coverable lines:	1549
Total lines:	3294
Line coverage:	2.7% (42 of 1549)
Covered branches:	0
Total branches:	0
Covered methods:	10
Total methods:	161
Method coverage:	6.2% (10 of 161)
Coverage History	

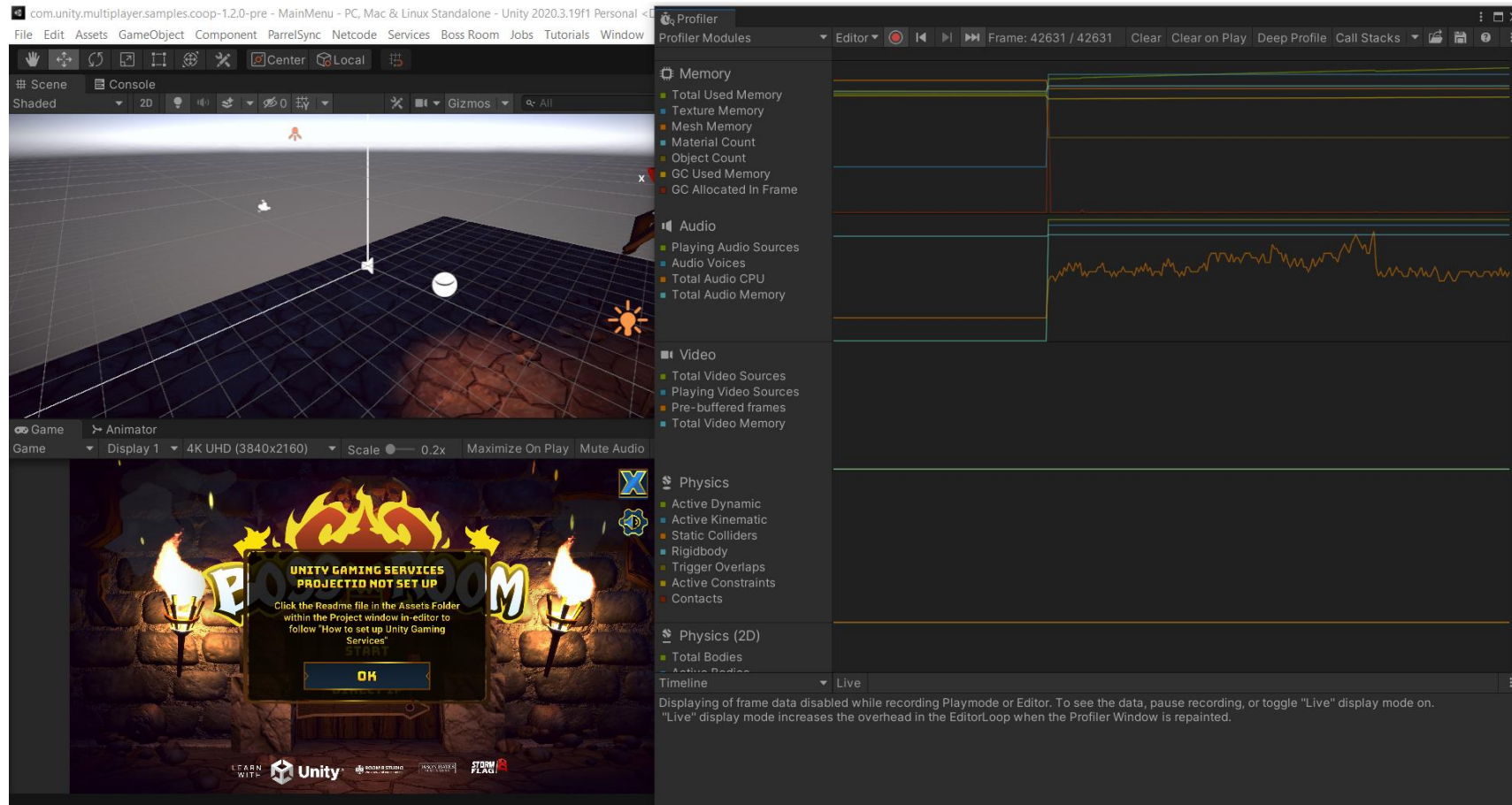
Metrics					
Method	Branch coverage ⓘ	Crap Score ⓘ	Cyclomatic complexity ⓘ	NPath complexity ⓘ	Sequence coverage ⓘ
InputField()	0%	0	0	0	0%
InputField()	0%	0	0	0	0%
SetTextWithoutNotify(...)	0%	0	0	0	0%
SetText(...)	0%	0	0	0	0%
ClampPos(...)	0%	0	0	0	0%
OnValidate()	0%	0	0	0	0%
OnEnable()	0%	0	0	0	0%
OnDisable()	0%	0	0	0	100%
CaretBlink()	0%	0	0	0	0%
SetCaretVisible()	0%	0	0	0	0%
SetCaretActive()	0%	0	0	0	0%
UpdateCaretMaterial()	0%	0	0	0	0%
OnFocus()	0%	0	0	0	0%
SelectAll()	0%	0	0	0	0%
MoveTextEnd(...)	0%	0	0	0	0%
MoveTextStart(...)	0%	0	0	0	0%
TouchScreenKeyboard ShouldBeUsed()	0%	0	0	0	0%
InPlaceEditing()	0%	0	0	0	0%
InPlaceEditingChanged ()	0%	0	0	0	0%
UpdateCaretFromKeyb oard()	0%	0	0	0	0%
LateUpdate()	0%	0	0	0	7.53%

Unity - Profiler

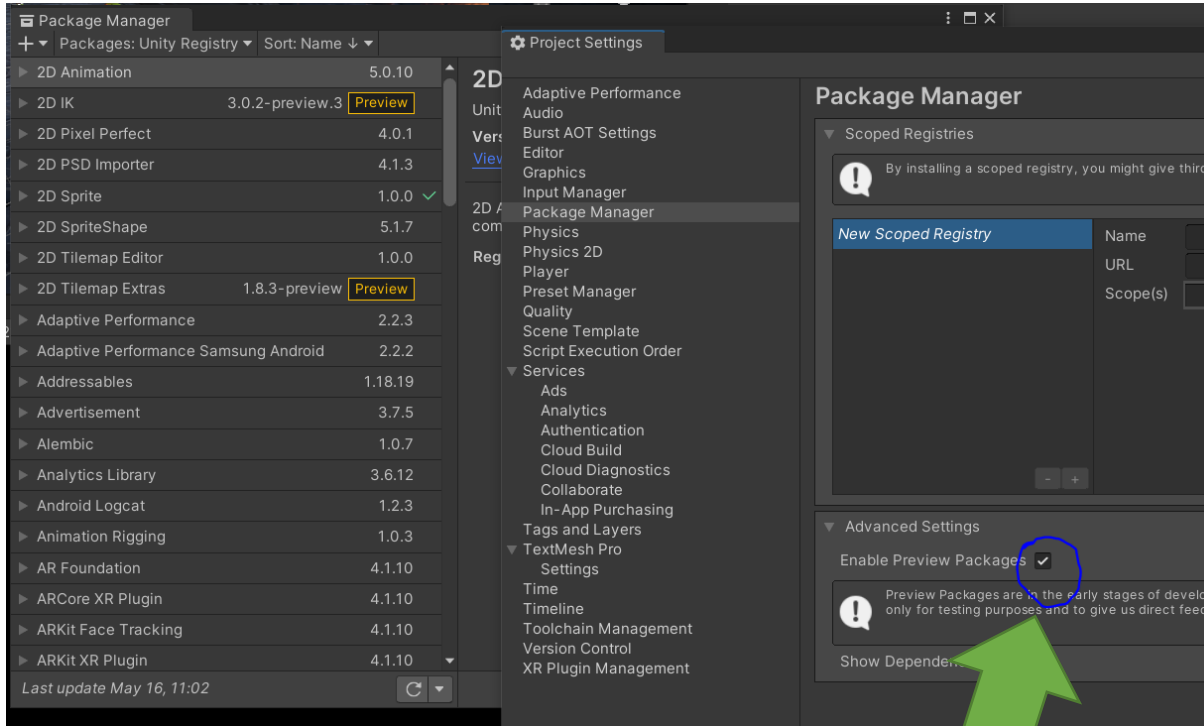
- Window – Analysis – Profiler
- Sirve para medir en tiempo de ejecución las métricas de rendimiento de CPU, Renderizado, Memoria, Audio, Video, Físicas, Iluminación, entre otros.



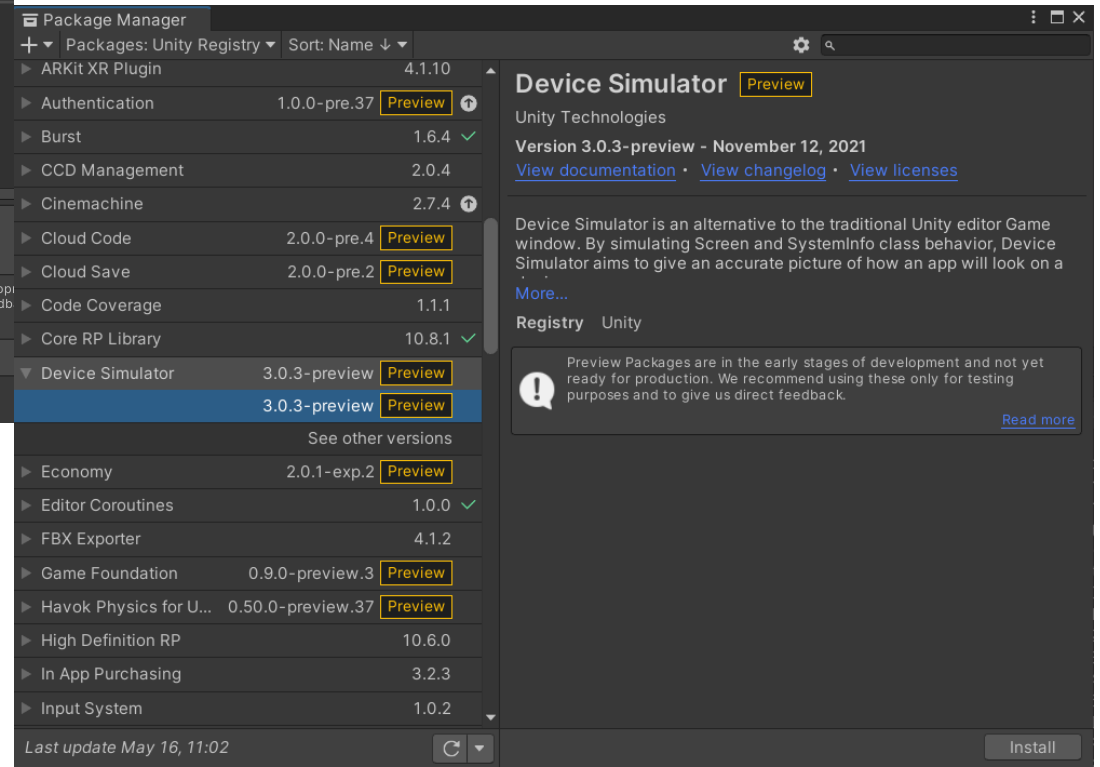
Unity - Profiler



Unity - Device Simulator



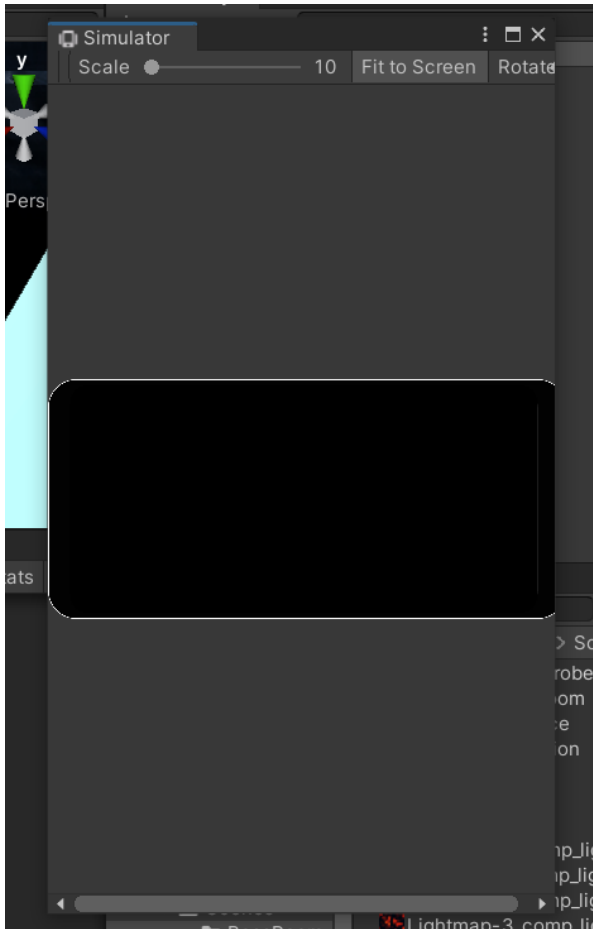
Versiones actuales de Unity ya aparece en la pestaña de Game del propio reproductor y cambiando a Simulator. Para las versiones antiguas de Unity sin actualizar habilitarlo tal y como aparece a continuación vía habilitar el Preview Packages en Project Settings – Package Manager



Unity - Device Simulator

- Window – General – Device Simulator

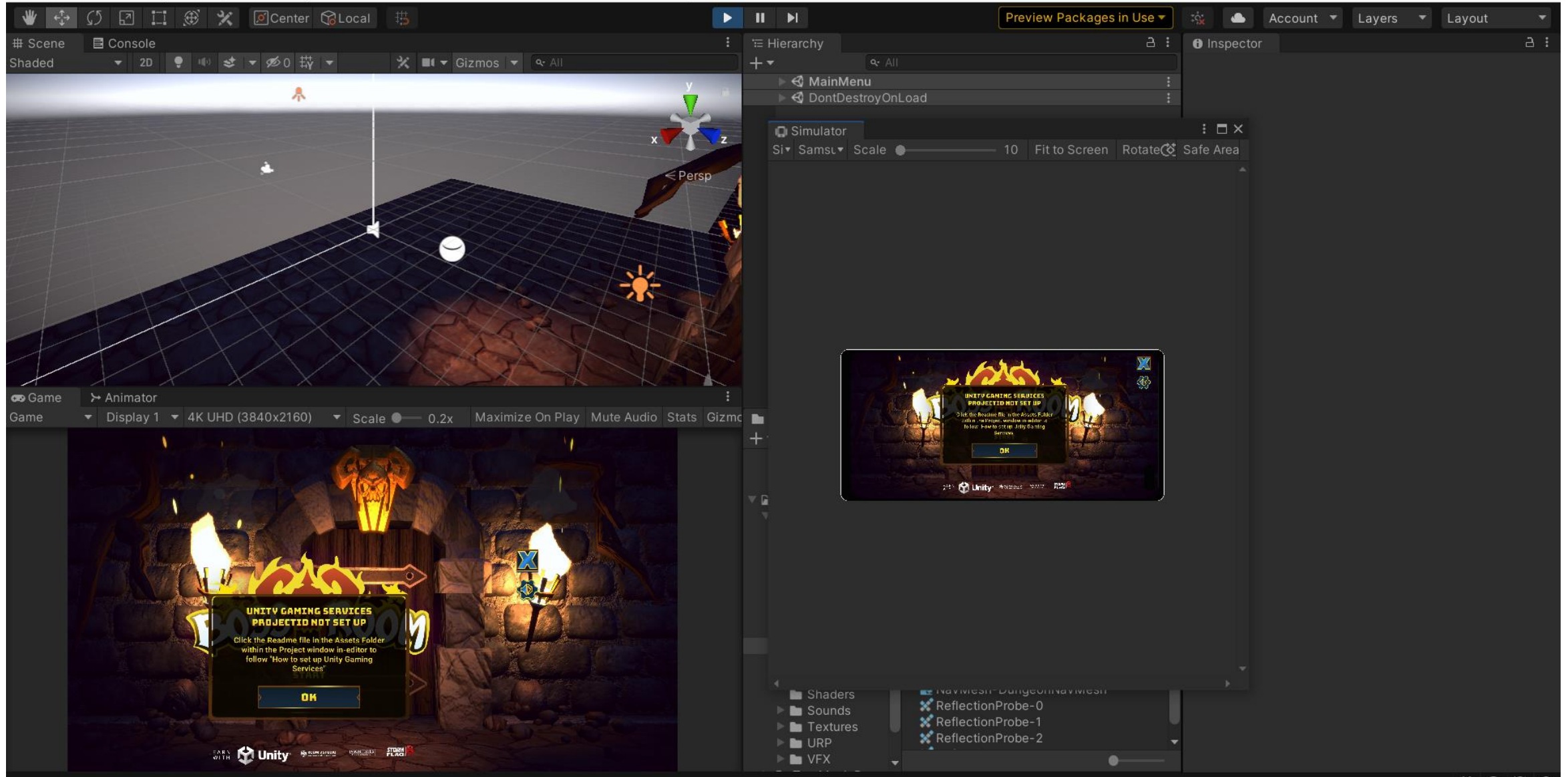
Nota: Darle a maximizar el Simulator para que os salga el panel con los diferentes tipos de dispositivos.



Unity - Device Simulator

com.unity.multiplayer.samples.coop-1.2.0-pre - MainMenu - PC, Mac & Linux Standalone - Unity 2020.3.19f1 Personal <DX11>

File Edit Assets GameObject Component ParrelSync Netcode Services Boss Room Jobs Tutorials Window Help



Documentación del guión

Un documento de diseño de videojuegos puede contener texto, imágenes, diagramas, arte conceptual, o cualquier contenido multimedia que logre ilustrar mejor las decisiones de diseño.

Algunos documentos de diseño pueden incluir prototipos funcionales o el motor de juego escogido para algunas de las secciones del juego.

A pesar de que se considera un requisito por muchas compañías, un GDD no tiene ninguna forma estándar de la industria. Por ejemplo, los desarrolladores pueden escoger mantener el documento como un documento de texto con formato, o realizarlo en una herramienta de colaboración en línea.

Documentación del guión (II)

Para que el creador haga esto en el desarrollo de un videojuego generalmente hace el siguiente proceso:

1. **Concepción de la idea del videojuego**
2. **Diseño**
3. **Planificación**
4. **Preproducción**
5. Producción
6. Pruebas
7. Mantenimiento

El proceso es similar a la creación de software en general, aunque difiere en la gran cantidad de aportes creativos (**música, historia, diseño de personajes, niveles, etc**) necesarios.

El desarrollo también varía en función de la plataforma objetivo (PC, móviles, consolas), el género (estrategia en tiempo real, RPG, aventura gráfica, plataformas, etc) y la forma de visualización (2D, 2.5D y 3D).

Cabe mencionar que el diseño de juegos es usualmente considerado un proceso de creación iterativo, esto quiere decir que los diseñadores tendrán que pasar por cada uno de estos pasos repetidas veces (cambiando y mejorando aspectos) hasta que consideren que el resultado sea el mejor.

Documentación del guión (III)

Concepción de la idea del videojuego.

Al tener una idea inicial en mente comienza esta etapa en la cual deberán plantear los aspectos fundamentales que conformarán el videojuego, entre los que se encuentran:

Género: dentro de qué géneros se va a desarrollar el juego. De no corresponder a un género muy conocido, se deben especificar las características.

Gameplay: lo que generará diversión a la hora de jugarlo.

Conceptos: algunas ideas sueltas acerca de cómo debe lucir el juego en cuanto a personajes, ambientación, música, etc.

Para el desarrollo de estos y para facilitar su creación se pueden utilizar métodos de pensamiento activo o lluvias de ideas, ya que es la etapa inicial, o conceptualización. Pueden usar técnicas simples como red o burbuja de palabras.

Documentación del guión (IV)

Diseño

En esta fase se detallan todos los elementos que compondrán el juego, dando una idea clara a todos los miembros del grupo desarrollador acerca de cómo son. Entre estos elementos tenemos:

Historia: forma en que se desenvolverán los personajes del juego y la historia del mundo (o un planeta en específico) representado. Casi todos los juegos tienen historia.

Guion: el proceso comienza con una reunión de todo el equipo de desarrollo, para que todo el equipo tenga la oportunidad de aportar sus ideas o sugerencias al proyecto. A partir de aquí el equipo de guion trabaja por conseguir un borrador en el que queden plasmados cuales serán los objetivos en el juego, las partes en las que se dividirá, el contexto en el que se desarrollará la acción, cuales y cómo serán los personajes principales del juego, etc.

Arte conceptual: se establece el aspecto general del juego. En esta etapa un grupo de artistas se encargan de visualizar o conceptualizar los personajes, escenarios, criaturas, objetos, etc. Estos artistas se basan en las ideas originales de los creadores y luego entregan una serie de propuestas impresas o digitales de cómo lucirá el juego. Posteriormente, el director de arte se encargará de escoger de entre las opciones aquellas que se apeguen más a la idea original. Algunas veces los artistas conceptuales permanecen durante todo el proceso de producción, pero lo usual es que sólo participen en las primeras etapas del proceso.

Sonido: detallada descripción de todos los elementos sonoros que el juego necesita para su realización. Voces, sonidos ambientales, efectos sonoros y música.

Mecánica de juego: es la especificación del funcionamiento general del juego. Es dependiente del género y señala la forma en que los diferentes entes virtuales interactuarán dentro del juego, es decir, las reglas que rigen este.

Diseño de programación: describe la manera en que el videojuego será implementado en una máquina real (PC, consola, móviles, etc) mediante un cierto lenguaje de programación y siguiendo una determinada metodología. Generalmente en esta fase se generan diagramas de UML que describen el funcionamiento estático y dinámico, la interacción con los usuarios y los diferentes estados que atravesará el videojuego como software.

De toda la fase de diseño es necesario generar un documento llamado Documento de Diseño, que contiene todas las especificaciones de arte, mecánicas y programación.

Documentación del guión (V)

Planificación

En esta fase se identifican las tareas necesarias para la ejecución del videojuego y se reparten entre los distintos componentes del equipo desarrollador. También se fijan plazos para la ejecución de dichas tareas y reuniones clave, con la ayuda de herramientas de diagramación de actividades como **GANTT y PERT**.

Preproducción

Durante la etapa de preproducción se le asigna el proyecto a un pequeño equipo, con la finalidad de verificar la factibilidad de la idea.

Este equipo trabajará para crear un nivel o ambiente del juego, acercándose lo más que se pueda al producto final. La preproducción es una de las partes más críticas del proceso ya que determinará la viabilidad del juego.

Documentación del guión (VI)

Producción

Aquí se llevan a cabo todas las tareas especificadas en la fase de planificación, teniendo como guía fundamental el documento de diseño. Esto incluye, entre otras cosas, la codificación del programa, la creación de sprites, tiles y modelos 3D, grabación de sonidos, voces y música, creación de herramientas para acelerar el proceso de desarrollo, entre otras.

Programación: la mayoría de los juegos se programan utilizando el lenguaje C++ dado que es un lenguaje de nivel medio que permite un rápido acceso a los componentes de hardware de una computadora o consola de juegos que lo hace más accesible.

Ilustración: los juegos 2D deben ser ilustrados por artistas experimentados, quienes trabajan tomando en consideración las limitaciones técnicas del hardware sobre el cual correrá el juego, esto incluye: cantidad de colores disponibles, tamaño de los sprites, resolución final de los sprites y formatos a utilizar. Los artistas 2D también son los encargados de elaborar las animaciones del juego.

Documentación del guión (VII)

Producción

Interfaz: es la forma en que se verán los elementos de la interfaz gráfica de usuario y el HUD, mediante los cuales el usuario interactuará con el juego.

Animación y modelado 3D: los artistas utilizan herramientas comerciales de modelado y animación tridimensional como 3DS Max, Maya, XSI/Softimage, Blender (el cual no es comercial), etc. Pero, además, usan herramientas desarrolladas internamente que facilitan algunas de las funciones más comunes del proceso de creación de juegos.

Sonido: los ingenieros de sonido se encargan de crear sonidos para cada objeto o personaje del juego. Pueden crear sonidos desde cero o utilizar sonidos del ambiente y modificarlos según sus necesidades.

Documentación del guión (VIII)

Al igual que en otros tipos de software, los videojuegos deben pasar en su desarrollo por una etapa donde se corrigen los errores inherentes al proceso de programación y se asegura su funcionalidad. Además, a diferencia de aquellos, los videojuegos requieren un refinamiento de su característica fundamental, la de producir diversión de manera interactiva (jugabilidad). Generalmente, esta etapa se lleva a cabo en tres fases:

Pruebas físicas: se llevan a cabo por los diseñadores y programadores del juego. Se crean prototipos que simulan los eventos que pueden suceder en el juego. Un prototipo físico puede utilizar papel y lápiz, tarjetas de índice, o incluso ser actuado fuera. Sobre la base de los resultados de estas pruebas se puede hacer una mejor aproximación al balance del videojuego, pueden prevenir problemas de programación. El objetivo es jugar y perfeccionar este simplista modelo antes de que un solo programador, productor o artista gráfico estén cada vez más introducidos en el proyecto. De esta manera, el diseñador del juego recibe retroalimentación instantánea en lo que piensan los jugadores del juego y pueden ver inmediatamente si están logrando su metas.

Pruebas alpha: se llevan a cabo por un pequeño grupo de personas, que con anterioridad estén involucradas en el desarrollo, lo que puede incluir artistas, programadores, coordinadores, etc. El propósito es corregir los defectos más graves y mejorar características de jugabilidad no contempladas en el documento de diseño.

Pruebas beta: estas pruebas se llevan a cabo por un equipo externo de jugadores, bien sea que sean contratados para la ocasión o que sean un grupo componente del proyecto (grupo QA). De estas pruebas, el videojuego debe salir con la menor cantidad posible de defectos menores y ningún defecto medio o crítico.

Documentación del guión (IX)

Una vez que **el juego alcanza su versión final (RTM) y se publica**, aparecerán nuevos errores o se detectarán posibles mejoras. Es necesario recopilar toda la información posible de los jugadores y a partir de ahí realizar los cambios oportunos para mejorar el juego en todos sus aspectos, ya sea de diseño, jugabilidad, etc.

Estas correcciones o mejoras se hacen llegar a los usuarios en forma de parches o actualizaciones, que en ocasiones pueden incluir algunas características nuevas para el juego.

En ocasiones, los desarrolladores van más allá -especialmente si el videojuego ha funcionado bien comercialmente- y realizan una ampliación considerable en los contenidos o en las fases del videojuego que se pone a la venta normalmente a un precio inferior al del juego original y que se conoce como una expansión.

Bibliografía

- <https://es.wikipedia.org/wiki/Multimedia>
- <https://docs.unity3d.com/Manual/TimeFrameManagement.html>
- [https://es.frwiki.wiki/wiki/Test \(jeu vid%C3%A9o\)](https://es.frwiki.wiki/wiki/Test_(jeu_vid%C3%A9o))
- <https://unity.com/es/demos/small-scale-coop-sample>
- [https://es.wikipedia.org/wiki/Documento de dise%C3%B1o de vide
ojuegos](https://es.wikipedia.org/wiki/Documento_de_dise%C3%B1o_de_videojuegos)