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

DGPP – Curso Especialización Desarrollo Videojuegos – AGL – Curso 21/22

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

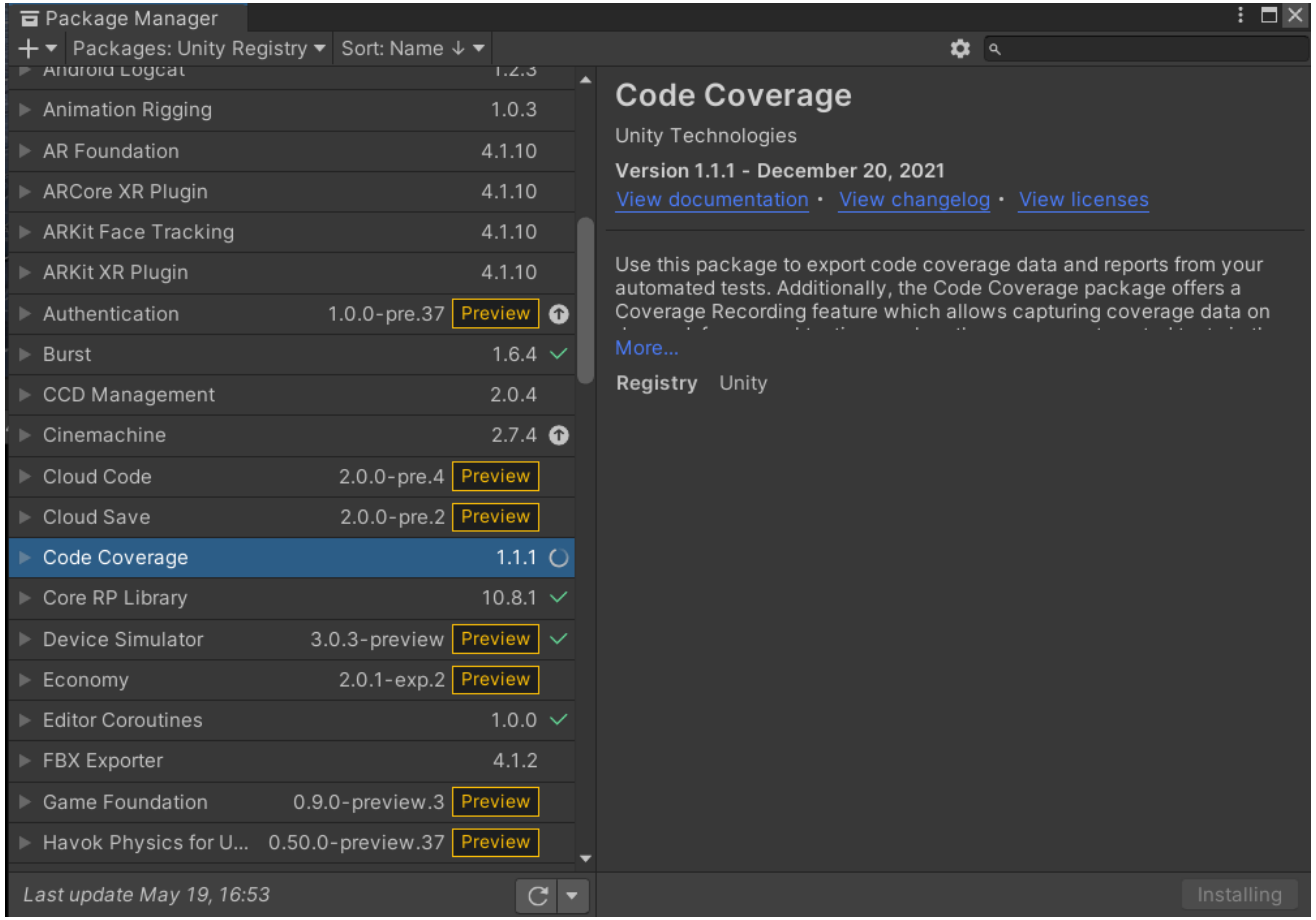
Arreglo del fallo

- 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

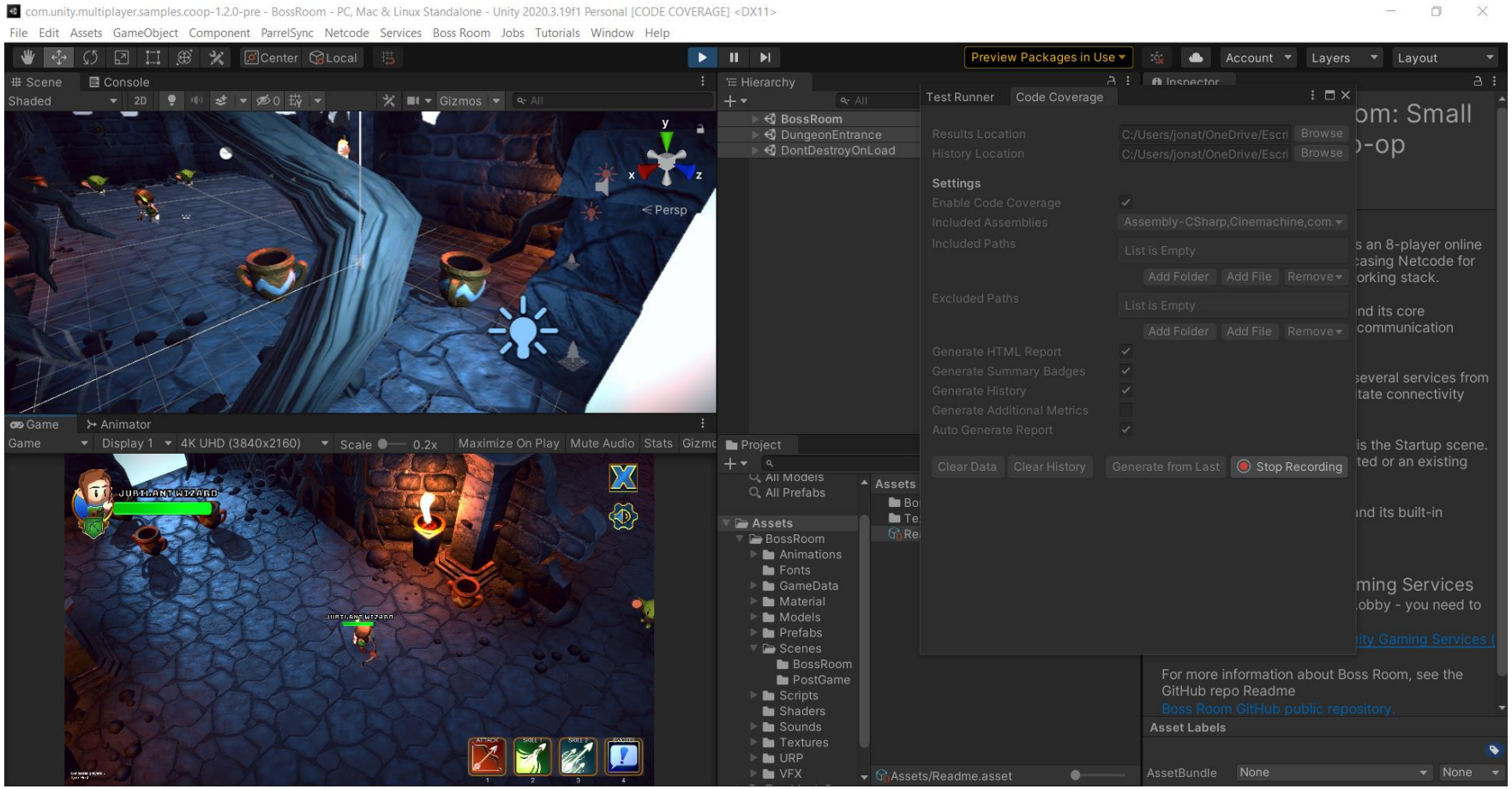
Unity - Coverage



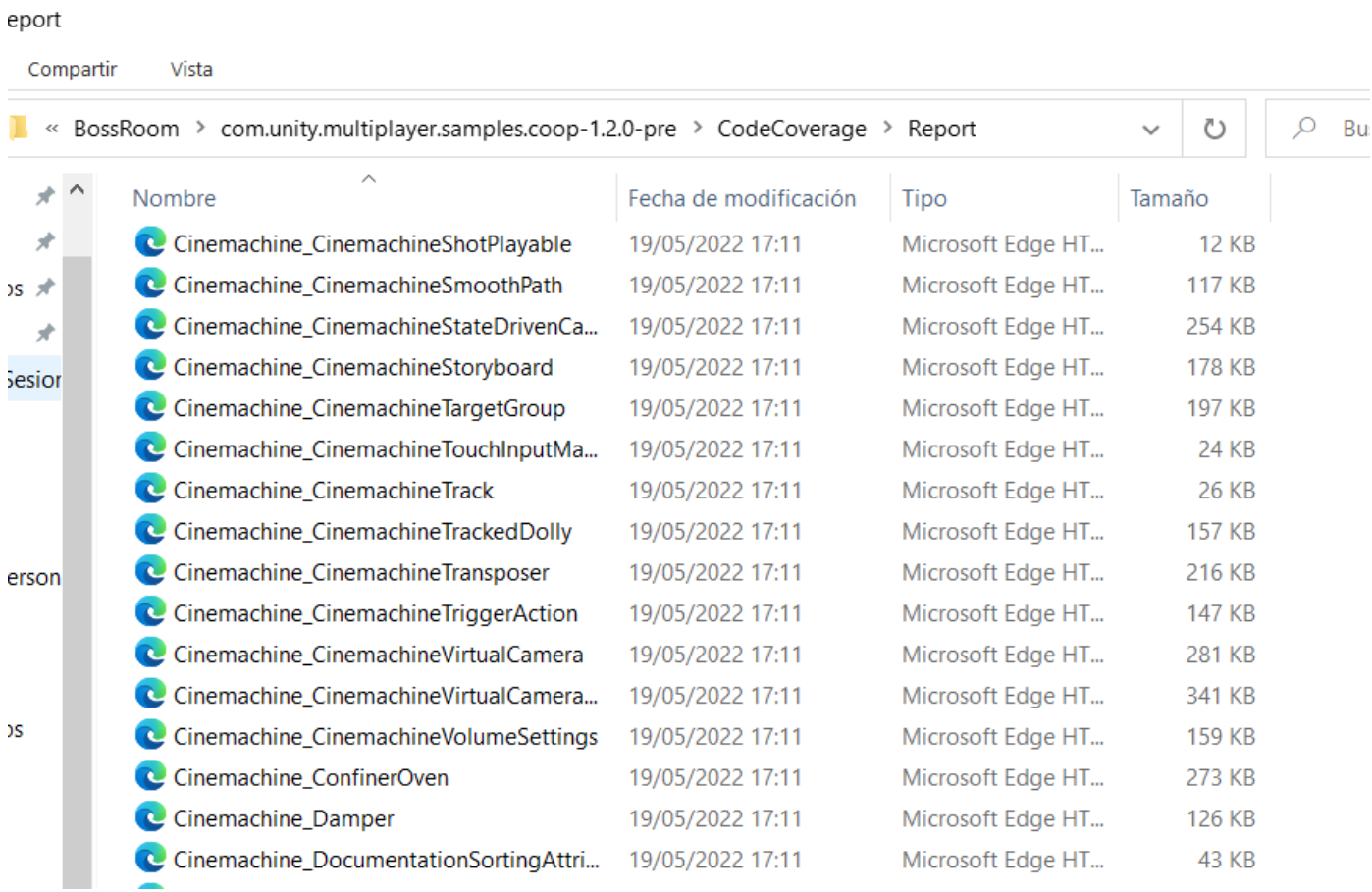
Unity - Coverage

- 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



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)

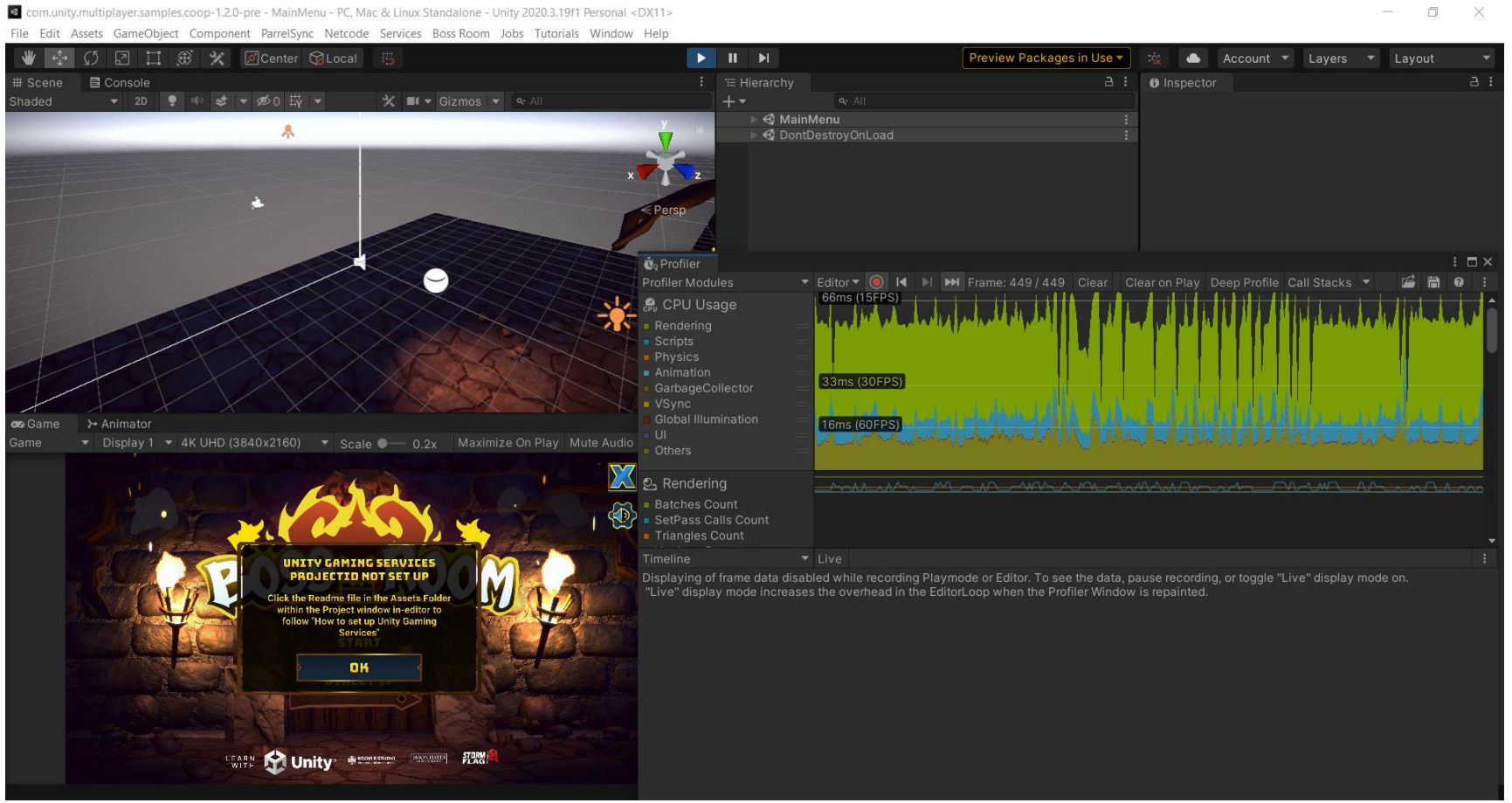
Coverage History

Metrics

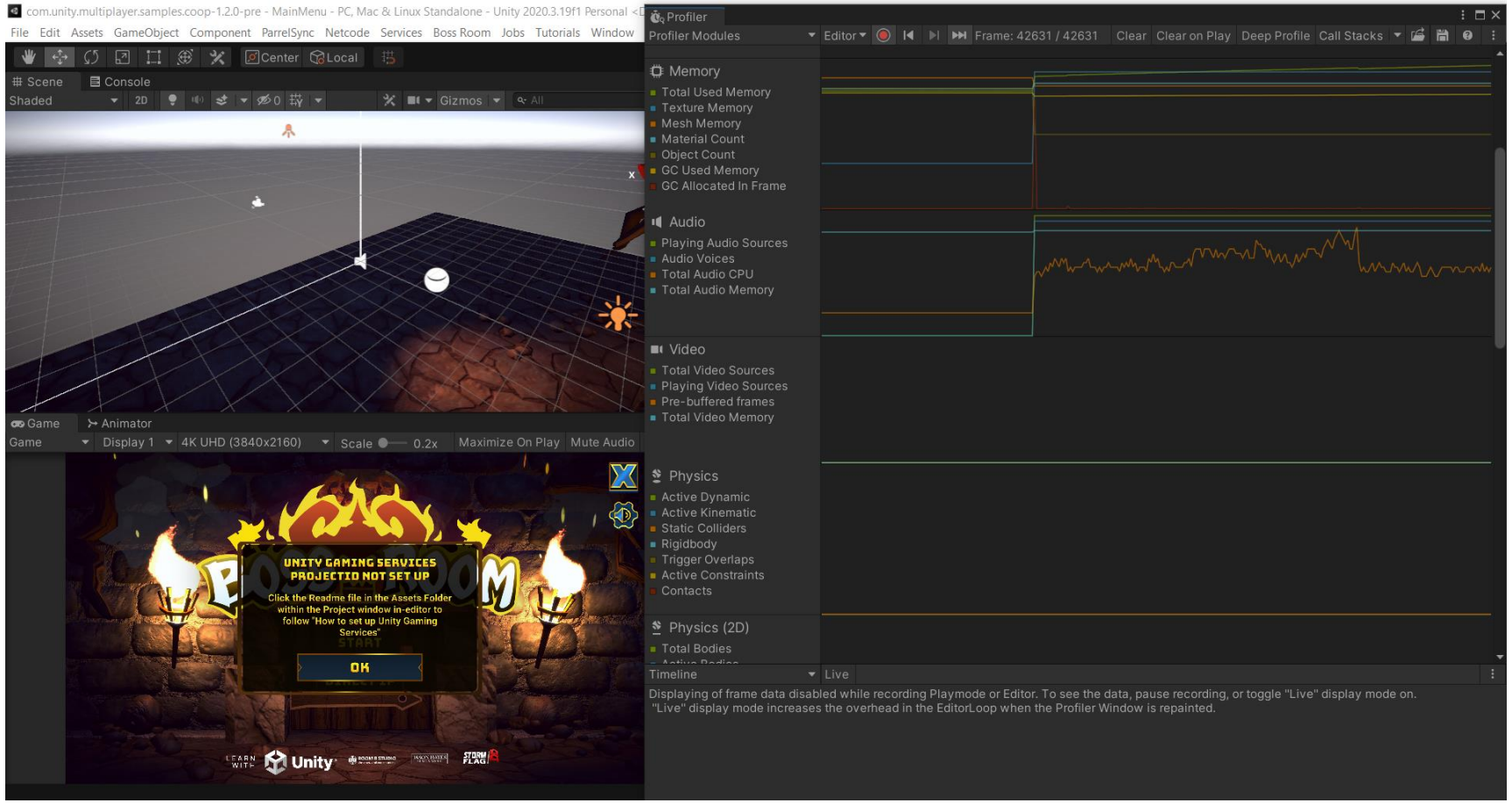
Method	Branch coverage	Crap Score	Cyclomatic complexity	NPath complexity	Sequence coverage
InputField()	0%	0	0	0	0%
InputField()	0%	0	0	0	0%
SetTextWithoutNotify(...)	0%	0	0	0	0%
SetText(...)	0%	0	0	0	0%
ClampPos(...)	0%	0	0	0	0%
OnValidate()	0%	0	0	0	0%
OnEnable()	0%	0	0	0	0%
OnDisable()	0%	0	0	0	100%
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%
TouchScreenKeyboardShouldBeUsed()	0%	0	0	0	0%
InPlaceEditing()	0%	0	0	0	0%
InPlaceEditingChanged()	0%	0	0	0	0%
UpdateCaretFromKeyboard()	0%	0	0	0	0%
LateUpdate()	0%	0	0	0	7.53%

Unity - Profiler

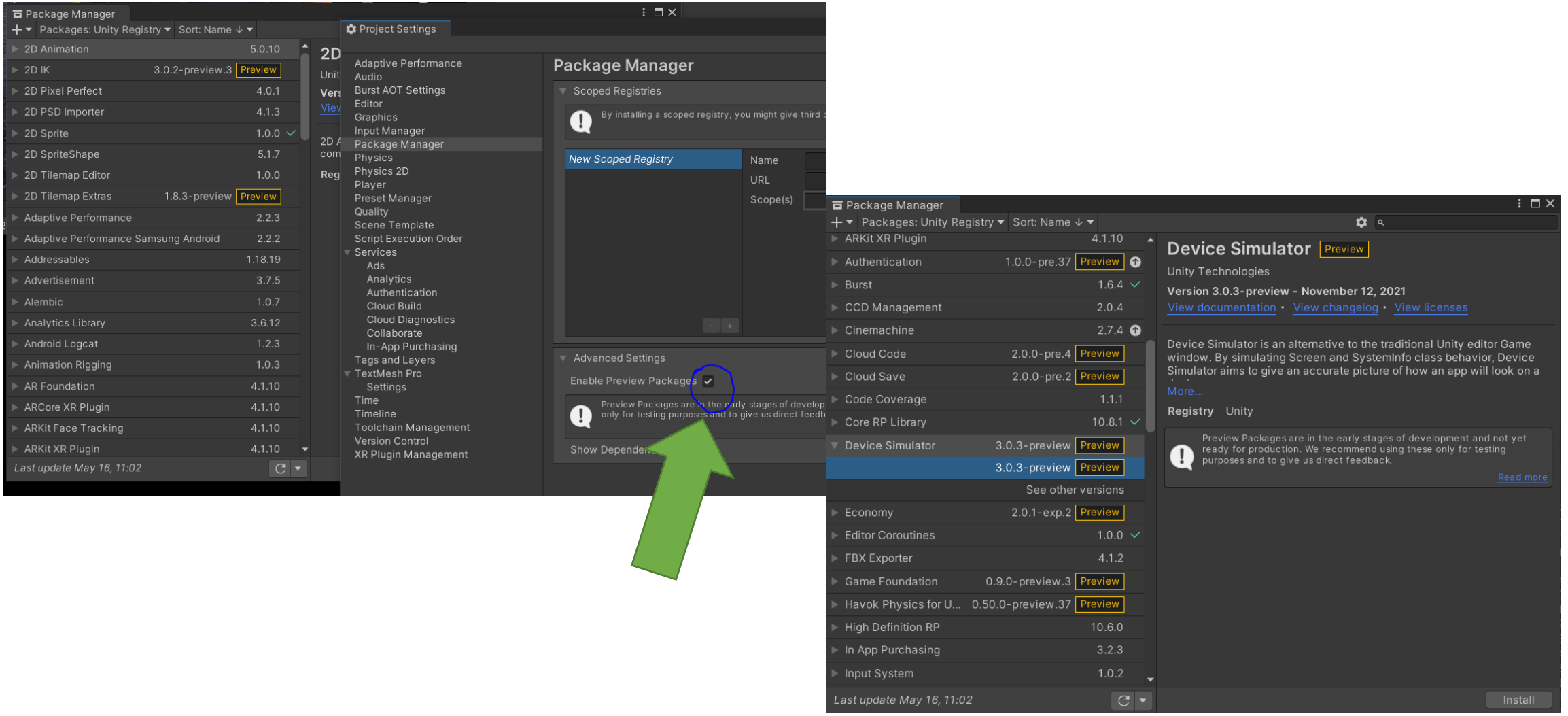
- Window – Analysis – Profiler
- Sirve para medir en tiempo de ejecución las métricas de rendimiento de CPU, Renderizado, Memoria, Audio, Video, Físicas, Iluminación, entre otros.



Unity - Profiler

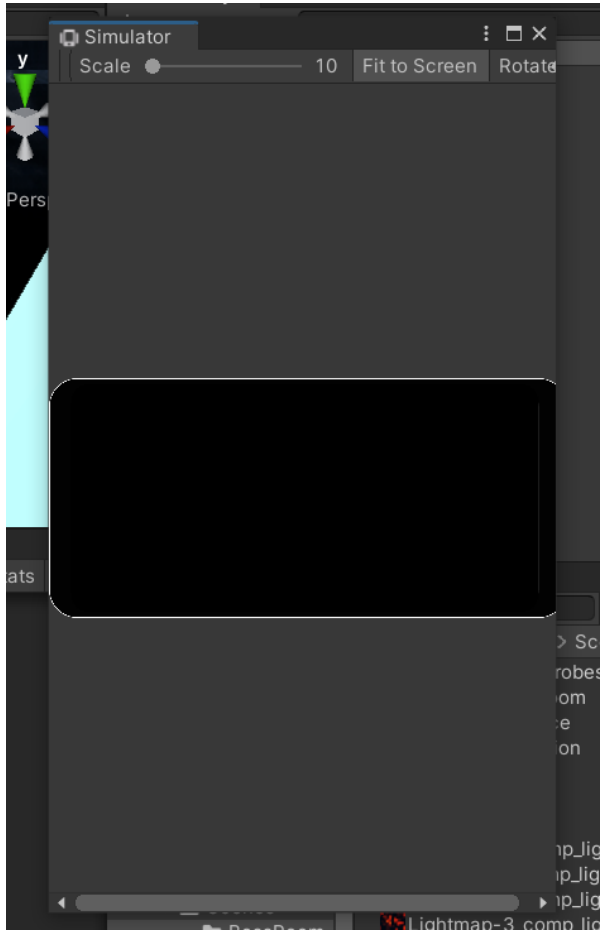


Unity - Device Simulator

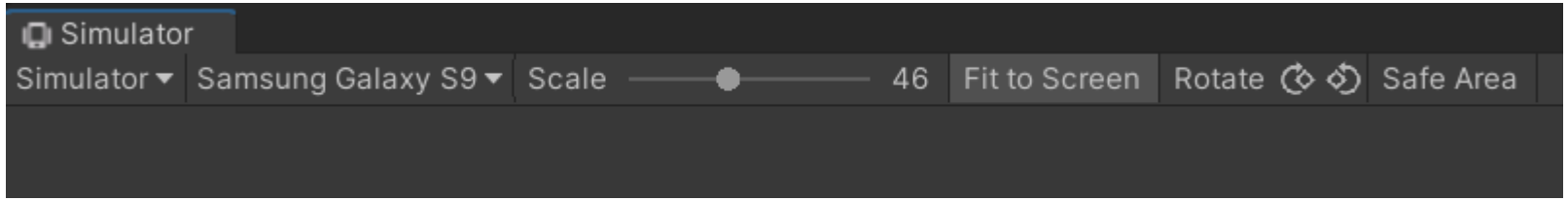


Unity - Device Simulator

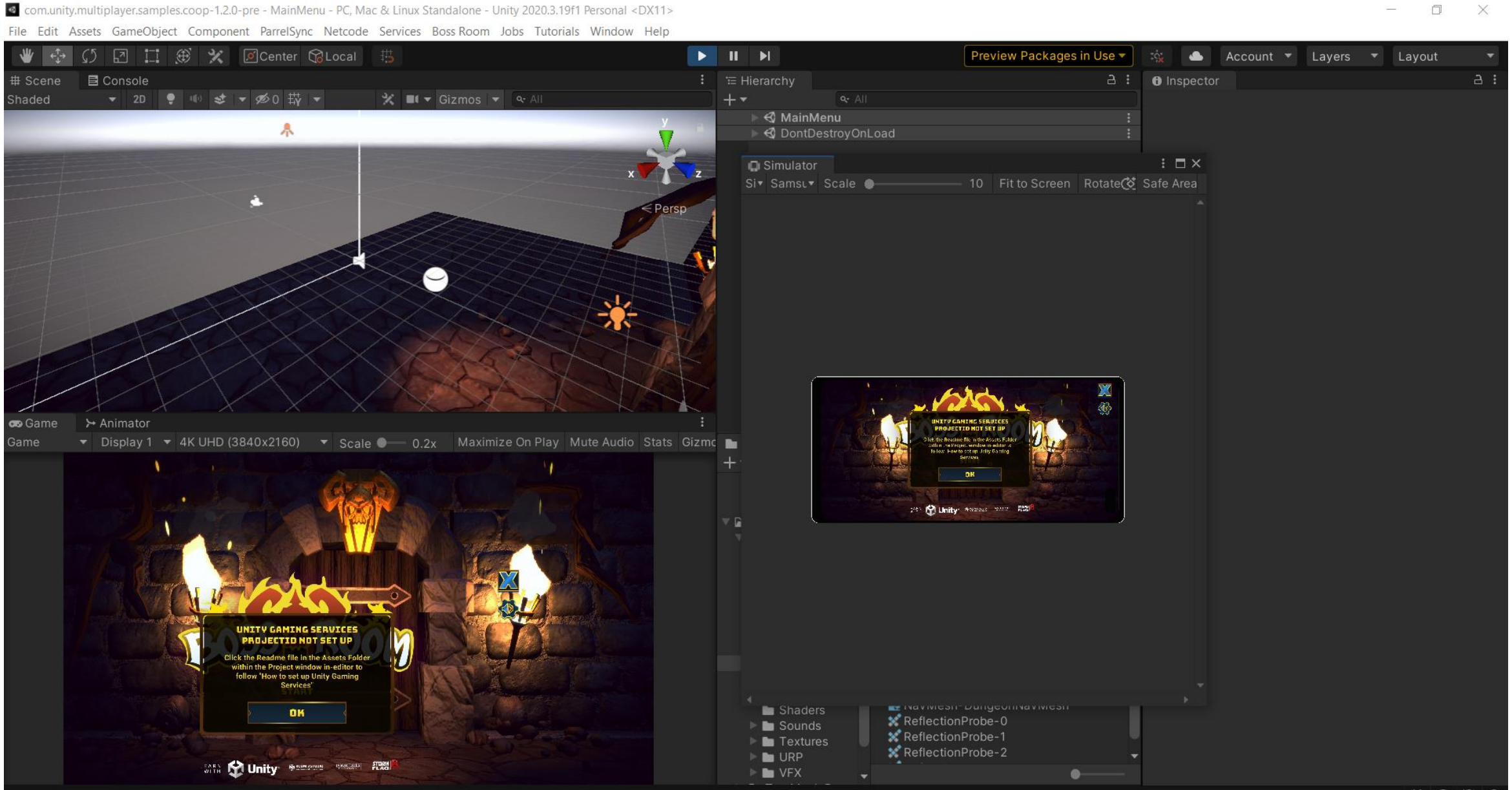
- Window – General – Device Simulator



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



Unity - Device Simulator



Referencias

- https://es.frwiki.wiki/wiki/Test_(jeu_vid%C3%A9o)
- https://unity.com/es/demos/small-scale-coop-sample