UNIVERSAL RENDER PIPELINE - BUILT-IN RENDER PIPELINE

COMPARACIÓN DE ILUMINACIONES EN DOS DIMENSIONES

GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS

ESNE
Escuela Universitaria
de Diseño, Innovación
y Tecnología

DAVID MARTÍN ALMAZÁN

21 JUNIO 2021

Índice

Índice de figuras

Dedicatorias y agradecimientos

Agradecer en este trabajo a todos los profesores de la universidad ESNE por brindarme su apoyo a la hora de realizar este trabajo, en especial a Ángel, Luis Rubio, Fernando y Sergio Urbano.

En estos últimos cuatro años he conocido a personas que me han apoyado tanto en las buenas como en las malas situaciones y que sin ellos no podría haber llegado a donde estoy. Hoy en día, muchas de esas personas las considero de la familia, y espero que en un futuro muy lejano nos podamos dedicar a lo que nos gusta, desarrollar videojuegos juntos.

Dedicar este trabajo a mi grupo de amigos de toda la vida, Ortiz, Kike y Costy por poner el hombro cuando lo necesito, y por apoyarme en cada proyecto que realizo.

Por último, dedicar este trabajo a mi familia, que siempre me ha apoyado en cada paso que he realizado en la universidad, por interesarse por cada proyecto que salía públicamente. A mí tía Maribel, que en paz descanse, que me enseño la valiosa lección de no rendirme y vivir cada minuto de la vida, y, por último, a mis padres, el mayor pilar que tengo en la vida. Ellos siempre están a mi lado cuando estoy por rendirme y siempre me levantan, los primeros con los que siempre celebró cada victoria y gracias a ellos he podido tener la posibilidad de tener una vida universitaria y poder haber estudiado en esta universidad.

Resumen

Algo muy importante a la hora de hablar de la iluminación en dos dimensiones de Unity, es que antiguamente se usaba una iluminación en tres dimensiones sobre gráficos en dos dimensiones, lo que daba la sensación de que era en dos dimensiones. Pero con la llegada del Universal Render Pipeline, y su iluminación propia de la iluminación en dos dimensiones nos da una iluminación mucho más realista, y que consume mucho menos.

Aun así, ¿es mejor que la antigua iluminación en dos dimensiones?

Palabras Clave: Universal Render Pipeline, Built-in Render Pipeline, Iluminación 2D, Post-Procesado, Rendimiento, Facilidad, Comparación, Diferencia.

Abstract

Something very important when talking about Unity's two-dimensional lighting is that in the past, three-dimensional lighting was used on two-dimensional graphics, which gave the impression that it was two-dimensional. But with the advent of the Universal Render Pipeline, and its own two dimensional lighting it gives us a much more realistic lighting, and it consumes much less.

Still, is it better than the old 2D lighting?

Keywords: Universal Render Pipeline, Built-in Render Pipeline, 2D Lighting, Post-Processing, Performance, Ease, Comparison, Difference.

Introducción

Antiguamente se usaba la iluminación en tres dimensiones sobre gráficos en dos dimensiones, haciendo creer que era una iluminación en dos dimensiones. Con la llegada del Universal Render Pipeline, trajo consigo una iluminación propia para gráficos en dos dimensiones. Al no haber suficientes estudios, la investigación recopila todos los aspectos del render que se utilizaba en los últimos años para realizar la iluminación en tres dimensiones sobre gráficos en dos dimensiones y el Universal Render Pipeline. Al igual que sus diferencias en características a la hora de la iluminación, el rendimiento que generan en la CPU y GPU y, por último, la facilidad a la hora de usar cada render.

Con esta idea de investigación se propone una ayuda a los desarrolladores sobre el uso de un render determinado para una iluminación en dos dimensiones. Se tratarán temas como los efectos de post-procesado o los distintos tipos de luz que tiene cada render. Además, se utilizarán varias herramientas que tiene el motor gráfico de Unity para visualizar el rendimiento que realiza la iluminación sobre la GPU y CPU.

La idea de investigación se divide en tres partes fundamentales:

- Los distintos aspectos de la iluminación propia del Universal Render Pipeline, donde se explican los distintos filtros de post-procesado tales como Vignette, Bloom, etc. que son filtros que afectan sobre la iluminación. Además, se comentarán los distintos tipos de luz: Point Light, Parametric Light, etc.

- Los distintos aspectos de la iluminación propia del Built-in Render Pipeline, donde se explican los distintos filtros de post-procesado tales como Tonemapping, Color Grading, etc. que afectan a la iluminación de la escena, y se definen los distintos tipos de luz que trae este render: Spot Light, Global Light, etc.

- Las distintas pruebas que se han hecho para comparar los dos renders: Built-in y Universal Render Pipeline. Estas pruebas consisten en: el rendimiento que hacen ambos renders sobre la CPU y GPU a través del uso de estadísticas que nos proporciona el motor gráfico de Unity, la facilidad de uso mediante el número de pasos a seguir, y por último las características que tiene cada render respecto a la iluminación como por ejemplo el número de luces presente en escena.

Objetivos generales

Uno de los objetivos principales de la investigación es entrar en detalle con los apartados relacionados con la iluminación en cada render.

Con el Universal Render Pipeline, se comenta los distintos filtros de post-procesado relacionado con la iluminación, es decir, que tenga un carácter que influya sobre esta tales como la intensidad, color de las sombras, de los tonos medios y de las luces, calidad (si es mucha la calidad que queremos dar nos dará un aumento en el rendimiento de la CPU y GPU) y, por último, si se le quiere dar una textura granulado a la escena. También se comenta los distintos tipos de luz como son: Point Light, Parametric Light, Freeform Light, Sprite Light y Global Light con sus propias propiedades y características.

Con el Built-in Render Pipeline, se describen los distintos filtros de post-procesado relacionado con la iluminación, es decir, que influye sobre propiedades como la calidad, intensidad, color de las sombras, tonos medios y luces, o saturación. También se comentan los distintos tipos de luz como son: Point Light, Spot Light, Directional Light, Area Light con sus propiedades y características.

Los filtros de post-procesado y tipos de luz de ambos renders estarán apoyados con una imagen de referencia para que el lector tenga una clara idea de cómo se ven en escena y cómo funcionan.

Por último, comentar que el objetivo más importante es la comparación entre los dos renders dichos anteriormente en varios aspectos importantes a la hora de realizar un videojuego, tales como el rendimiento que generan en la CPU y GPU, la facilidad a la hora de usar los modelos de iluminación de cada render, y las características que influyen en la iluminación que se diferencian de cada render, de esta manera, el lector tiene una mejor idea de que modelo de iluminación es más recomendable para su videojuego.

Marco Conceptual y contextual: el estado del arte

Universal Render Pipeline - Built-in Render Pipeline

Comparación de iluminaciones en dos dimensiones

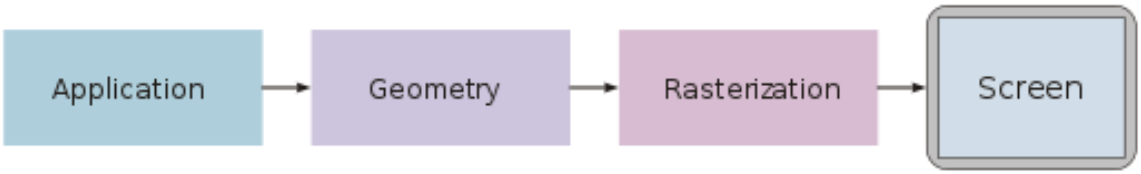
ÍNDICE

Introducción

La iluminación es un aspecto muy importante en los juegos hoy en día, pues es un componente que influye mucho a la hora de darle un sentimiento a la escena.

En el motor gráfico de Unity, existen varios tipos de renders pipeline cuya función es realizar una serie de operaciones que toman el contenido de la escena y lo muestran por pantalla. Entre estas operaciones se encuentran Culling, Rendering y Post-procesado. Culling es la función que desactiva la representación de objetos cuando la cámara no los ve, esta propiedad coge más peso cuando el proyecto es en 3D. La función de Rendering es hacer que un objeto aparezca por pantalla. Y, por último, el post-procesado, que trae una serie de efectos de pantalla que mejoran la apariencia del juego/aplicación con poco tiempo de configuración.

El render pipeline es la renderización basada en la implementación hardware de gráficos. La estructura que sigue la canalización de gráficos es:



1.- Aplicación: este paso lo ejecuta el software en el procesador principal, la CPU. Un ejemplo de las tareas que realiza es la detección de colisiones, animación, transformación, etc., también se utiliza para reducir la cantidad de memoria principal en un momento dado.

2.- Geometría: responsable de la mayoría de las operaciones con polígonos y sus vértices. Dentro de este paso, nos encontramos la transformación del modelado y la cámara, la iluminación donde la geometría es iluminada según la ubicación de las fuentes de luz, reflejo y otras propiedades de las superficies. La implementación más común, procesa la iluminación sólo en los vértices de los polígonos, esto se conoce como la iluminación por vértice. Después nos encontramos con la etapa de la proyección, donde la geometría es transformada en el espacio visual de la cámara de renderizado a un espacio de imagen 2D, la etapa clipping donde las primitivas geométricas que quedan fuera del polígono de visión no serán visibles y descartadas del escenario. Este paso acelera el proceso de renderización. Por último, tenemos la transformación ventana-ventana gráfica, para enviar la imagen a cualquier área de destino de la pantalla.

3.- Rasterización: proceso donde la representación de la escena en el espacio de la imagen en 2D es convertida a un formato ráster de imagen 2D y se determina el correcto valor de cada píxel resultante. Para evitar que el usuario vea la rasterización gradual de las primitivas se realiza un almacenamiento en búfer doble. Se realiza en un área de memoria espacial, y una vez la imagen se ha rasterizado por completo, se copia en el área visible de la memoria de imagen.

4.- Visualización: los píxeles coloreados se muestran por pantalla.

Antiguamente, para la iluminación sobre gráficos en dos dimensiones se usaba el render que venía por defecto Built-in Render, que aplicaba una iluminación en tres dimensiones sobre estos gráficos. Con el paso del tiempo, han ido apareciendo varios renders como HDR, o Universal Render Pipeline. Este último, trajo consigo una iluminación en dos dimensiones. Pero realmente ¿cuál es más rentable usar, el Built-in Render, o Universal Render Pipeline? ¿Cual afecta menos al rendimiento de la CPU y GPU? ¿En que se diferencian sus componentes de iluminación?

Iluminación 2D propia de Universal Render Pipeline

Universal Render Pipeline, antiguamente conocido como Lightweight Render Pipeline (LWRP), es una arquitectura de renderización, que es rápida y fácil de personalizar. Permite crear gráficos optimizados haciendo uso de las últimas APIs gráficas disponibles y está disponible a través de una amplia gama de plataformas. Una de las peculiaridades que tiene este render es el uso de este en el nuevo motor de iluminación 2D de Unity.

Beneficios de rendimiento de Universal Render Pipeline

El Universal Render Pipeline es ideal para desarrolladores de todos los niveles del espectro de experiencia en gráfica. Para los menos experimentados que no ven la necesidad de esta flexibilidad adicional que trae el render,

pueden usarlo sin modificaciones. Es decir, la mayor ventaja que tiene es que se adapta al usuario que lo utilice.

Como se ha dicho con anterioridad, y más tarde entraremos en detalle, una de las características más importantes dentro del motor de iluminación 2D es que este render, utiliza un renderizado hacia delante de una sola pasada, lo que reduce significativamente el costo de la CPU en comparación con otros renders que pasan múltiples veces.

Shaders del Universal Render Pipeline

El propio render trae consigo diversos tipos de shaders entre los que se encuentran:

- Lit: según Unity, este material permite renderizar superficies del mundo real como piedra, madera, vidrio, plástico y metales con calidad fotorrealista. Sus niveles de luz y reflejos se ven realistas y reaccionan adecuadamente en diversas condiciones de iluminación, por ejemplo, luz solar brillante o una cueva oscura. Este shader utiliza el modelo de sombreado más pesado en Universal Render Pipeline. Se puede ver un ejemplo a continuación en la ilustración 1.

Ilustración 1: Material Lit

- Simple Lit: según Unity, este tipo de material se utiliza cuando nos importa más el rendimiento que el fotorrealismo ya que utiliza una simple aproximación de la iluminación. Se puede ver un ejemplo a continuación en la ilustración 2.

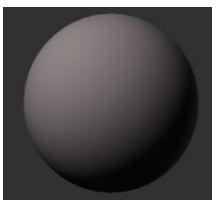


Ilustración 2: Material Simple Lit

- Baked Lit: según Unity, se utiliza este material para juegos o aplicaciones estilizadas ya que solo requiere de iluminación horneada a través de

mapas de luz y sondas de luz. No utiliza un sombreado basado en la física y no tiene iluminación en tiempo real, y al eliminarse el código de sombreado, se hace más rápido de calcular. Se puede ver un ejemplo a continuación en la ilustración 3.

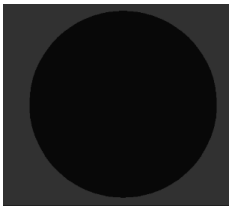


Ilustración 3: Material Baked Lit

- Particles Lit: según Unity, este material sirve para usar partículas casi fotorrealistas, como, por ejemplo: partículas de fogata, gotas de lluvia o humo de antorchas. Este shader produce unas imágenes más realistas por lo que utiliza un modelo de sombreado más pesado, lo que afecta el rendimiento. Se puede ver un ejemplo a continuación en ilustración 4.

Ilustración 4: Material Particles Lit

- Unlit: según Unity, este material se utiliza para efectos u objetos únicos en sus imágenes que no necesitan iluminación. Se puede ver un ejemplo a continuación en la ilustración 5.

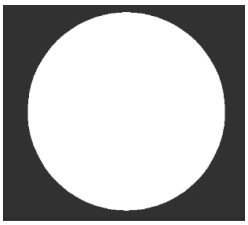


Ilustración 5: Material Unlit

Post-Procesado usado por Universal Render Pipeline

El post-procesado aplica filtros y efectos de pantalla, simulando una cámara. Funciona de una manera que se dibuja en la pantalla o se captura como una textura, un búfer de imagen antes de que la imagen aparezca en la pantalla. Puede utilizar efectos para simular las propiedades físicas de la cámara y de la película.

Más centrado en el Universal Render Pipeline, este incluye su propia solución de post-procesado, que Unity instala cuando instalas el package de URP. Los efectos que trae Universal Render Pipeline son:

Bloom

Según la API de Unity, consiste en un efecto que realza los bordes de las áreas más brillantes de la imagen.

Las propiedades de este efecto son: Intensity (valor que representa la fuerza del filtro), Threshold (valor que filtra los píxeles por debajo del nivel de intensidad), Scatter (radio del efecto), Tint (color del brillo), Clamp (valor que representa la intensidad máxima para calcular este efecto), High Quality Filtering (booleano para reducir el parpadeo y mejorar la suavidad general a cambio de consumir más recursos) y, por último, Skip Iterations (número de iteraciones finales a omitir).

Propiedad	Función
Intensity	Fuerza del filtro
Threshold	Establece el valor de brillo mínimo al que el Universal Render Pipeline aplica Bloom. Por debajo de este efecto no se le aplicará este efecto. No hay un valor máximo.
Scatter	Radio en el que expande la iluminación. Rango de 0 a 1.
Tint	Color del brillo de Bloom.
Clamp	Intensidad máxima que usa Unity para calcular Bloom.
High Quality Filtering	Habilitar para utilizar un Super Sampling[1]. Esto reduce el parpadeo y mejora la suavidad

	general, pero consume más recursos y puede afectar al rendimiento.
Skip Iterations	Número de iteraciones finales a omitir. Cuanto mayor sea este número, mayor rendimiento y menor carga de procesamiento.

Adicionalmente, este efecto tiene la característica de poder añadirle Lens Dirt, que sirve para añadirle una capa superficial, pudiendo darle una textura, por ejemplo, si queremos hacer un efecto de polvo. Este tiene su propio apartado con sus propias propiedades:

Propiedad	Función
Texture	La textura que se aplica al Bloom.
Intensity	La fuerza que tiene la textura.

Se puede ver un ejemplo de este filtro en la ilustración 6.

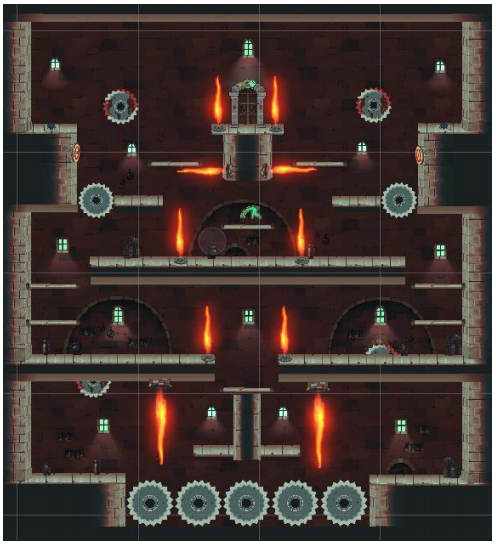


Ilustración 6: Post-procesado Bloom

Film Grain

Según la API de Unity, Film Grain es un efecto que simula la textura óptica de una película fotográfica, generalmente causada por pequeñas partículas que se encuentran en la escena. Se suele usar este efecto para dar

mayor credibilidad a la escena y darle un toque cinematográfico con el que dar un toque de nostalgia a los jugadores.

Las propiedades que tiene son: Type (Tipo de grano que usa el efecto), Texture (si se quiere usar una textura), Intensity (número que representa la intensidad del efecto) y Response (valor que representa el ruido que hace el efecto).

Propiedad	Función
Type	Tipo de grano a usar. Hay varios tipos, pero si quieres meterle una textura, tienes que ponerlo en Custom.
Texture	La textura por asignar al efecto.
Intensity	La intensidad de los granos del efecto.
Response	Representa el ruido que hace el efecto. A mayor valor, menor ruido en las áreas brillantes.

Se puede ver un ejemplo de este filtro en la ilustración 7.

Ilustración 7: Post-procesado Film Grain

Este efecto se basa en el estándar ASC CDL (American Society of Cinematographers Color Decision List), que es un formato que intercambia la información básica de los colores primarios. Este formato, a su vez define tres funciones matemáticas: pendiente, desplazamiento y potencia. Y cada función utiliza un número para los canales de rojo, azul y verde, siendo en total nueve números para definir un solo color, más un último número, el décimo, que recoge la saturación.

Las propiedades que incluye este efecto son: Lift que sirve para controlar los tonos oscuros, haciendo más potente el efecto en las sombras, gamma que se usa para controlar los tonos de rango medio con una función de potencia y, por último, Gain que se usa para aumentar la señal y hacer que los reflejos sean más brillantes.

Propiedad	Función
Lift - Tonos Oscuros	Controla los tonos oscuros, por ende, en las sombras será más potente el efecto. Se puede elegir el color de los tonos oscuros.
Gamma - Tonos Medios	Controla tonos de rango medio con una función de potencia. Se puede elegir el color de los tonos medios.
Gain - Iluminación	Aumenta la señal y hace que los reflejos sean más brillantes. Se puede elegir el color de las iluminaciones.

Se puede ver un ejemplo a continuación en la ilustración 8.

Ilustración 8: Post-procesado Lift Gamma Gain

Según la API de Unity, Shadows Midtones Highlights es un efecto que sirve para definir con precisión el rango tonal de las sombras, los tonos medios e iluminaciones.

Las propiedades que tiene este efecto son: Shadows (Color de los tonos oscuros. Sirve para controlar las sombras), Midtones (Color de los tonos medios. Sirve para controlar los tonos medios), y Highlights (Color de los reflejos. Sirve para controlar los reflejos). Por otro lado, tenemos las propiedades de Shadow Limits - Start y End (puntos de inicio y final de transición entre las sombras y los tonos medios) y Highlights - Start y End (puntos de inicio y final de transición entre los tonos medios y los reflejos).

Propiedad	Función
Shadows	Para controlar las sombras. Se puede elegir el color de los tonos oscuros.
Midtones	Para controlar los tonos medios. Se puede elegir el color de los tonos medios.
Highlights	Para controlar la iluminación. Se puede elegir el color de los reflejos.
Shadow Limits - Start	Punto de inicio de la transición entre las sombras y los medios tonos del render.
Shadow Limits - End	Punto final de la transición entre las sombras y los medios tonos del render.
Highlights Limits - Start	Punto de inicio de la transición entre los tonos medios y los aspectos más destacados del renderizado.
Highlights Limits - End	Punto final de la transición entre los tonos medios y los aspectos más destacados del renderizado.

Se puede ver un ejemplo de este efecto en la ilustración 9

Ilustración 9: Post-procesado Shadows Midtones Highlights

Split Toning

Según la API de Unity, es un efecto que pinta las sombras y reflejos de un determinado color, para lograr un aspecto un tanto distintivo.

Las propiedades que tiene son Shadows (color para teñir las sombras), Highlights (color para teñir los reflejos), y Balance (valor que representa un equilibrio entre los colores de sombras y reflejos. Los valores más bajos dan como resultado un tono de sombra más pronunciado en comparación con el tono del resaltado, y de lo contrario, tendríamos un efecto donde el tono de resaltado es más pronunciado que el tono de sombra).

Propiedad	Función
Shadows	Color que tinta las sombras.
HighLights	Color que tinta los reflejos.
Balance	Equilibrio entre los dos colores elegidos anteriormente.

Se puede ver un ejemplo de este filtro en la ilustración 10.

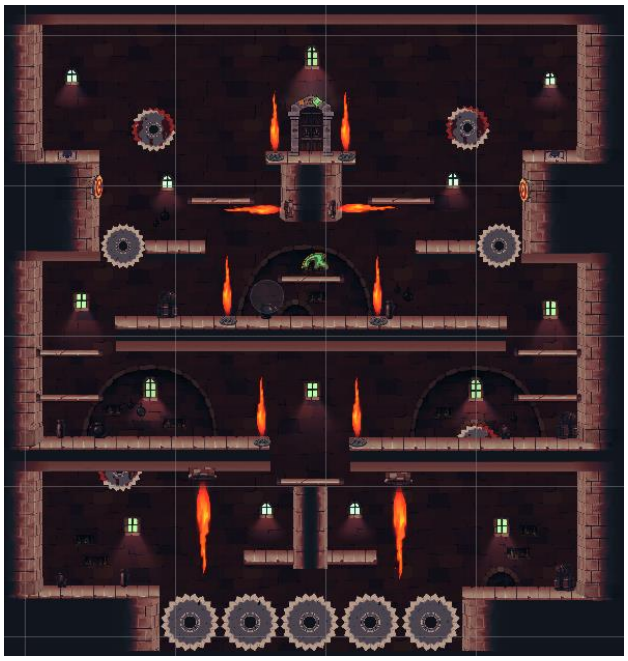


Ilustración 10: Post-procesado Split Toning

Tonemapping

Según Wikipedia, el Tonemapping es un proceso fotográfico que consiste en enriquecer el número de tonos de una imagen de modo que podamos ver al mismo tiempo sus tonos medios, sus tonos más oscuros y sus tonos luminosos, reduciéndose así el contraste global que se mitiga ante tanta variedad de tonos.

En Unity, es un efecto que realiza el proceso de Tonemapping reasignando los valores HDR de una imagen a un nuevo rango de valores.

La única propiedad que tiene es el modo, que puede ser ninguno, neutro (impacto mínimo en el tono y saturación del color), o ACES (apariencia más cinematográfica).

Propiedad	Función
Mode	Ninguno, Neutro o ACES

Se puede ver un ejemplo de este filtro en la ilustración 11

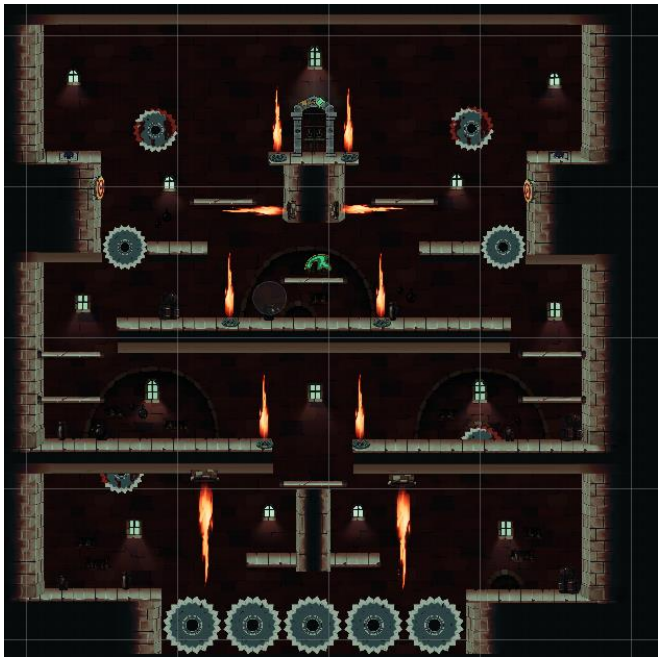


Ilustración 11: Post-procesado Tonemapping ACES

Vignette

Según la API de Unity, Vignette es un efecto de post-procesado que oscurece y/o desatura los bordes de una imagen en comparación con el centro. Normalmente, es utilizado para enfocar el centro de una imagen.

Tiene las siguientes propiedades: Color (color de la viñeta), Center (punto central de la viñeta), Intensity (valor que representa la fuerza del efecto), Smoothness (control deslizante para establecer la suavidad de los bordes de la viñeta), y Rounded (Booleano que sirve para darle una forma circular).

Propiedad	Función
Color	Color de la viñeta
Center	Punto central de la viñeta
Intensity	Fuerza del efecto
Smoothness	Suavidad de los bordes
Rounded	Forma circular de los bordes.

Se puede ver un ejemplo de este filtro en la ilustración 12.

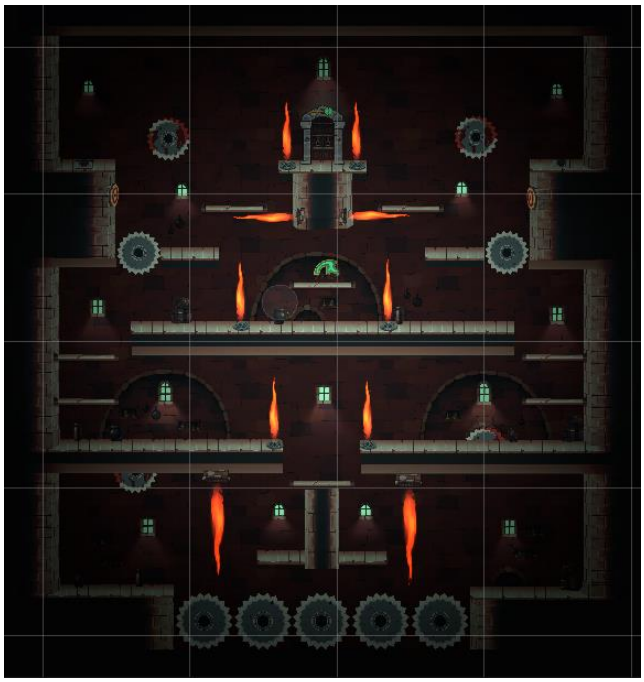


Ilustración 12: Post-procesado Vignette

White Balance

Según la API de Unity, es un efecto que elimina las tonalidades de color poco realistas, de esta manera, aquellas tonalidades o elementos que sean blancos en la vida real, se procesan como blancos en la imagen final. También sirve para darle una sensación más fría o cálida en el renderizado final.

Las propiedades que tiene son: Temperature (valor que representa el balance de blancos en una temperatura de color personalizada. Azul si se quiere dar una sensación más fría, y amarillo si se quiere dar una sensación más caliente), y Tint (Valor que representa el color con la que se tinta la imagen. Valor que va desde el color verde hasta el color morado).

Propiedad	Función
Temperature	Temperatura de color personalizada
Tint	Tintar la imagen de verde o de magenta.

Se puede ver un ejemplo de este filtro en la ilustración 13

Ilustración 13: Post-procesado White Balance

Tipos de luces

Point Light 2D

Tanto en la iluminación tradicional como en la iluminación 2D que implementa Universal Render Pipeline, funcionan de la misma manera. Un punto dentro de la escena que emite en un radio, una intensidad de luz en todas las direcciones. Esta intensidad va disminuyendo con la distancia.

Las propiedades que tiene este punto son:

- Inner Radius, donde la intensidad de la iluminación es el 100%.
- Outer Radius donde la intensidad empieza a disminuir con la distancia.
- Inner Angle es el ángulo que determina la superficie en la que actúa el Inner Radius.
- Outer Angle es el ángulo que determina la superficie en la que actúa el Outer Radius.
- Falloff Intensity es la intensidad con la que va decreciendo según la distancia (si está en 0 significa que la luz no va a perder intensidad).
- Cookie recoge un sprite como value, y crea una máscara con ese sprite para que la luz que emita salga con esa forma.
- Alpha Blend on Overlap: sirve para controlar la forma en que interactúan las luces en un mismo punto. Si está habilitado, la que tenga mayor número de light Order, se sobrepondrá a la luz que esté por debajo, y si

está deshabilitada esta opción, entonces, harán como una pequeña fusión (los valores de los píxeles de las luces que se cruzan se suman).
- Light Order por si tenemos más de una luz poderla organizar por layers. De esta manera, también se determina la posición de la luz en la cola de renderizado.
- Color: color de la propia luz.
- Intensity: valor de la intensidad con la que comienza a emitir la luz.
- Use Normal Map: por si queremos que la luz se adapte al normal map de los materiales de los sprites. De esta manera, le daríamos más realismo a la escena.
- Volume Opacity: controla la visibilidad del volumen de la luz.
- Shadow Intensity: la fuerza con la que se representa la sombra de los objetos.
- Target Sorting Layers: se puede poner la layer a la que afectará el punto de luz. Por ejemplo, si tenemos un sprite en la capa decorados, solo le afectará si la propia luz también se encuentra dentro de esa capa.

Con el uso de las propiedades de Inner Radius y Outer Radius se puede hacer una luz similar al tipo de luz Spot Light propio de Unity.

Se puede ver un ejemplo de este tipo de luz en la ilustración 14 y sus propiedades en la ilustración 15.

Ilustración 14: Point Light 2D URP

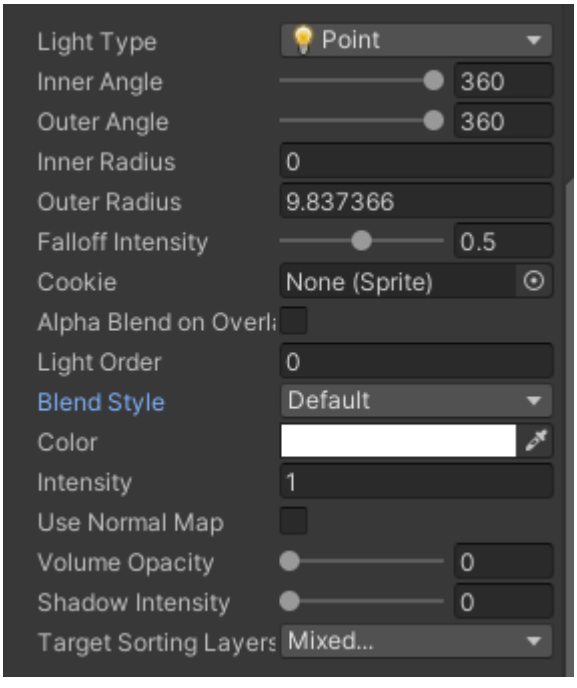


Ilustración 15: Propiedades de la ilustración 14.

Parametric Light 2D

Polígono de n-lados que desprende una determinada intensidad de luz dentro de esta, que va decreciendo con la distancia.

Las propiedades que tiene son:

- Radius: tamaño del polígono.
- Sides: los lados que tiene el polígono.
- Angle Offset: la rotación del polígono. Se mide en ángulos.
- Falloff: distancia a la que empieza a decrecer la luz que emite el polígono, comenzando desde el centro de la forma hasta sus bordes.
- Falloff Intensity: intensidad con la que decrece la iluminación. Si es 0 no decrece, y si es 1, se verá oscuro, como si no hubiese iluminación.
- Falloff Offset: establece el desplazamiento del área de decrecimiento de intensidad.
- Alpha Blend on Overlap: sirve para controlar la forma en que interactúan las luces en un mismo punto. Si está habilitado, la que tenga mayor número de light Order, se sobrepondrá a la luz que esté por debajo, y si está deshabilitada esta opción, entonces, harán como una pequeña fusión (los valores de los píxeles de las luces que se cruzan se suman).
- Light Order por si tenemos más de una luz poderla organizar por layers. De esta manera, también se determina la posición de la luz en la cola de renderizado.
- Color: color de la propia luz.
- Intensity: valor de la intensidad con la que comienza a emitir la luz.
- Use Normal Map: por si queremos que la luz se adapte al normal map de los materiales de los sprites. De esta manera, le daríamos más realismo a la escena.
- Volume Opacity: controla la visibilidad del volumen de la luz.
- Shadow Intensity: la fuerza con la que se representa la sombra de los objetos.
- Target Sorting Layers: se puede poner la layer a la que afectará el punto de luz. Por ejemplo, si tenemos un sprite en la capa decorados, solo le afectará si la propia luz también se encuentra dentro de esa capa.

Se puede ver un ejemplo de este tipo de luz en la ilustración 16 y sus propiedades en la ilustración 17.

Ilustración 16: Parametric Light 2D URP

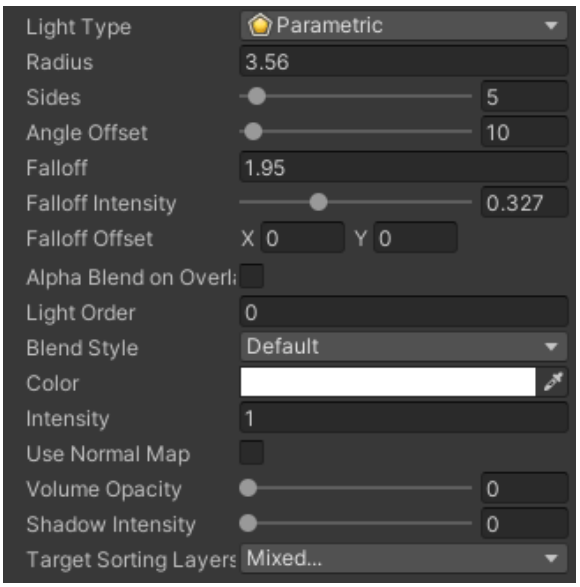


Ilustración 17: Propiedades de la ilustración 16.

Freeform Light 2D

Polígono con forma editable a través del botón edit shape que se encuentra en el inspector. Se puede agregar nuevos puntos de control haciendo clic en el contorno y eliminarlo presionando la tecla suprimir.

Las propiedades que tiene son:

- Falloff: distancia a la que empieza a decrecer la luz que emite el polígono, comenzando desde el centro de la forma hasta sus bordes.
- Falloff Intensity: intensidad con la que decrece la iluminación. Si es 0 no decrece, y si es 1, se verá oscuro, como si no hubiese iluminación.
- Falloff Offset: establece el desplazamiento del área de decrecimiento de intensidad.
- Alpha Blend on Overlap: sirve para controlar la forma en que interactúan las luces en un mismo punto. Si está habilitado, la que tenga mayor número de light Order, se sobrepondrá a la luz que esté por debajo, y si está deshabilitada esta opción, entonces, harán como una pequeña fusión (los valores de los píxeles de las luces que se cruzan se suman).
- Light Order por si tenemos más de una luz poderla organizar por layers. De esta manera, también se determina la posición de la luz en la cola de renderizado.
- Color: color de la propia luz.
- Intensity: valor de la intensidad con la que comienza a emitir la luz.
- Use Normal Map: por si queremos que la luz se adapte al normal map de los materiales de los sprites. De esta manera, le daríamos más realismo a la escena.
- Volume Opacity: controla la visibilidad del volumen de la luz.

- Shadow Intensity: la fuerza con la que se representa la sombra de los objetos.
- Target Sorting Layers: se puede poner la layer a la que afectará la freeform light. Por ejemplo, si tenemos un sprite en la capa decorados, solo le afectará si la propia luz también se encuentra dentro de esa capa.
- Edit Shape: opción que sirve para retocar la forma del polígono.

Se puede ver un ejemplo de este tipo de luz en la ilustración 18 y sus propiedades en la ilustración 19.

Ilustración 18: Freeform Light 2D URP

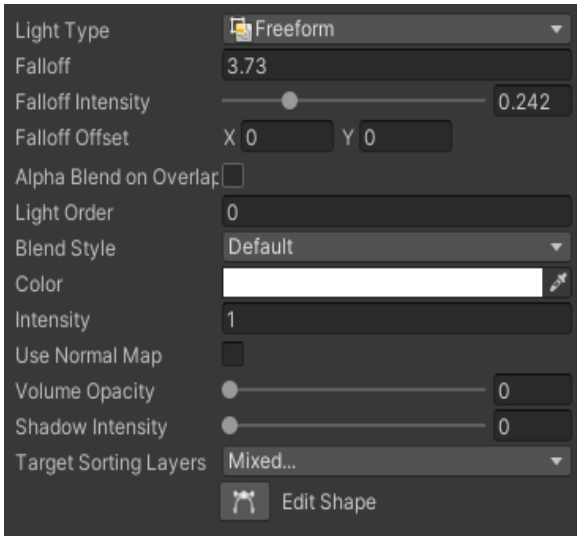


Ilustración 19: Propiedades de la ilustración 18

Sprite Light 2D

Este tipo de luz utiliza una sprite como fuente de luz. Suelen ser utilizadas para alguna animación de transición.

Las propiedades que tienen son:

- Sprite: sprite que coge como fuente de luz.
- Alpha Blend on Overlap: sirve para controlar la forma en que interactúan las luces en un mismo punto. Si está habilitado, la que tenga mayor número de light Order, se sobrepondrá a la luz que esté por debajo, y si está deshabilitada esta opción, entonces, harán como una pequeña fusión (los valores de los píxeles de las luces que se cruzan se suman).
- Light Order por si tenemos más de una luz poderla organizar por layers. De esta manera, también se determina la posición de la luz en la cola de renderizado.
- Color: color de la propia luz.
- Intensity: valor de la intensidad con la que comienza a emitir la luz.

- Use Normal Map: por si queremos que la luz se adapte al normal map de los materiales de los sprites. De esta manera, le daríamos más realismo a la escena.
- Volume Opacity: controla la visibilidad del volumen de la luz.
- Shadow Intensity: la fuerza con la que se representa la sombra de los objetos.
- Target Sorting Layers: se puede poner la layer a la que afectará el sprite Light. Por ejemplo, si tenemos un sprite en la capa decorados, solo le afectará si la propia luz también se encuentra dentro de esa capa.
- Edit Shape: opción que sirve para retocar la forma del polígono.

Se puede ver un ejemplo de este tipo de luz en la ilustración 20, y sus propiedades en la ilustración 21.

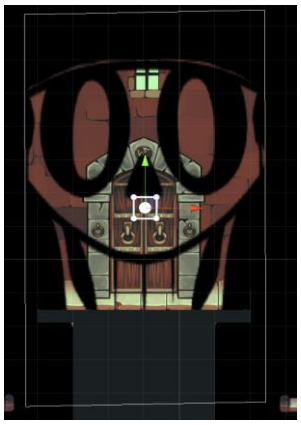


Ilustración 20: Sprite Light 2D URP

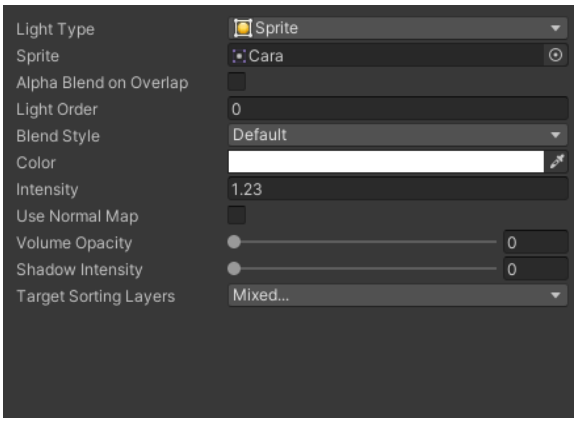


Ilustración 21: Propiedades de la ilustración 20.

Global Light 2D

Tipo de luz que sirve para iluminar todos los objetos de esa capa al mismo tiempo. Cuidado, no puede haber más de dos Global Light por capa seleccionada o por Blend Style.

Las propiedades de Global Light son:

- Light Order: por si se tiene más de una luz poder organizar por layers. De esta manera, se determina la posición de la luz en la cola de renderizado.
- Blend Style: la capa en la que se encuentra. Los tipos de Blend Style se pueden cambiar en las opciones del 2D Renderer Data.
- Color: color de la propia luz.
- Intensity: valor de la intensidad con la que comienza a emitir la luz.
- Target Sorting Layers: Se puede poner la layer a la que afectará la global light. Por ejemplo, si tenemos un sprite en la capa decorados, solo le afectará si la propia luz también se encuentra dentro de esa capa.

Se puede ver un ejemplo de este tipo de luz en la ilustración 23 y sus propiedades en la ilustración 24, además de su diferencia con la ilustración 22.

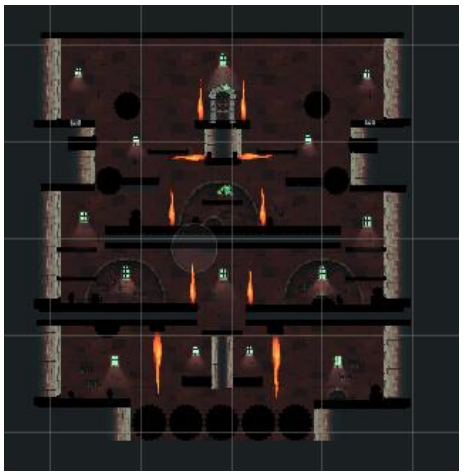


Ilustración 22: Escenario sin Global Light

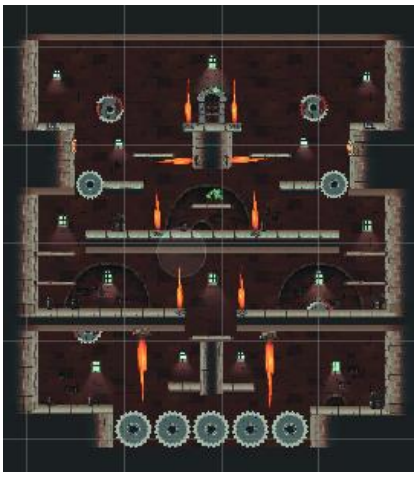


Ilustración 23: Escenario con Global Light

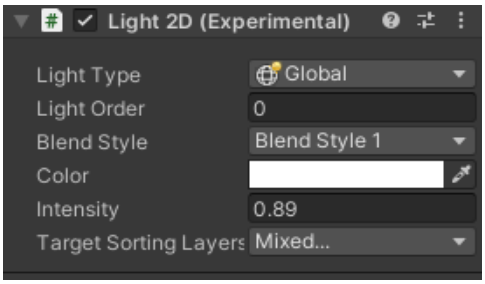


Ilustración 24: Propiedades de la ilustración 23

Secondary Textures

Una opción que permite el Universal Render Pipeline es usar normal maps, y máscaras para conseguir efectos de luz más avanzados, como, por ejemplo, que la luz se amolde al objeto. Normalmente, suele dar el efecto de delineado.

Para conseguir usar las segundas texturas, se tiene que abrir el sprite editor del sprite en cuestión que se quiera iluminar. Si se despliega el menú, y se ha actualizado correctamente el proyecto al Universal Render Pipeline, tendría que salir la opción de Secondary Textures. En ese apartado, se da la opción de incluir un normal map, o una máscara. y un nombre para este objeto. Este nombre es personalizado, pero se sugiere que el nombre de la textura permita ser utilizada en sus shaders, como por ejemplo _MainTexture o _MaskTex.

Se puede ver un ejemplo de la interfaz del Sprite Editor en la ilustración 25.

Ilustración 25: Interfaz del Sprite Editor para las Secondary Texture

Para qué funcione la luz sobre este tipo de objetos, es necesario activar la propiedad de Use Normal Map, para que se adapte al normal map que se le ha pasado anteriormente en el sprite editor.

Se puede ver la diferencia entre usar Secondary Texture y no usarlas en las ilustraciones 26 y 27.

Ilustración 26: Captura sin Secondary Texture

Ilustración 27: Captura con Secondary Texture

Ejemplo

El tipo de luz Point Light 2D se ha usado para representar ese rayo de luz que entra por las ventanas. Para ello, se ha tenido que transformar el Point Light en Spot Light, como se puede ver en la ilustración 28 y sus propiedades en la ilustración 29.

Ilustración 28: Rayo de luz. Spot Light

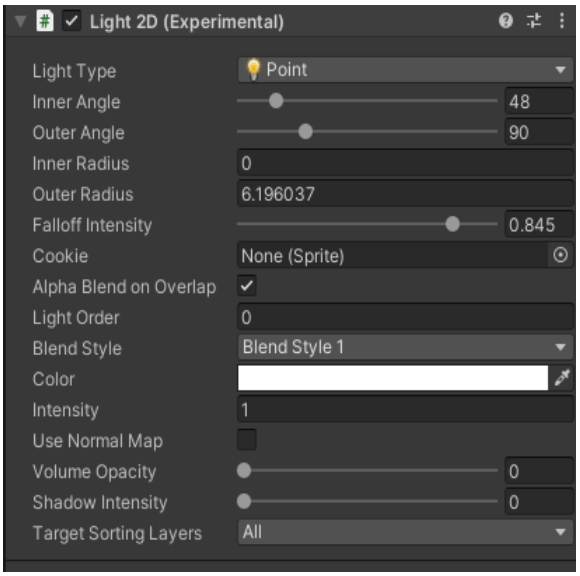


Ilustración 29: Propiedades de la ilustración 28.

Para realizar el rayo de luz, se ha bajado tanto el inner angle como el outer angle, para así transformar Point Light en Spotlight (forma de cono), además de tocar el Inner radius y el outer radius para dar la sensación de que la luz, se ve reducida en los bordes.

Además, aparte de tener un Point Light, tiene una freeform para que no se vea tan fuerte el camino del rayo en el interior de la ventana, como se puede apreciar en la ilustración 30 y sus propiedades en la ilustración 31:

Ilustración 30: Ventana con iluminación 2D

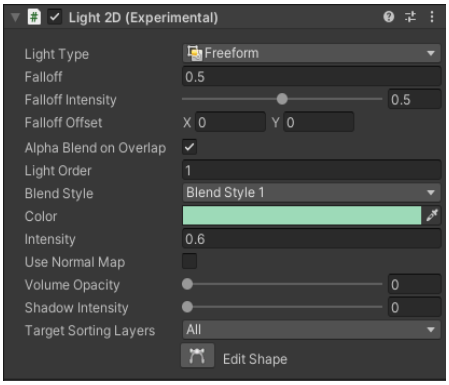


Ilustración 31: Propiedades de la ilustración 30

Algo muy importante es que se pone la variable de Alpha Blend on Overlap, para que se anteponga la freeform light sobre el Point Light. Porque si no se activa esta función, se quedarían ambas luces, quedando un contraste raro, como se puede apreciar en la ilustración 32:

Ilustración 32: Ejemplo del uso de Blend Overlap

Y se le agrega una tercera luz, una Parametric Light, porque el borde no está iluminado, y aunque el rayo de luz no choque directamente con este, acabará chocando por algún rebote. Por lo que se le agrega una luz muy tenue alrededor del marco.

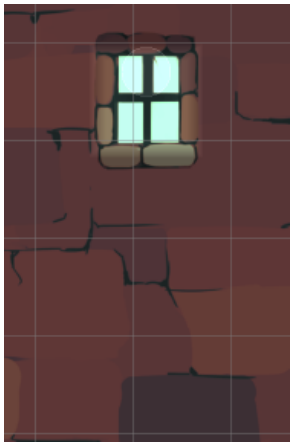


Ilustración 33: Ventana iluminada Terminada

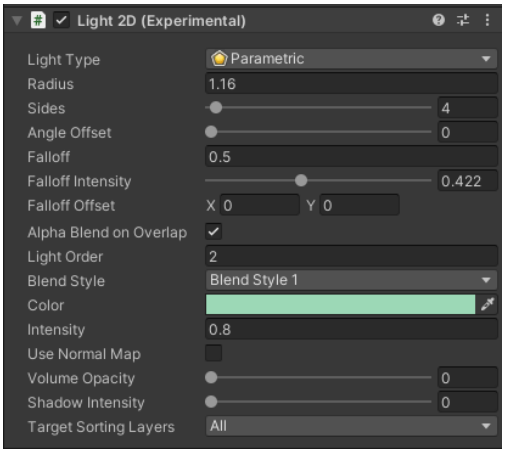


Ilustración 34: Propiedades de la ilustración 33

Iluminación propia de Built-in Render Pipeline

El render pipeline que viene por defecto en los proyectos de Unity 2D es el Built-in Render Pipeline, render que se creó hace diez años. Built-in Render tiene varios caminos de renderizado, esto significa, que tiene varias opciones para realizar una serie de operaciones relativas a la iluminación y sombreado. Cada camino de renderizado tiene sus propias características y capacidades, que se pueden modificar en la pestaña de Graphics de tu proyecto de Unity.

Si no se elige ningún camino de renderizado o la GPU del dispositivo no es compatible con la ruta de representación seleccionada, se usará el que viene por defecto: Forward Rendering.

Forward Rendering es una ruta de renderizado de uso general, donde las luces en tiempo real son muy caras de renderizar, y para solucionar este problema deja elegir el número de luces a representar por píxel, haciendo que el resto de las luces en la escena tengan menor fidelidad.

Shader

El shader por excelencia del Built-in Render es Standard Shader, que representa lo mismo que Lit Shader en Universal Render Pipeline. Se utiliza para renderizar los objetos del mundo real, como piedras, madera, etc.... y soporta un gran número de tipos de shaders y combinaciones.

Standard Shader incorpora un avanzado modelo de iluminación llamado PBR (Physically Based Shading) que simula la interacción entre materiales y la luz de una forma realista.

Con Standard Shader se tiene un gran número de shaders (Diffuse, Specular, Reflective) que son combinadas en un único shader destinado a ser utilizado en todos los materiales. Esto trae un gran beneficio, y es que se utilizan los mismos cálculos de iluminación en todas las áreas de la escena, ofreciendo una distribución coherente y creíble de la luz y la sombra en todos los modelos que utilizan el sombreador.

Post-Procesado

Para usar el post-procesado con el render Built-in se necesita descargar un package llamado Postprocessing v_2. Los efectos que contiene son:

Auto Exposure

Según la API de Unity, el efecto auto exposure sirve para ajustar la exposición de una imagen[2] de acuerdo con el rango de niveles de brillo que contiene la imagen.

Las propiedades que tiene son: Filtering (rango para establecer los porcentajes superiores e inferiores para una luminancia media estable), minimum EV (Luminancia[3] mínima a considerar), maximum EV (Luminancia máxima a considerar), Exposure Compensation (Valor de gris medio), Type (el tipo de auto exposure, puede ser: progresivo (anima el efecto) o fijo (no anima el efecto), Speed up (Velocidad de adaptación de un entorno oscuro a uno claro) y Speed down (Velocidad de adaptación de un entorno claro a uno oscuro).

Propiedad	Función
Filtering	Rango para establecer los porcentajes superiores e inferiores para una luminancia media estable. Todos los valores que queden fuera de este rango se descartan y no contribuirán a la luminancia promedio.
Minimum (EV)	Luminancia mínima para auto-exposure en EV.
Maximum (EV)	Luminancia máxima para auto-exposure en EV.
Exposure Compensation	Valor de gris medio.
Type	Progresivo: anima el auto-exposure. Fija: no anima el auto-exposure.
Speed Up	Velocidad de adaptación de un entorno oscuro a uno claro.
Speed Down	Velocidad de adaptación de un entorno claro a uno oscuro.

Se puede ver un ejemplo en la ilustración 35:

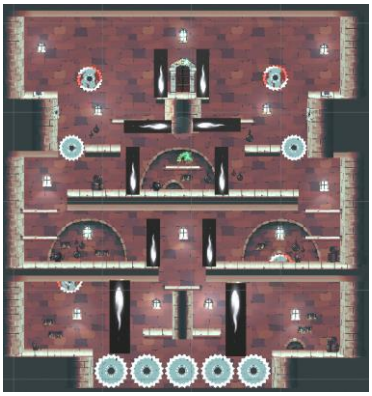


Ilustración 35: Auto-Exposure Built-in Render

Bloom

Según la API de Unity, se considera Bloom al efecto de post-procesado que realza los bordes de las áreas más brillantes de la imagen.

Las propiedades son: Intensity (valor que representa la intensidad del efecto), Threshold (nivel de brillo), Soft Knee (valor que representa el umbral para las transiciones, siendo 0 un umbral duro, y 1 un umbral suave), Clamp (valor para controlar la cantidad de Bloom), Diffusion (valor que representa el alcance de los efectos de velo, cuánto más bajo sea este valor, más rápido irá el efecto), Anamorphic Ratio (valor que emula el efecto de una lente anamórfica[4], cuánto más alto este valor, más lento irá el efecto), Color (color del efecto) y por último, Fast Mode (Booleano para mejorar el rendimiento a cambio de reducir la calidad del efecto).

Propiedad	Función
Intensity	Intensidad del efecto
Threshold	El nivel de brillo.
Soft Knee	Umbral gradual para las transiciones entre por debajo/encima del umbral.
Clamp	Valor de los pixeles de sujeción para controlar la cantidad de Bloom.
Diffusion	Alcance de los efectos de velo de forma independiente a la resolución de la pantalla. Cuanto más bajo sea, más

	rápido será el efecto.
Anamorphic Ratio	Emula el efecto de una lente anamórfica. Cuanto más alto sea el valor, más lento irá el efecto.
Color	Color del efecto.
Fast Mode	Para mejorar el rendimiento al reducir la calidad del efecto.

Se puede ver un ejemplo en la ilustración 36:

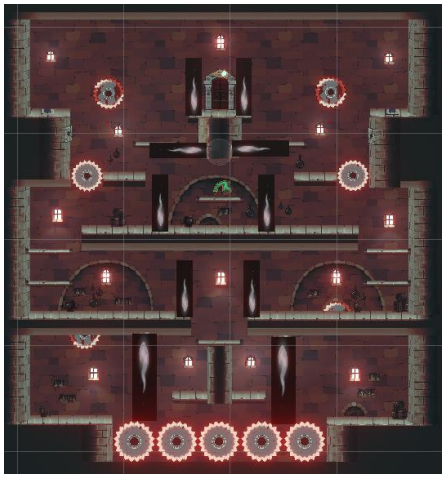


Ilustración 36: Bloom Built-in Render

Color Grading

Según la API de Unity, es un efecto que altera o corrige el color y la luminancia de la imagen final.

Las propiedades principales que tiene son: Mode (el modo de color grading que son: Low Definition Range (LDR), High Definition Range (HDR) y External), Lookup Texture (Textura del efecto. Solo funciona si el modo elegido es LDR o External) y por último Contribution (cantidad de textura que contribuirá al efecto. Solo funciona si el modo elegido es LDR).

Propiedad	Función
Mode	- Low Definition Range: ideal para plataformas de gama baja. - High Definition Range: ideal para plataformas que admiten renderizado HDR. - External

Lookup Texture	Solo admite textura si se encuentra en el modo de Low Definition Range of External.
Contribution	Solo está disponible si el modo seleccionado es Low Definition Range.

A parte de estas opciones, este efecto, se divide por funciones, que en el Universal Render Pipeline son considerados efectos independientes, y son:

Tonemapping

Según la API de Unity, tonemapping reasigna los valores HDR de una imagen en un rango adecuado para mostrarse por pantalla. Solo funciona con el modo de Color Grading: High Definition Range (HDR).

Las propiedades del tonemapping son: Mode (el modo de HDR, que puede ser: None (no se aplica tonemapping), Neutral (impacto mínimo en el tono y saturación del color), ACES (se utiliza más para una apariencia cinematográfica,), Custom (tonemapping totalmente paramétrico, siendo ese el único con su propia configuración dentro de todos los modos)), Toe Strength (valor para la transición entre la punta y la sección media de la curva, siendo 0 que no hay curva, y 1 que sea una transición difícil), Toe Length (valor de cuánto del rango dinámico hay en la punta), Shoulder Strength (valor para la transición media y el hombro de la curva, siendo 0 que no hay hombro, y 1 siendo una transición difícil), Shoulder Length (valor de cuántos pasos F (EV) agregar al rango dinámico de la curva), Shoulder Angle (Valor de cuánto rebasamiento agregar al hombro) y Gamma (valor para aplicar una función gamma a la curva).

Propiedad	Función
Mode	- None: no se aplica ningún tonemapping - Neutral: reasignación de rango que provoca un impacto mínimo en el tono y saturación del color. - ACES: apariencia cinematográfica. Todas las operaciones se realizan en los espacios de color ACES para obtener resultados y precisión óptimos. - Custom: único tonemapping con su

	propia configuración.
Toe Strength	Valor para la transición entre la punta y la sección media de la curva. Siendo 0 que no hay curva, y 1 que sea una transición difícil.
Toe Length	Valor de cuánto del rango dinámico hay en la punta.
Shoulder Strength	Valor para la transición media y el hombro de la curva. Siendo 0 que no hay hombro, y 1 siendo una transición difícil.
Shoulder Length	Valor de cuántos pasos F (EV) agregar al rango dinámico de la curva.
Shoulder Angle	Valor de cuánto rebasamiento agregar al hombro.
Gamma	Valor para aplicar una función gamma a la curva

Tone

Las propiedades de Tone son: Post-exposure (solo disponible con el modo HDR, y establece el valor de la exposición general de la escena en unidades EV), Color Filter (Color con el que se renderizará), Hue Shift (Tono de todos los colores), Saturation (Valor que representa la intensidad de todos los colores), Brigthness (solo disponible durante el modo LDR, y ajusta el brillo de la imagen) y Contrast (Valor que sirve para ajustar el rango general de los valores tonales).

Propiedad	Función
Post-Exposure	Valor de la exposición general de la escena en unidad EV. Solo disponible con el modo HDR.

Color Filter	Color del tinte del render.
Hue Shift	Tono de todos los colores.
Saturation	Valor que representa la intensidad de todos los colores.
Brightness	Ajusta el brillo de la imagen. Solo disponible con el modo LDR.
Contrast	Valor que sirve para ajustar el rango general de los valores tonales.

Trackballs

Según la API de Unity, esta función se utiliza para realizar una graduación de color de tres vías. Se utilizan diferentes rangos dentro de la imagen. Con el Universal Render Pipeline tiene otro nombre: Lift Gamma y Gain.

Las propiedades de esta función son: Lift (Ajusta los tonos oscuros), Gamma (Ajusta los tonos medios) y Gain (Ajusta los tonos de los reflejos).

Propiedad	Función
Lift	Ajusta los tonos oscuros.
Gamma	Ajusta los tonos medios.
Gain	Ajusta los tonos de los reflejos.

Grading Curves

Según la API de Unity, es una manera avanzada de ajustar los rangos de saturación, hue o luminosidad de la imagen / escena. En nuestro caso, solo se va a comentar la función de luminosidad que es la que tiene relación con la iluminación.

Lum vs Sat que sirve para ajustar la saturación en ciertas áreas de luminosidad. Esta curva ajusta la saturación (en el eje Y) de acuerdo con la luminosidad de entrada (en el eje X), y utiliza este ajuste para desaturar áreas oscuras para proporcionar un contraste visual interesante.

Se puede ver un ejemplo de Color Grading que engloba estas últimas cuatro propiedades descritas en la ilustración 37:

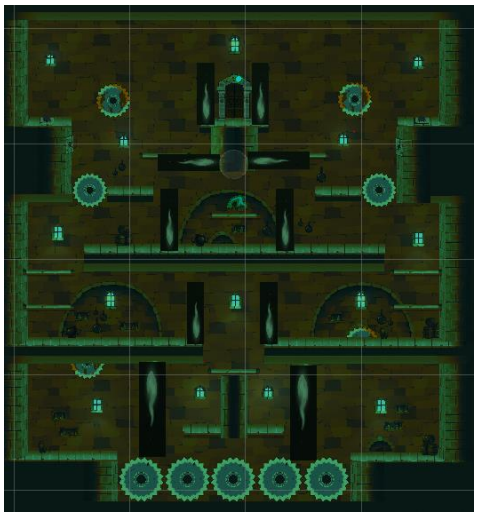


Ilustración 37: Color Grading Built-in

Screen Space Reflections

Según la API de Unity, es un efecto ideal para limitar la cantidad de luz especular que se filtra.

Las propiedades son: Preset (Sirve para ajustar la calidad), Maximum Iteration Count (número máximo de pasos en el pase de raymarching), Thickness (valor del grosor del rayo), Resolution (Tamaño del buffer interno), Maximum March Distance (valor que representa la distancia máxima a recorrer en la escena después de la cual dejará de dibujar reflejos), Distance Fade (valor de la distancia para atenuar los reflejos cerca del plano cercano), y Vignette (valor para atenuar los reflejos cerca de los bordes de la pantalla).

Propiedad	Función
Preset	Sirve para ajustar la calidad.
Maximum Iteration Count	Número máximo de pasos en el pase de raymarching. Valores más altos significan más reflejos.

Thickness	Valor del grosor del rayo. Cuanto más bajo sea, más recursos, pero detectan detalles más pequeños.
Resolution	Tamaño del buffer interno. Tiene dos opciones: - Downsample: maximiza el rendimiento. - Supersample: más lento, pero produce resultados de mayor calidad.
Maximum March Distance	Valor que representa la distancia máxima a recorrer en la escena después de la cual dejará de dibujar reflejos
Distance Fade	Valor de la distancia para atenuar los reflejos cerca del plano cercano.
Vignette	Valor para atenuar los reflejos cerca de los bordes de la pantalla

Se puede ver un ejemplo en la ilustración 38:

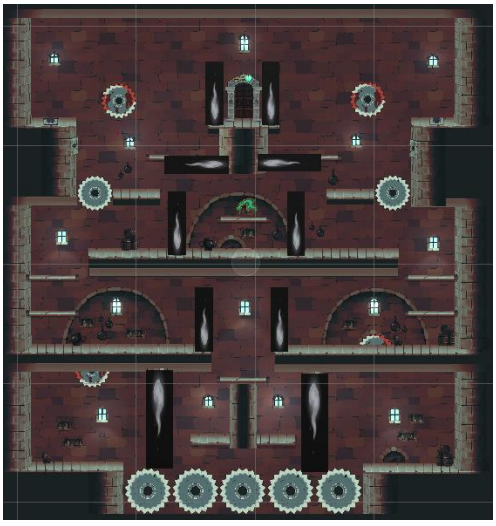


Ilustración 38:Screen Space Reflection Built-in

Vignette

Según la API de Unity, Vignette es un efecto que oscurece los bordes de una imagen, dejando el centro de la imagen más brillante. Normalmente, es utilizado para enfocar el centro de una imagen.

Este efecto tiene dos modos con sus propias características y propiedades. El primer modo es el clásico que tiene controles paramétricos para la posición, forma e intensidad de la viñeta. Las propiedades que tiene son: Color (Color del efecto), Center (Punto central del efecto), Intensity (Fuerza del efecto), Smoothness (Suavidad de los bordes del efecto), Roundness (Valor para redondear el efecto), y Rounded (Booleano que sirve para que la forma del efecto sea perfectamente redonda).

Propiedad	Función
Color	Color del efecto.
Center	Punto central del efecto. El punto por defecto es (0'5, 0'5)
Intensity	Fuerza del efecto.
Smoothness	Suavidad de los bordes del efecto.
Roundness	Valor para redondear el efecto. Cuanto más bajo sea este valor, más cuadrado será el efecto, y viceversa, cuanto más alto, más redondo.
Rounded	(Booleano que sirve para que la forma del efecto sea perfectamente redonda.

Se puede ver un ejemplo en la ilustración 39:

Ilustración 39: Vignette Built-in Render

Tipos de luces

Point Light

Un tipo de luz que se define como un punto en el espacio 3D que emite luz en todas las direcciones, y que va decreciendo con la distancia. La intensidad de la luz es inversamente proporcional al cuadrado de la distancia desde la fuente. Simulando el comportamiento de la luz en la vida real. El único inconveniente es que, si queremos iluminar otro punto de luz, este tiene un vector L distinto, por lo que habría que ir calculando el módulo de cada vector, lo que se hace pesado.

Las propiedades de Point Light son:

- Range: Distancia a la que la luz se va a emitir. Se podría definir como el radio del área donde se emite la luz.
- Color: Color que emite la luz.
- Mode: Determina si la luz va a ser baqueada, es decir, las sombras se imprimen una sola vez.
- Intensity: el valor de la intensidad con la que se emite la luz.
- Indirect Multiplier: intensidad de las luces indirectas. Si es menor que 1, la luz que rebota se atenúa con cada rebote, y si es mayor que 1 hace que la luz sea más brillante con cada rebote.
- Shadow Type: determina el tipo de sombra: Hard Shadow (produce sombras con un borde afilado), Soft Shadows (más realistas que el primer modo, pero implica menos procesamiento) y No Shadows (no emite sombras).
- Render Mode: establece la prioridad de reproducción de la luz seleccionada. Según la API de Unity, esto puede afectar a la fidelidad y rendimiento de la iluminación. Los modos son: auto (este modo se determina en tiempo de ejecución, según el brillo de las luces cercanas y la configuración de calidad actual), importante (la luz se renderiza con una calidad por píxel, por lo que será más lento y costoso, pero más realista).
- Culling Mask: layer a la que afecta esta luz, de esta manera, afectará únicamente a los objetos de esas layers.

Se puede ver un ejemplo en la ilustración 40 y sus propiedades en la ilustración 41.

Ilustración 40: Point Light Built-in

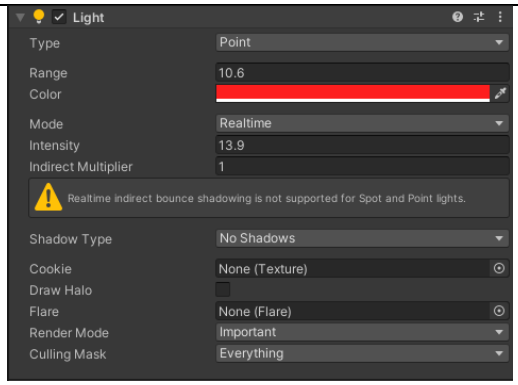


Ilustración 41: Propiedades de la ilustración 40

Spot Lights

Son un tipo de luces que emanan de un punto en cualquier dirección contenida dentro de un cono. Estas fuentes de luces suelen venir definidas por un punto, un vector y un radio de superficie de cono. Al igual que el tipo anterior, Point Light, la luz disminuye en los bordes del cono del foco, a este efecto de desvanecimiento se le conoce como “penumbra”.

Las propiedades de Spot Light son:

- Range: Distancia a la que la luz va a emitir.
- Inner/Outer Spot Angle: Inner angle es el ángulo que determina la superficie en la que actúa el radio interior. Outer Angle es el ángulo que determina la superficie en la que actúa el radio externo.
- Color: Color de la luz.
- Mode: Determina si la luz va a ser baqueada, es decir, las sombras se imprimen una sola vez.
- Intensity: el valor de la intensidad con la que se emite la luz.
- Indirect Multiplier: intensidad de las luces indirectas. Si es menor que 1, la luz que rebota se atenúa con cada rebote, y si es mayor que 1 hace que la luz sea más brillante con cada rebote.
- Shadow Type: determina el tipo de sombra: Hard Shadow (produce sombras con un borde afilado), Soft Shadows (más realistas que el primer modo, pero implica menos procesamiento) y No Shadows (no emite sombras).
- Render Mode: establece la prioridad de reproducción de la luz seleccionada. Según la API de Unity, esto puede afectar a la fidelidad y rendimiento de la iluminación. Los modos son: auto (este modo se determina en tiempo de ejecución, según el brillo de las luces cercanas y la configuración de calidad actual), importante (la luz se renderiza con

una calidad por píxel, por lo que será más lento y costoso, pero más realista).
- Culling Mask: layer a la que afecta esta luz, de esta manera, afectará únicamente a los objetos de esas layers.

Se puede ver un ejemplo de Spot Light en la ilustración 42 y sus propiedades en la ilustración 43.

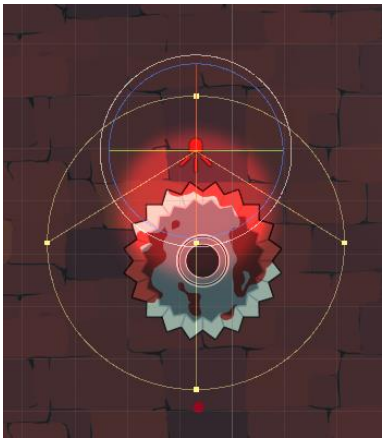


Ilustración 42: Spot Light Built-in

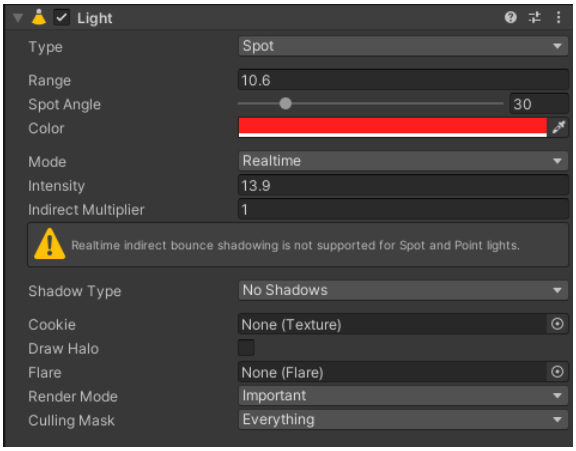


Ilustración 43: Propiedades de la ilustración 42

Directional Light

Según la API de Unity, son un tipo de luz que puede considerarse como una fuente de luz distantes que existen infinitamente lejos. Una luz direccional no tiene ninguna posición de fuente identificable, por lo que el objeto de luz se puede colocar en cualquier lugar de la escena. Todos los objetos de la escena se iluminan como si la luz procediera siempre de la misma luz.

Una ventaja de este tipo de luz es que todos los rayos de luz tienen el mismo vector L porque son paralelos, lo que conlleva a que todos tienen el mismo módulo, y no tenga que calcularse para cada rayo. Gracias a esto se tiene un mejor rendimiento.

Las propiedades son:

- Color: color de la luz.
- Mode: Determina si la luz va a ser baqueada, es decir, las sombras se imprimen una sola vez.
- Intensity: el valor de la intensidad con la que se emite la luz.
- Indirect Multiplier: intensidad de las luces indirectas. Si es menor que 1, la luz que rebota se atenúa con cada rebote, y si es mayor que 1 hace que la luz sea más brillante con cada rebote.
- Shadow Type: determina el tipo de sombra: Hard Shadow (produce sombras con un borde afilado), Soft Shadows (más realistas que el

primer modo, pero implica menos procesamiento) y No Shadows (no emite sombras).
- Render Mode: establece la prioridad de reproducción de la luz seleccionada. Según la API de Unity, esto puede afectar a la fidelidad y rendimiento de la iluminación. Los modos son: auto (este modo se determina en tiempo de ejecución, según el brillo de las luces cercanas y la configuración de calidad actual), importante (la luz se renderiza con una calidad por píxel, por lo que será más lento y costoso, pero más realista).
- Culling Mask: layer a la que afecta esta luz, de esta manera, afectará únicamente a los objetos de esas layers.

Se puede ver un ejemplo con Directional Light en la ilustración 44 y sin Directional Light en la ilustración 45.

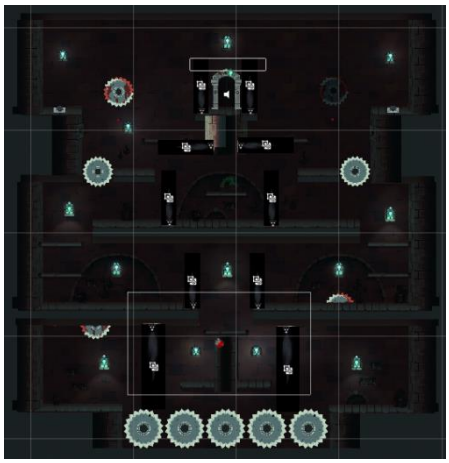


Ilustración 45: Sin Directional Light

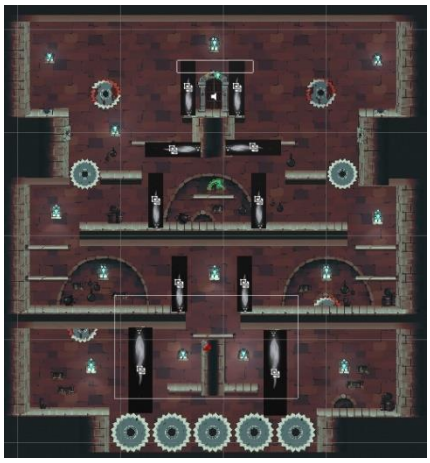


Ilustración 44: Con Directional Light

Area Light

Estos tipos de luz son superficies emisoras de luz. Esto es importante cuando se quiere modelar las transiciones de la parte iluminada a la parte de sombra, para que sea un borde difuminado. Solo se pueden modelar cuando usamos Ray Tracing (en lugar de un único rayo de sombra, trazar varios) o Path Tracing.

Según la API de Unity, una Area Light ilumina un objeto desde varias direcciones diferentes a la vez, el sombreado tiende a ser más suave y sutil que los otros tipos de luz.

Las propiedades son:

- Shape: variable que sirve para indicar si la forma es rectangular, o en forma de disco.
- Range: Distancia a la que la luz va a emitir.
- Width: Ancho de la forma de la Area Light.

- Height: Alto de la forma de la Area Light.
- Color: Color de la luz.
- Intensity: el valor de la intensidad con la que se emite la luz.
- Indirect Multiplier: intensidad de las luces indirectas. Si es menor que 1, la luz que rebota se atenúa con cada rebote, y si es mayor que 1 hace que la luz sea más brillante con cada rebote.
- Cast Shadows: booleano que indica si se castean Soft Shadows o no se castea ninguna.
- Render Mode: establece la prioridad de reproducción de la luz seleccionada. Según la API de Unity, esto puede afectar a la fidelidad y rendimiento de la iluminación. Los modos son: auto (este modo se determina en tiempo de ejecución, según el brillo de las luces cercanas y la configuración de calidad actual), importante (la luz se renderiza con una calidad por píxel, por lo que será más lento y costoso, pero más realista).
- Culling Mask: layer a la que afecta esta luz, de esta manera, afectará únicamente a los objetos de esas layers.

Un problema que tiene la Area Light es que no funciona sobre Sprites, por ello, normalmente se utiliza el Point Light como sustituto del Area Light.

Ejemplos

Dentro del proyecto, como se ha visto anteriormente en el apartado de la iluminación propia de Universal Render Pipeline, un ejemplo son las ventanas del nivel jugable de la cocina.

Primero de todo, comentar, que para que la luz funcionase sobre los elementos de la escena, se crea un material de tipo Sprites/Diffuse. Por defecto, suelen venir dos propiedades de shader para objetos 2D o Sprites, y son: default (no interactúa con las luces de la escena), y diffuse (interactúa con las luces de la escena).

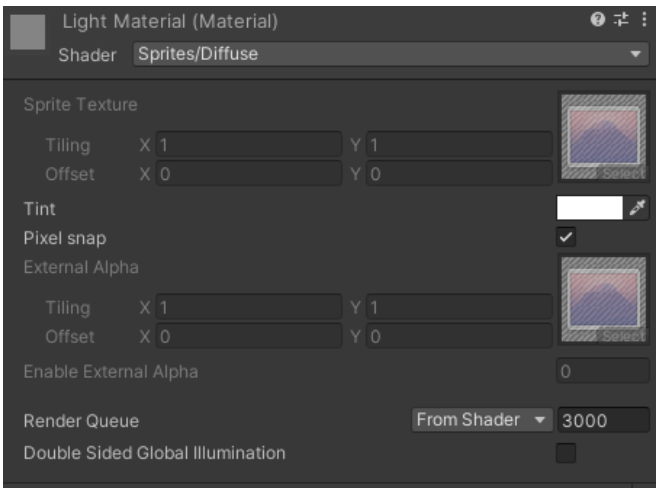


Ilustración 46: Propiedades del material

Ilustración 47: Material Built-in

Hay que tener en cuenta, que a la hora de introducir luces 3D con Sprites, hay que fijarse en las coordenadas, porque muchas veces, existe una luz, pero no interactúa con el sprite porque puede estar detrás del sprite.

La primera luz que se pone es Spot Light para simular el rayo de luz que entra por la ventana:

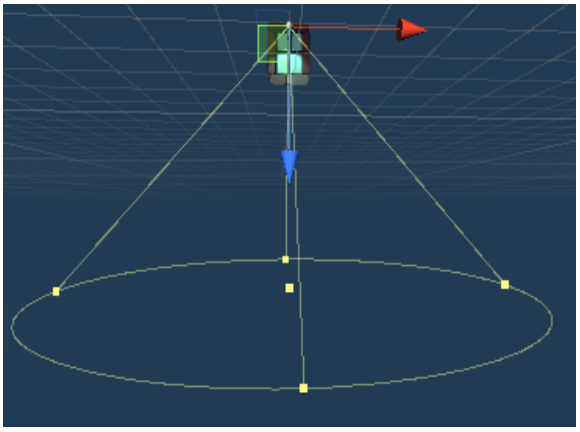


Ilustración 48: Point Light Built-in Render

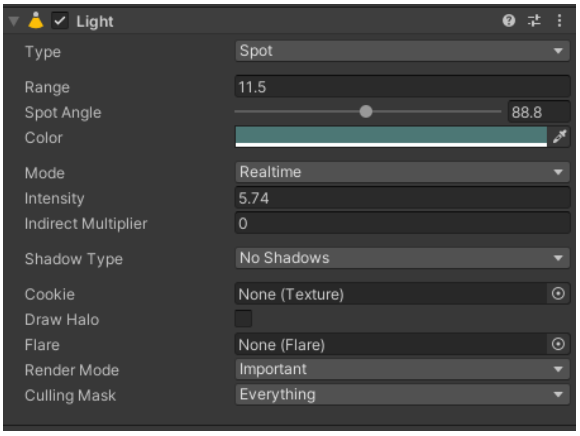


Ilustración 49: Propiedades de la ilustración 48.

La segunda luz es un Point Light, para simular la entrada de luz en su totalidad, y para camuflar ese rayo de luz tan intenso. Cabe destacar que para esta iluminación se le ha dado unas capas determinadas sobre las que actuar, por ejemplo, solo se quiere actuar sobre el objeto, no sobre el fondo o sobre otros objetos, ya que el objetivo con esta luz es que no se vea tan lineada el cono de luz que entra por la ventana. Como se puede ver en la ilustración 50 y sus propiedades en la ilustración 51.

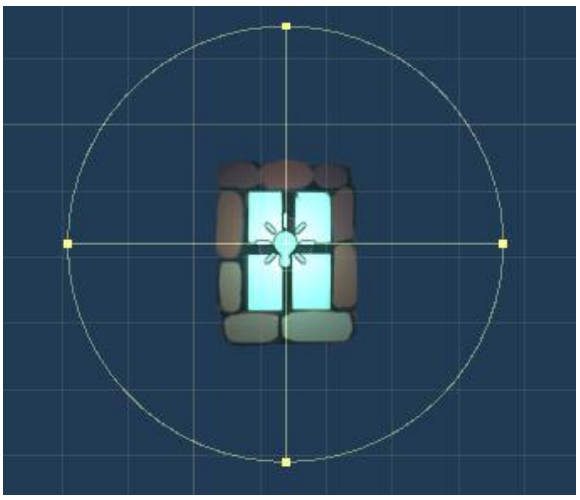


Ilustración 50: Ejemplo Built-in

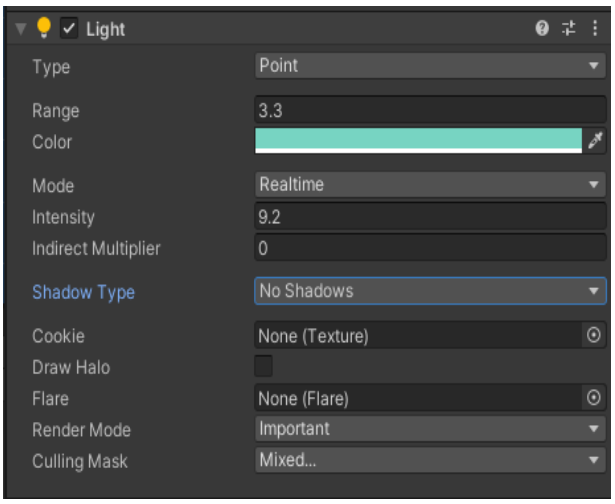


Ilustración 51: Propiedades de la ilustración 50.

Pruebas

1.- Rendimiento de la CPU y GPU

En esta prueba se va a comprobar cuánto afecta a la CPU y las GPU ambas iluminaciones.

Para empezar, hay que saber realmente qué es la CPU y la GPU.

Según la Wikipedia: “CPU (Central Processing Unit) es el hardware dentro de una computadora u otros dispositivos programables. Su trabajo es interpretar las instrucciones de un programa informático mediante la realización de las operaciones básicas aritméticas, lógicas y externas (provenientes de la unidad de entrada/salida”.

Según la Wikipedia: “GPU (Graphics Processing Unit) es un coprocesador dedicado al procesamiento de gráficos u operaciones de coma flotante, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos o aplicaciones 3d interactivas”.

En resumen, en el ámbito de la programación para videojuegos, la CPU sería el encargado de hacer los algoritmos y la GPU se encarga de leer esos algoritmos y traducirlos gráficamente. Estos dos componentes son fundamentales para pintar gráficos por pantalla.

Se va a realizar la prueba a través de los stats que proporciona Unity, siendo estos la ilustración 52 y 53.

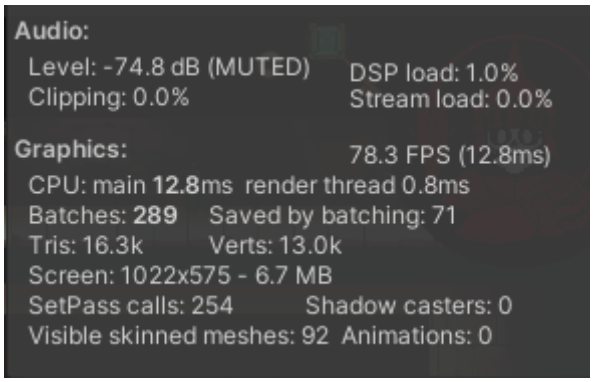


Ilustración 53: Stats URP

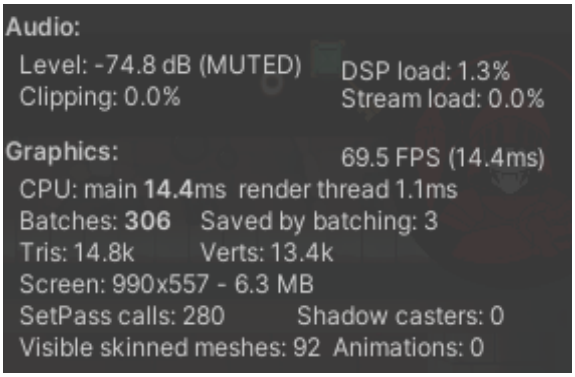


Ilustración 52: Stats Built-in

Con estos datos, se puede ver como en el apartado de la CPU, se ha tardado menos en procesar la actualización del marco de su aplicación, así como el tiempo que ha tardado el Editor en actualizar la vista de la escena, y como se renderiza en menos tiempo la vista del juego, con el Universal Render Pipeline que el Built-in Render Pipeline. El URP tarda en procesar un fotograma 12.8 ms y en renderizarlo 0.8 ms, mientras que el BRP tiene un tiempo de 14.4 ms en procesar un fotograma y en renderizarlo un 1.1 ms.

Esto se debe a que haya menos pasadas a la hora de sombrear en el primer render que en el segundo, con una diferencia de treinta y seis pasadas (URP ha hecho 254 pasadas, mientras que BRP ha hecho 280 pasadas). También, es importante el dato de batches[5], siendo más pequeño en el URP con 289, que con el Built-in Render Pipeline que tiene 206.

Otra forma de conocer datos sobre la CPU y GPU es a través de la ventana de Analysis: Profiler.

Se puede ver los gráficos del rendimiento sobre la GPU que hace la iluminación, en las ilustraciones 54 y 55.

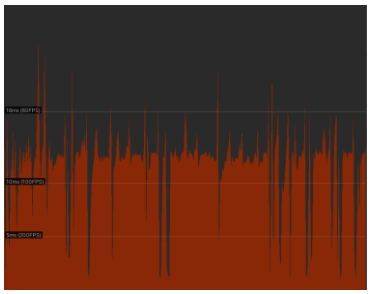


Ilustración 55: GPU Built-in Render

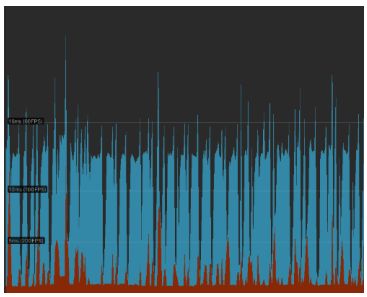


Ilustración 54: GPU URP

Como se puede ver, en la primera imagen, la gráfica de la GPU, con el Built-in Render, está todo el tiempo renderizando objetos invisibles, mientras que, en el Universal Render, es todo parte del render, como secuencias de comando.

Se puede ver los gráficos del rendimiento sobre la GPU que hace la iluminación, en las ilustraciones 56 y 57.

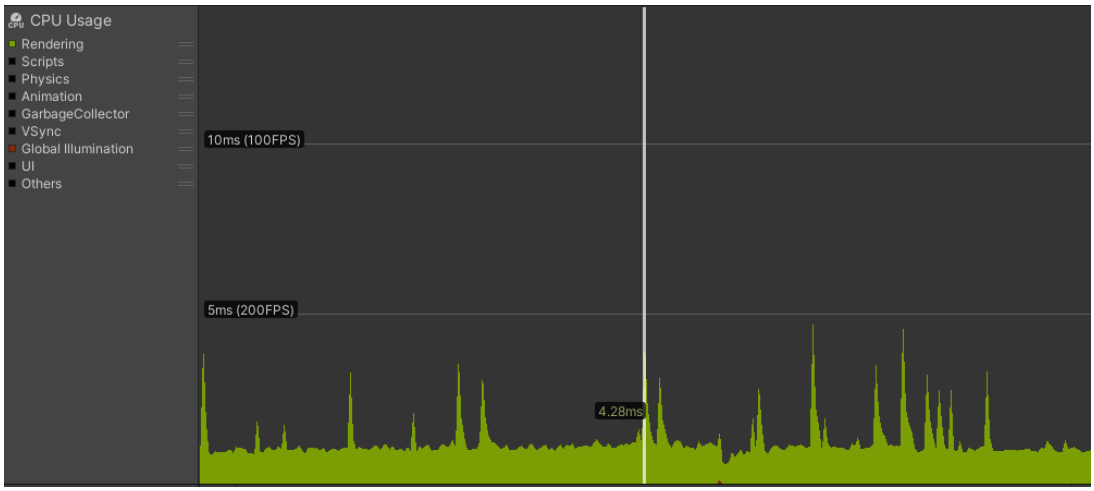


Ilustración 56: CPU URP

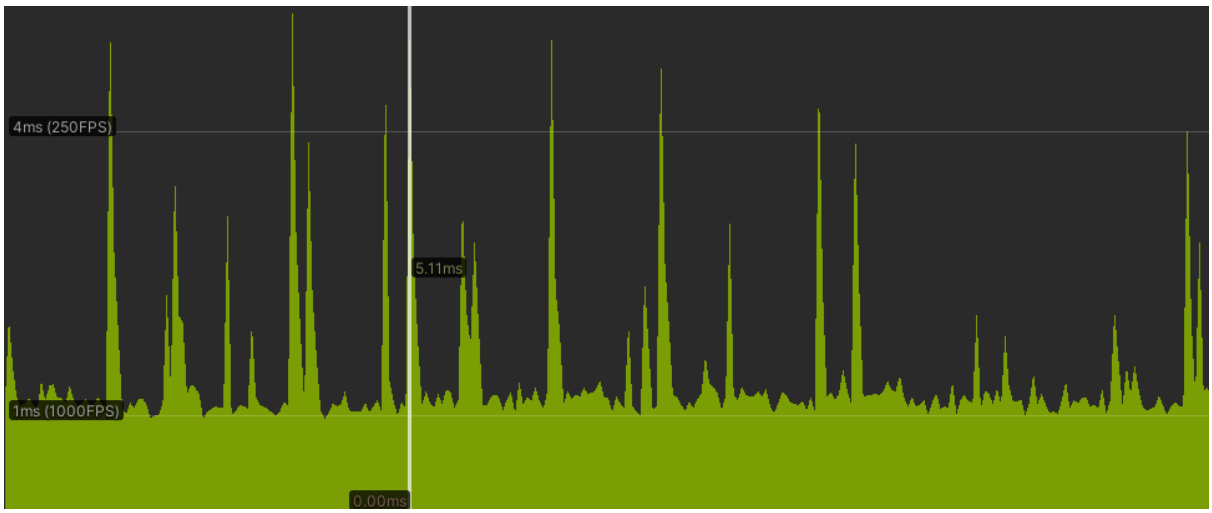


Ilustración 57: CPU Built-in

Como se puede apreciar, hay picos más altos afectando a la CPU en el Universal Render Pipeline que en el Built-in Render, esto es debido a lo que se comentó anteriormente de las pasadas que hace el render para calcular las sombras.

Además, esta ventana, detalla más los datos dichos anteriormente, como se puede ver en las siguientes ilustraciones:

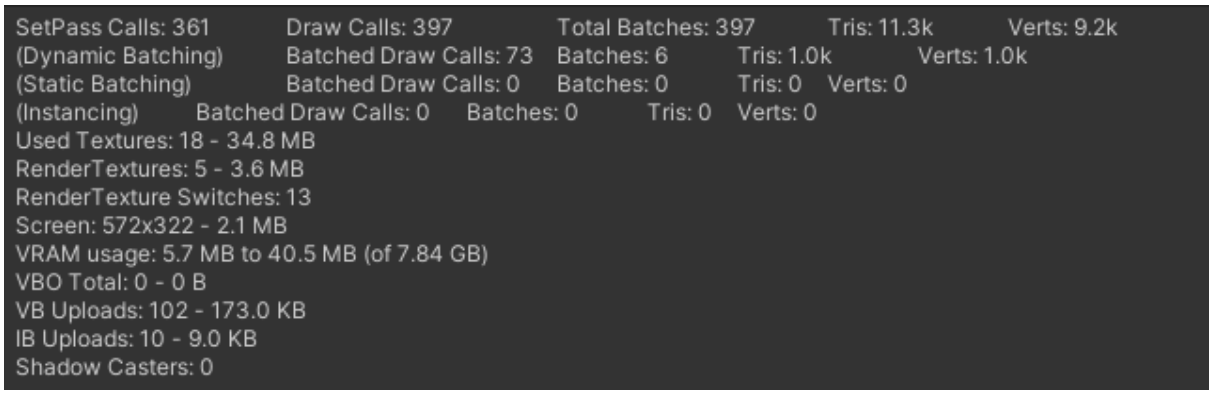


Ilustración 58: Stats_2 URP

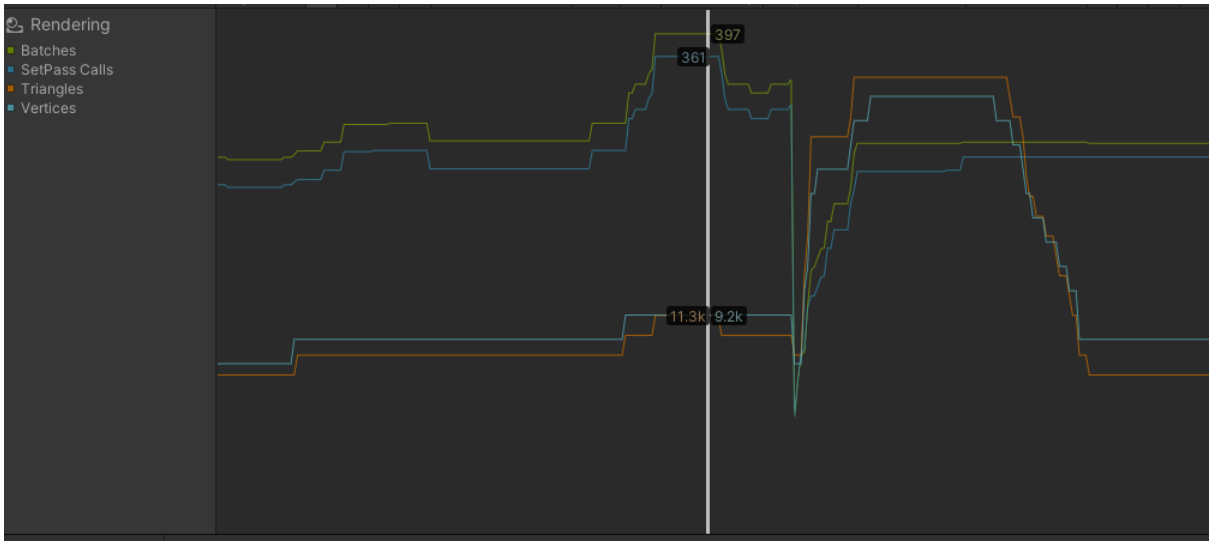


Ilustración 59: Stats_2 URP

Ilustración 60: Stats_2 Built-in

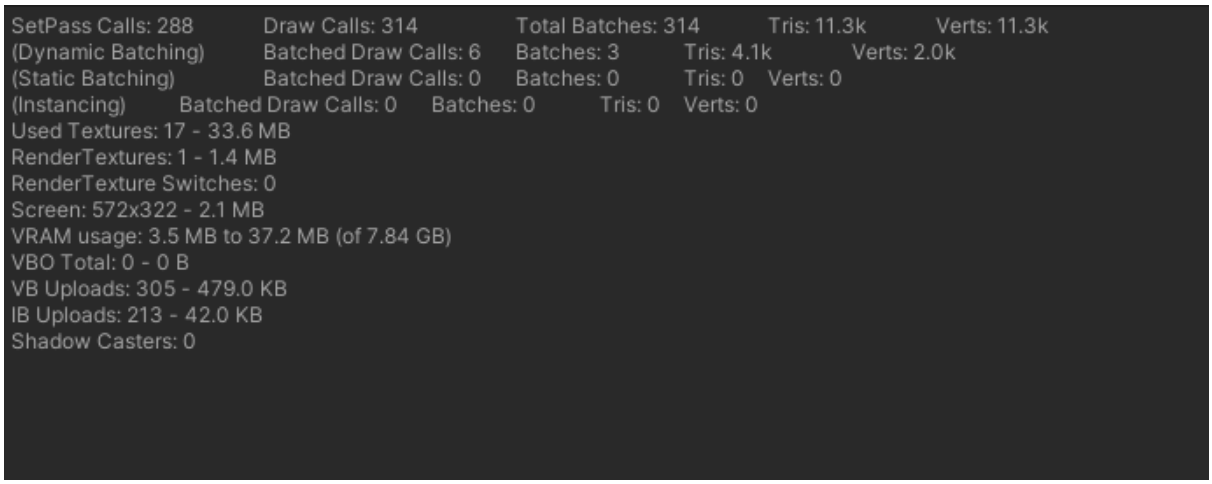


Ilustración 61: Stats_2 Built-in

2.- Facilidad de uso.

En cuanto a la facilidad de uso de cada iluminación, ambas son sencillas de manejar, pero en su propio ámbito. La iluminación del Built-in Render se utiliza para escenas en tres dimensiones mientras que el Universal Render Pipeline tiene una iluminación propia de dos dimensiones. Si se habla en el mismo contexto, al principio es más fácil usar Built-in Render, por una simple razón, el URP necesita ser descargado, a no ser que comiences un proyecto de cero con este render, y para cambiar el render, se necesita crear varios objetos: el Universal Render Pipeline Asset y el Renderer 2D Data.

Pero una vez están los dos en el mismo punto, la hora de implementar iluminación a la escena, es más fácil el Universal Render Pipeline, debido a que los tipos de luz tienen más propiedades lo que lo hace más versátil, más opciones de iluminar la escena. Además, no es necesario prestar atención al eje Z, mientras que con el Built-in Render, si es necesario debido a que dependiendo en qué zona del eje Z este la luz y el sprite, puede ser iluminado o no. Y, por último, visualmente se ve mejor en cuanto a los gizmos que tiene, y en cuanto a la capa sobre la que actúa, pues a cada luz, se le puede poner sobre que objetos puede actuar.

3.- Comparación de características entre ambos Render

Según la tabla realizada por Pierre Yves Donzallaz, donde muestra la comparación entre los distintos Render Pipeline en la versión 2019.3 de Unity, Built-in Render sería compatible con todas las plataformas, mientras que Universal Render Pipeline, solo sería con móvil, XR (Realidad extendida[6]) y PC/Consolas. Otro aspecto a tener en cuenta es el Rendering Path, con el render Built-in, hace varias pasadas a la hora de sombrear (tanto en forward

(más rápido) como deferred (menos costoso, pero más lento), mientras que con el Universal Render hace una única pasada lo que reduce significativamente el costo de la CPU.

En cuanto a los modos de luz, existen varios tipos de luz mixtas Substractive (Según la API de Unity, este modelo no proporciona efectos de iluminación realistas, pero es muy útil en juegos donde el rendimiento es una preocupación, y solo se necesita una luz de proyección de sombras en tiempo real.), Baked Indirect (Según la API de Unity, este solo calcula previamente la iluminación indirecta y no realiza cálculos previos de sombras. Además, las sombras son en tiempo real si se encuentra dentro de la distancia de sombra, más allá de esta, no habrá sombras. Se podría decir que actúa igual que el frustrum culling[7]), Shadowmask Mode (Según la API de Unity, es un modo que combina iluminación directa en tiempo real con baked indirect, la diferencia está en cómo representan las sombras, ya que este modo usa un light map texture conocida como máscara de sombras, de esta manera renderiza también sombras en la distancia). Descrito los distintos tipos de luz mixta, el Built-in Render Pipeline es compatible con todos los tipos de luz mixtas, mientras que el Universal Render Pipeline solo es compatible con Substractive y Baked Indirect.

En cuanto a las entidades de luz que se pueden tener dentro de la escena, ambos pueden tener luces no basadas en físicas, y tener luces realtime de los tipos Directional, Spot y Point light. Las diferencias son que Built-in Render Pipeline admite con el path rendering forward 8 luces por píxel, y con el path rendering deferred admite un número ilimitado de luces. Además, admite un máximo de luces por vértices de 4. Mientras que el Universal Render Pipeline admite una luz direccional más 8 por objeto y 256 por cámara (32 por móvil).

Característica	Built-in Render Pipeline	Universal Render Pipeline
Compatibilidad de plataforma	Todas las plataformas.	- Móvil - XR - PC/Consola

Modos de luz	- Baked
	- Mixed:
	  - Substractive
	  - Baked Indirect
	  - Shadowmask Mode
	- Realtime
		- Baked
		- Mixed:
		  - Substractive
		  - Baked Indirect
		- Realtime

Entidades de luz	- Luces no basadas en	- Luces no basadas en

	físicas. - Realtime Directional, Spot y Point Light. - Máximo de luces en el path rendering forward: 8 - Máximo de luces en el path rendering deferred: ilimitado. - Máximo de luces por vértice: 4	físicas. - Realtime Directional, Spot y Point Light. - Máximo de luces direccionales: 1 más 8 por objeto. - Máximo de luces por cámara: 256 (32 por móvil)

Conclusiones.

La iluminación Built-in Render tiene muy buenos beneficios como que es multiplataforma (es compatible con cualquier plataforma), permite tener más luces en escena, y permite tener más modos de luz. También, tiene menos filtros de post-procesado, pero porque intenta empaquetar propiedades de muchos filtros en uno, como se puede ver en el filtro de Color Grading que recoge el Tonemapping, Trackballs, etc. Pero en contra parte, afecta bastante al rendimiento de la CPU y GPU debido a su alto número de pasadas a la hora de crear las sombras. Además, es una iluminación en tres dimensiones, motivo por el que es difícil acoplar esa iluminación a gráficos en dos dimensiones.

La iluminación Universal Render Pipeline tiene muy buenos beneficios como el número de pasadas para crear sombras, lo que provoca que haya un menor rendimiento de la CPU y GPU. Trae consigo numerosos filtros de post-procesado que permite realizar cualquier distorsión de la escena, y algo muy importante de este render es que tiene su propia iluminación única para gráficos en dos dimensiones lo que permite tener una iluminación más realista que con una iluminación en tres dimensiones. Pero, tiene varios inconvenientes, como que existe poca información debido a que esta iluminación es reciente, y está mucho más limitada que el Built-in Render en aspectos como luces por escena o las plataformas con las que son compatibles.

Gracias a las pruebas que se han realizado entre la iluminación de Built-in Render con la iluminación de Universal Render Pipeline se puede deducir que la primera iluminación en cuanto a costes de rendimiento de la CPU y GPU es mucho más costosa que la segunda iluminación. Además, es menos realista, pues intenta camuflar una iluminación en tres dimensiones como si fuese de dos dimensiones, y no queda un aspecto realista, como si queda con el Universal Render Pipeline. Por último, decir que la iluminación de Built-in tiene beneficios como ser compatible con todas las plataformas mientras que el Universal Render Pipeline no lo es.

Metodología y organización

El método usado en la idea de investigación ha sido inductivo porque se basa en la observación de hechos como las estadísticas que representan el rendimiento de la CPU y GPU, Pero al mismo tiempo, establece un método deductivo pues se razona estableciendo conclusiones a partir de generalizaciones. Ambas son útiles, pues se utiliza el segundo método antes de realizar cualquier tarea del TFG para saber por dónde empezar y una vez realizada la tarea, se sacan conclusiones de la tarea que se ha terminado.

Se ha usado un tipo de investigación documental, pues toda la información ha sido documentada por varios profesionales del motor gráfico de Unity o profesionales que hayan trabajado en el propio Render/iluminación.

Los pasos que se ha seguido a la hora de realizar cada tarea:

1.- Buscar información sobre aspectos de los que consiste la práctica, por ejemplo, en el caso de la idea de investigación, primero se ha buscado información sobre los distintos aspectos de la iluminación como post-procesado o tipos de luz.

2.- Intentar realizar una prueba para ver cómo funciona, por ejemplo, en la idea de investigación, se ha hecho una prueba con cada post-procesado para ver cómo funciona, que cambios realiza en la escena, para que sirve cada propiedad, o que se podría realizar con ese objeto.

3.- Describir las sensaciones que se han tenido de la prueba realizada en el paso número dos. Si se necesita sacar datos como el rendimiento de CPU y GPU entonces se buscará alguna herramienta o aplicación que apoye la descripción.

Dentro del proyecto se ha desarrollado el rol de technical artist, desempeñando funciones como crear la iluminación del juego, programar varias mecánicas del juego como habilidades o trampas, o introducir el nuevo input System.

En cuanto a como se ha organizado el departamento de programación, en todo momento ha existido una comunicación. Ese ha sido el punto fuerte, porque cuando una persona no tenía tiempo para una tarea específica, y la otra persona del departamento podía realizarla, le quitaba trabajo a la otra persona dejándola con menos responsabilidad y viceversa. En todo momento se estaban cubriendo las espaldas. A la hora de organizarnos utilizamos al inicio un Excel con todas las tareas del equipo, pero a medida que iba pasando el tiempo se quitó por inactividad en otros departamentos y se empezó a usar la aplicación Trello.

En cuanto al calendario, los seis primero meses del desarrollo del proyecto se dedicaron al desarrollo de las habilidades principales del jugador, así como su movimiento, y las trampas que componen los niveles. Después de

esos seis meses, se dedicaron tres meses a la solución de bugs, donde el departamento de programación se puso en conjunto para sacar estas tareas lo más rápido posible. Por último, estos dos meses finales, se han dedicado a la introducción de las ideas de investigación como por ejemplo la iluminación.

Riesgos que se han encontrado a lo largo del desarrollo pueden ser: usar una misma rama del repositorio al mismo tiempo, lo que podría llegar a ocasionar algún conflicto a la hora de hacer algún merge. Esto se solucionó con mucha comunicación por parte del departamento y del grupo en general. Otro riesgo que se encontró fue el repetir varios códigos, pues muchas veces al tocar código de otra persona sin que se haya explicado anteriormente puede liar más el código, lo que supone horas extras a la producción.

Desarrollo del proyecto

Resultados, conclusiones y líneas de futuro

Input System

Para este proyecto, se ha usado el nuevo Input System, debido a que era más sencillo a la hora de realizar el input para cada device (teclado y/o mando). Este nuevo sistema de input separa la entrada del dispositivo de las acciones de código, no necesitando saber qué dispositivo está usando el jugador o en qué botón está haciendo click, centrando únicamente en manejar las acciones que desencadenan los jugadores.

La diferencia con el antiguo sistema input es que verifica la entrada de diferentes dispositivos para determinar si los jugadores tomaron una acción.

Según dice Por Ken Lee, el nuevo sistema tiene cuatro bloques de construcción que canalizan los eventos de los dispositivos de los jugadores a su código de juego:

- Archivos de acción de entrada: archivo de configuración que contiene las propiedades de las acciones y sus enlaces asociados.
- Acciones: definen los significados lógicos de la entrada.
- Enlaces: describen la conexión entre una acción y los controles del dispositivo de entrada.
- PlayerInput: script que administra y vincula eventos de acción a la lógica del código correspondiente.

La estructura de este nuevo input es:

1.- Unity Engine recopila la información de los devices (dispositivos) conectados y envía los eventos correspondientes.

2.- El Input System traduce esos eventos en acciones, según la acción y la información vinculante que se encuentra en los mapas de acción.

3.- Por último, se pasan las acciones al script PlayerInput, que invoca los métodos correspondientes.

Para ver un esquema con toda la estructura del nuevo Input System, mirar la ilustración

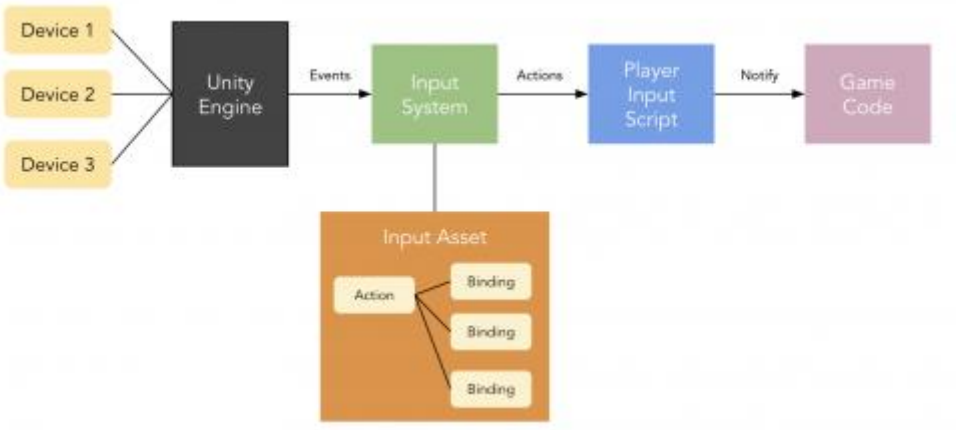


Ilustración 62: Estructura del nuevo Input System

Los pasos que se han realizado para conseguir introducir el nuevo Input System en el proyecto de Necrocandor son:
1.- Crear un InputActions que contiene todas las acciones que el jugador puede realizar dentro del juego.

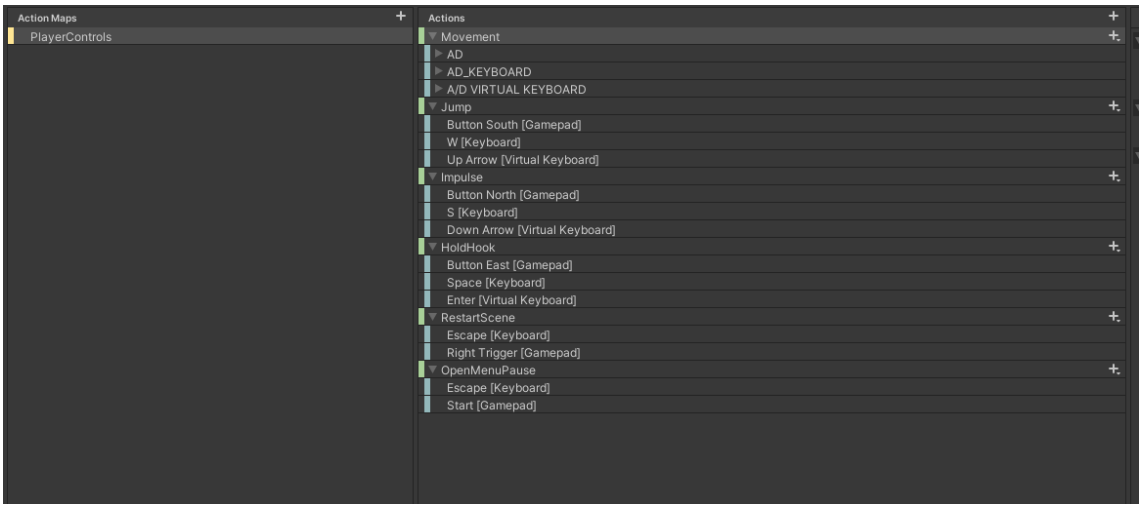


Ilustración 63: InputAction

2.- Añadir el componente de Player Input al jugador, y las acciones van a ser a través de Callbacks. Por lo que, la opción de behaviour tiene que ser Invoke Unity Events.

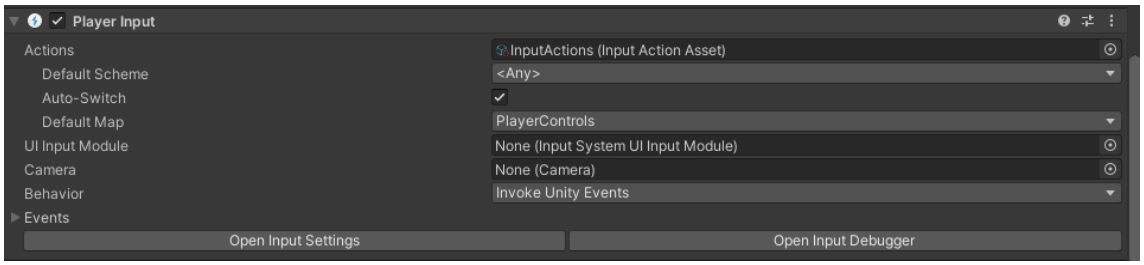


Ilustración 64: Player Input

3.- Crear las funciones en el script del PlayerController, que luego se le asignará en los eventos que tiene puesto el player Input.

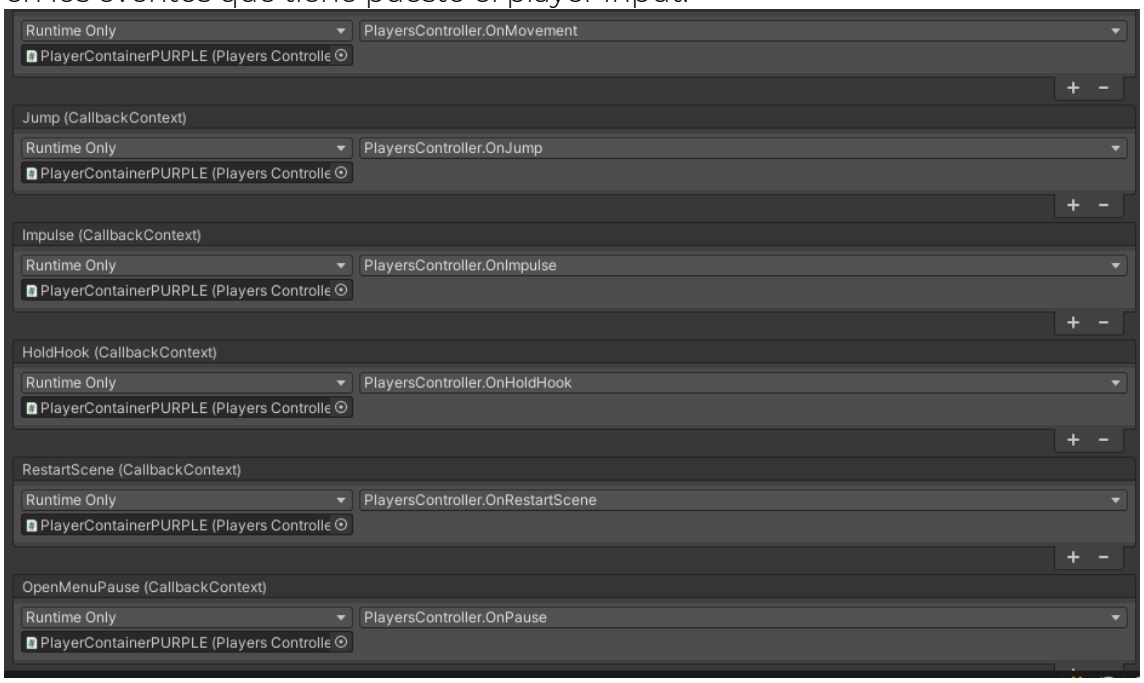


Ilustración 65: Eventos del PlayerInput

La dificultad que tuvo el añadir el nuevo Input System en el juego, es que, al haber dos personas, los devices no se ajustan a los propios jugadores. Esto se solucionó haciendo que los jugadores eligieran el dispositivo con el que van a jugar en el menú de selección de personaje. A la hora de spawnear los personajes, se llama a esta función pasándole el device. Al jugador 1, le corresponde los devices 0 (teclado) y 2 (mando), mientras que al jugador 2 le corresponde los devices 1 (teclado) y 3 (mando).

```
public void SetUpPlayer(int newPlayerID, int device)
{
    Id = newPlayerID;

    //currentControlScheme = playerInput.currentControlScheme;
    playerInput.SwitchCurrentControlScheme(InputSystem.devices[device]);

}
```

Ilustración 66: Función de selección de device

Pero hay una condición que es: si el jugador 1 no ha escogido mando, pero el jugador 2 si, entonces, el device del jugador 2 cambia de 3 a 2, para que pueda jugar con el mando número 1 en vez de con el mando número 2.

```
void SetUpActivePlayers()
{
    for (int i = 0; i < activePlayerControllers.Count; i++)
    {
        //Se llama a la función del script de PlayersController SetUpPl
        if (i == 0)
        {
            if (device1 == 0)
                Controller1_Open = true;

            activePlayerControllers[i].SetUpPlayer(i, device1);
        }
        else
        {
            if (Controller1_Open && device2 == 3)
                device2 = 2;
            activePlayerControllers[i].SetUpPlayer(i, device2);
        }

        //activePlayerControllers[i].GetComponent<SpriteRenderer>().col
    }
}
```

Ilustración 67: Función del Game Manager sobre los device

La programación de los devices tiene un problema, y es que solo funciona cuando pasas por el menú de selección, pero antes, solo puede manejar el jugador 1. Esto se intentará solucionar en un futuro.

HABILIDADES

Dentro del juego Necrocandor se encuentran múltiples acciones que puedan realizar los personajes. Estas mecánicas son: agarrar y lanzar al personaje, lanzar un gancho, usar el impulso, o clavarse en las paredes y techos.

Gancho

Esta habilidad consiste en lanzar un gancho, una herramienta que tiene por objetivo principal ayudar a los jugadores a desplazarse con mayor rapidez por el escenario. El funcionamiento de esta mecánica cambia dependiendo de la situación, si el jugador pulsa la Q sin pulsar ninguna dirección (teclas A o D), saldrá disparado en la dirección en al que se encuentra mirando el personaje. Y si el jugador mantiene pulsado la Q y pulsa una dirección (teclas A o D), entonces saldrá disparado en la dirección que ha elegido el jugador.

Esta mecánica ha ido cambiando bastante a lo largo del desarrollo del proyecto. En un principio había una tercera dirección, hacía arriba, lo que generaba bastante libertad al jugador, y se eliminó debido a que daba ciertos errores con otras mecánicas.

Otro cambio fue, que en un principio el gancho se iba a realizar estando el jugador en el suelo, pero se añadió que el personaje pudiera lanzar el gancho en el aire. Esto provocó un gran cambio en la programación de la mecánica, ya que se tuvo que hacer varios estados para el gancho, y en sus inicios, dio problemas, ya que en el aire no se quedaba del todo quieto, debido a la gravedad, lo que hacía que el gancho, tuviera varios bugs, ya que el inicio y el final de este, cambiaban de posición, haciendo que se fuera para abajo, o para arriba indefinidamente. Este problema, se solucionó poniendo la gravedad a 0, y la velocity del jugador a cero, y diciéndole a través de un booleano, si estaba quieto en el aire (comprobando su velocity en el eje Y), y si estaba quieto podía lanzar el gancho. Una vez, el gancho ha vuelto al jugador, ha hecho el desplazamiento, al jugador le volverá a actuar la gravedad inicial.

El último cambio que se hizo respecto a esta mecánica, y que influyó en la mecánica de agarrar y lanzar, fue que se unificaron en un mismo botón. Haciendo que el gancho se lanzará si había una cierta distancia respecto al otro jugador, y si estaban muy cerca, hacían la acción de agarrar y lanzar. Esto hizo que el departamento de programación se uniera para conseguir realizar esta tarea, ya que las mecánicas del juego se repartieron entre los miembros de esta área. Esta tarea se solucionó con un collider, que indica si el jugador está cerca o lejos, de esta manera, cuando se pulsa el botón, comprueba la distancia y ejecuta una acción u otra.

Las situaciones en las que el jugador no pueden utilizar el gancho son cuando tienen la habilidad en cooldown, se encuentran aturdidos o clavados en el techo o pared, cuando están siendo agarrados o cuando el jugador una vez lanzado por el otro jugador se encuentra en el aire.

En cuanto al propio gancho, es un GameObject que se instancia a través de una función que se llama en la animación de hook, de esta manera, parece más realista, porque sale del dedo del personaje, en el momento clave. A este gancho, se le pasa una dirección (derecha o izquierda) y lanza el gancho en esa dirección pasada, dándole una fuerza, hasta que supera un rango definido o choca con algún objeto o personaje. Dependiendo con lo que choque, le da una fuerza contraria al gancho, o le aplica una fuerza al objeto chocado.

La programación de esta mecánica está basada en el desarrollo del gancho usado por el personaje del Overwatch Roadhog.

Impulso

Esta habilidad consiste en dar un salto más alto de lo habitual, permitiendo llegar al personaje a mayores alturas que como lo haría con un salto normal, pero sin tener tanto control horizontal. Esta habilidad se divide en dos fases, la canalización y el salto. En la fase de canalización, el jugador tiene que mantener pulsado el botón de impulso, dependiendo del tiempo mantenido, se aplicará más fuerza o menos fuerza al personaje. Cada evolución se verá en el personaje de alguna manera, puede ser a través de una animación que se vaya agachando más, o a través de un sistema de partículas. Y la segunda fase, es cuando se aplica la fuerza correspondiente al jugador, y este se encuentra en el aire, quitando movimiento horizontalmente. La altura máxima que alcanza depende de cuánto tiempo mantuvo el jugador apretado el botón.

> Durante el desarrollo de esta mecánica, hubo varios cambios, llegando al punto de casi eliminarse completamente del juego. Esto es debido a la gran libertad de movimiento que tenía el jugador en el aire, cualidad que se capó tras un mes de desarrollo. Y el otro gran punto fue, que se usa la misma tecla para bajarse de la plataforma que para el impulso, lo que hizo que se volviera a reestructurar en el mapa de acciones, el apartado del impulso, haciendo que sea tap interaction o slow interaction. De esta manera, el input detecta si se ha pulsado más o menos tiempo, y dependiendo de este, el jugador realizará una acción u otra.

Al pulsar el botón el jugador, llama a la función de OnImpulse, que comprueba si el estado en el que se encuentra permite hacer al jugador la habilidad. Luego comprueba la interacción con el botón, para ello usamos dos comandos:

- value. interaction is UnityEngine.InputSystem. Interactions.TapInteraction
- value. interaction is UnityEngine.InputSystem. Interactions.HoldInteraction

TapInteraction ocurre cuando el usuario presiona y suelta el botón, y HoldInteraction ocurre cuando el jugador mantiene pulsado el botón. Si el jugador presiona y suelta el botón, se comprueba si está encima de una plataforma, y si es true, se le quitará el collider, para bajar de la plataforma.

En cuanto al holdInteraction, si el jugador mantiene pulsado, se comprueba si está colisionando con el suelo, y si se verifica esta afirmación, el jugador no podrá moverse, y por cada tick que esté pulsando el botón, se estará llamando a una función que recoge el estado de la animación, y cuando suelta el botón, le aplicará una fuerza dependiendo del tiempo pulsado. Cuando está en el aire, se le reduce el movimiento en el aire, mediante un booleano, que divide la máxima aceleración que tiene en el aire por el valor de la reducción de movimiento:

```
if (!OnJumpImpulsed)
    maxChange = maxAcceleration * Time.fixedDeltaTime;
else
    maxChange = (maxAcceleration / ReductionMovement) * Time.fixedDeltaTime;
```

Ilustración 68: Función de cuando el personaje está en el aire una vez usado el impulso

TRAMPAS

Una de las mecánicas más importantes del juego, es esquivar las trampas que contiene cada nivel. En función del papel que cumplen y de su visibilidad, podemos dividirlas en tres tipos diferentes: trampas simples, de oportunidad y ocultas.

- Las trampas simples son totalmente visibles y lucen un aspecto de peligro, para que así el jugador se dé cuenta de que es nocivo para él. Su función principal es obligar a desplazarse con cuidado por el nivel, y ponerles límites a sus habilidades.

- Las trampas de oportunidad son un conjunto de trampas distribuidas de tal manera que el jugador tenga que adivinar el patrón para poder avanzar. La idea de este tipo de trampas es obligar al jugador a usar sus habilidades para poder pasar a través de estas.

- Las trampas ocultas se encuentran no visibles, camuflándose con el entorno, de modo que pille de sorpresa a los jugadores. Su objetivo principal es frenar al que va delante del otro, durante el nivel, dando más posibilidades de alcanzar al jugador más atrasado al primero.

Trampa Para Osos

Este tipo de trampa se puede utilizar tanto en techos como en suelos, y permanece oculta de la mirada de los jugadores salvo por unos pequeños pinchos que sobresalen de la superficie sobre la que está colocada. Si un jugador pasa por encima de ella, después de dos segundos la trampa se activa y se cierra en forma de pinza desde dentro y matando todo lo que haya tocado a su paso. Después, la trampa vuelve lentamente a su posición original (3 segundos) y vuelve a estar funcional, de modo que si cualquier jugador vuelve a tocarla se activará.

Desde el principio se ha intentado hacer lo más global posible, para que, si el diseñador quiere darle otro uso, no se tuviera que cambiar una gran cantidad de la programación. Esta trampa consiste en una corrutina que se activa cuando un personaje entra en contacto con la trampa, haciendo que la trampa ejecute la animación de ataque. Animación que tiene dos eventos, uno está al principio donde pone el valor de la trampa a true, y otro al final que pone el valor a false. Este valor es muy importante porque si el personaje

sale de la trampa, y esta está activada, le matará haciéndolo spawnear en un punto cercano que esté antes de la trampa.

Trampas Muelle

Este tipo de trampa se encuentra únicamente en suelos y permanece oculta de la mirada de los jugadores salvo por una pequeña tonalidad de color distinta a la del suelo. El funcionamiento de esta trampa es a través del contacto, si el personaje pisa la trampa, este sale despedido con la misma potencia que saldría con la habilidad impulso cargada al máximo.

Existen tres tipos de trampa muelle:

- Impulso hacia la izquierda: el jugador sale despedido en un ángulo de 45 grados hacia la izquierda.
- Impulso hacia arriba: el jugador sale disparado hacia arriba.
- Impulso hacia la derecha: el jugador sale despedido en un ángulo de 45 grados hacia la derecha.

En cuanto al desarrollo de esta trampa, ha tenido varias fases, en un principio, se programó, para cuando se colisionara con cualquier objeto, hiciera una animación que tendría un evento para aplicar una fuerza en una dirección, al jugador con el que estuviese colisionando. Pero se cambió por algo más directo, más rápido, porque la trampa aplicaba la fuerza demasiado lento, y queríamos que fuera más fluida. Entonces, se cambió el orden, se aplica la fuerza instantáneamente el jugador colisiona con la trampa, y se ejecuta la animación de que ha sido activada.

Trampa de flechas

Este tipo de trampa suele estar ubicada en las paredes de la mansión. Está formada por tres aperturas dispuestas verticalmente por donde salen disparadas con una fuerza determinada las flechas. Cada vez que spawnean tres flechas, se disparará una flecha por cada apertura al mismo tiempo.

En cuanto a la programación de esta trampa, se divide en dos partes: la base y la flecha. En cuanto a la base, esta se compone de tres gameObject vacíos que sirven como lugares donde se spawnea las flechas. Para ayudar al diseñador, y que pueda cambiar el patrón que representa los lugares por donde van a salir las flechas, se hizo un array de int. Esta trampa siempre funciona con una corrutina, que, si se acaba, se vuelve a reiniciar. Además, tiene una pool de objetos, para que sea óptimo, y funciona de tal manera, que se van cogiendo objetos de una lista de GameObjects, y los va spawneando. Se reciclan los gameObjects, cuando no quedan más balas en la recámara. Por último, la flecha, que tiene comportamiento de bala, nada más spawnearse, sale con una fuerza determinada. Si choca con un jugador, llamará a la función de Damage de su caster (en este caso, la base de la trampa de flechas), que hará spawnear al personaje en un punto determinado (punto que puede elegir el diseñador) y la bala se desactivará, en vez de ser destruida, para luego podrá rehusarse.

Anexo de programación

Plataformas

- Windows 7 o superior
- Mac OS X
- Nintendo Switch

Windows y Mac son dos de los sistemas operativos más populares en el mercado, además, se ha planteado la portabilidad del juego a Nintendo Switch para lo que se compraría un kit de desarrollo. Sus mandos separados y portátiles se coordinan perfectamente con nuestros comandos simples. De esta manera los jugadores de Switch siempre tendrían dos mandos y podrían jugar una partida rápida en cualquier lugar.

Gráficos

- 2D
- Animación por huesos
- Cartoon

Los gráficos en dos dimensiones Cartoon se adaptan al estilo desenfadado que se le quiere dar al juego. Además, la animación por huesos hará la producción de animaciones más rápida y sencilla.

Almacenamiento

- Ejecutable: 211 MB.
- Proyecto: 1.12 GB.

Periféricos

- Mando/Joycon: Mejor experiencia de usuario.
- Teclado dividido para ambos jugadores: Universalidad de controles.

Engine y herramientas

- Unity 2020.1.9
- Herramienta de generación de niveles: herramienta que podrá ser utilizada por el usuario para generar niveles por procedimientos aportando al proyecto gran rejugabilidad y variedad ya que permite que se creen desde salas personalizadas a más variadas.
- Herramienta de creación de niveles: Herramienta que permite crear niveles pasándole los assets necesarios del juego.

Manual de Usuario

Para poder ejecutar Necrocandor es necesario descargarlo de la página oficial de itch.io.

Posteriormente ha de descomprimirse y hacer doble click en el archivo ejecutable.
https://david-martin-almazan.itch.io/necrocandor

Librerías / plugin

- Cinemachine: herramienta verificada de Unity que trae consigo un conjunto de características a la cámara de Unity. Resuelve las complejas matemáticas y la lógica del seguimiento de objetivos (mediante composiciones de la escena).
- Anima2D: herramienta verificada de Unity que permite animar sprites mediante huesos para conseguir animaciones más fluidas y realistas.
- Doxygen: Herramienta utilizada para la elaboración de la documentación de código fuente. Para su visualización, se ha de entrar en la carpeta Documentación->HTML y clicar en el archivo HTML llamado index.
- TextMeshPro: Herramienta utilizada para la personalización de los textos de las interfaces de usuario.
- Universal Render Pipeline (URP): al no haber empezado el proyecto con este render implementado se ha tenido que instalar dentro del proyecto. Este propio render trae consigo un modelo de iluminación propio en dos dimensiones.

Diagrama de componentes

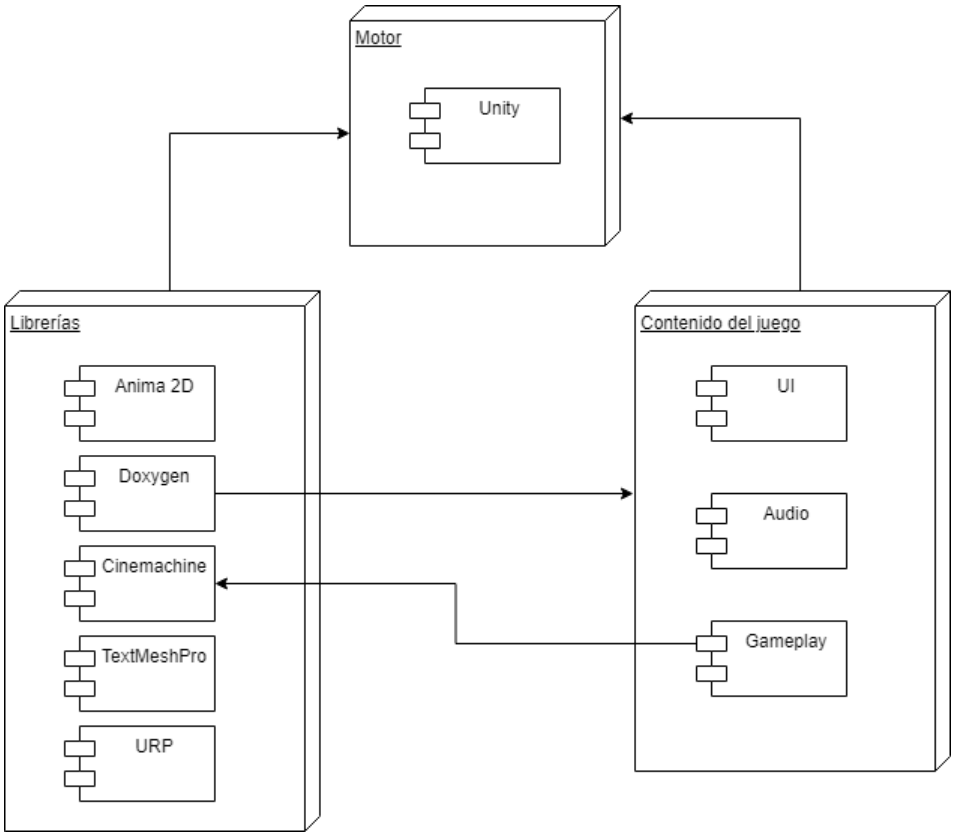


Ilustración 69: Diagrama de componentes

El motor gráfico usado ha sido Unity 3D en la versión 2020.1.9, y se han usado las librerías de Anima2D que permite realizar animaciones por hueso, Doxygen que administra y genera la documentación del código, Cinemachine que trae numerosas funciones y características nuevas a la cámara de Unity, TextMeshPro para una personalización más sofisticada de los textos y, por último, URP que trae consigo un modelo de iluminación en dos dimensiones propio.

Diagrama de clases

Control Personaje

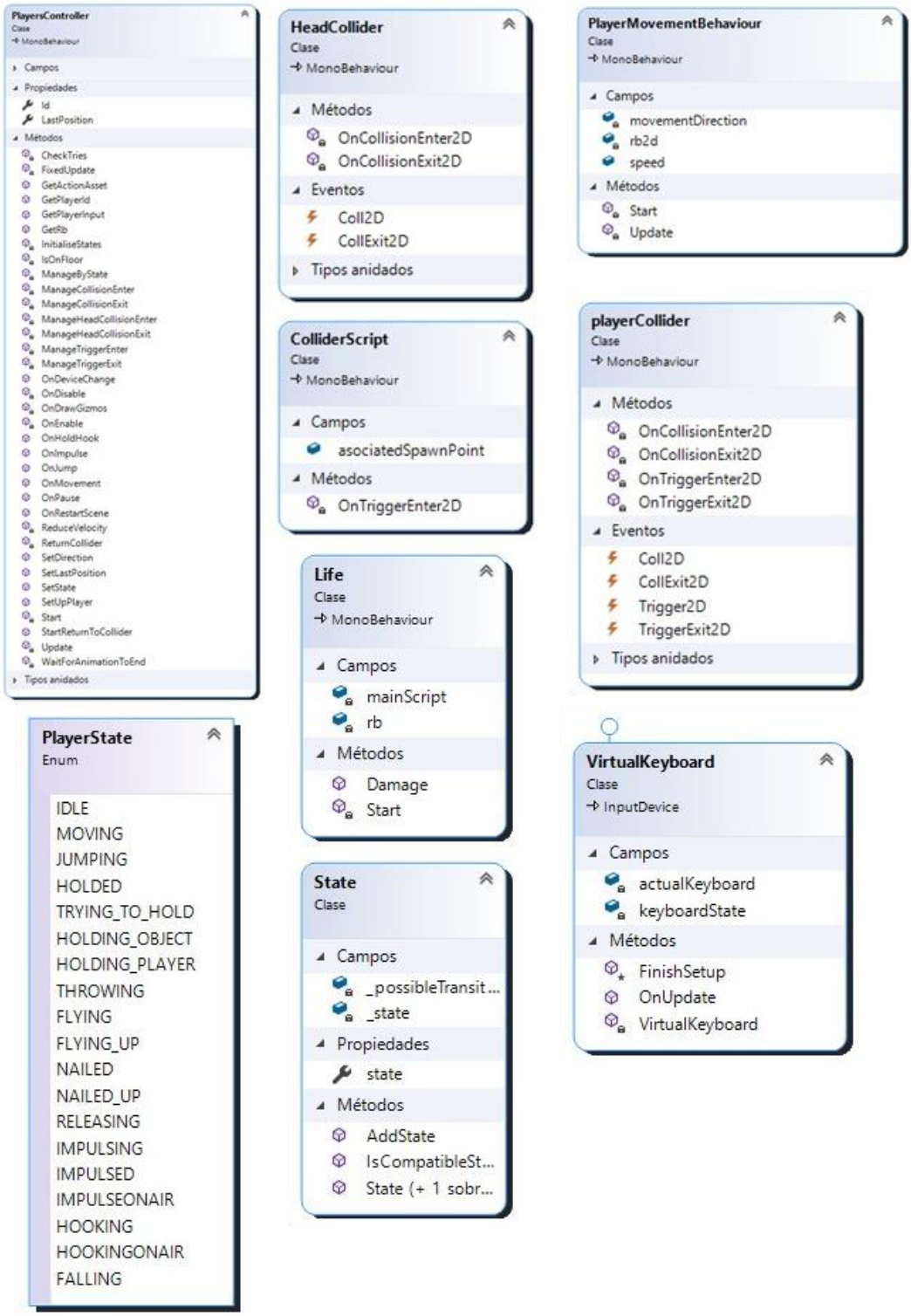


Ilustración 70: Diagrama de clase. Control Personaje

El trabajo realizado fue el script de Virtual Keyboard, que sirve para que dos jugadores puedan utilizar el teclado al mismo tiempo, de esta manera, es como si estuvieran jugando en dos teclados al mismo tiempo, así no se interrumpe ninguna acción. Por parte del player controller, contiene el movimiento de los personajes, las funciones que corresponden con las habilidades del personaje, y como se conectan los jugadores al nuevo Input Action, asignándoles un device (mando o teclado). Por último, los estados de las principales habilidades, en este caso, el gancho, e impulso. Estos estados recogen el funcionamiento de estas mecánicas.

Escenas

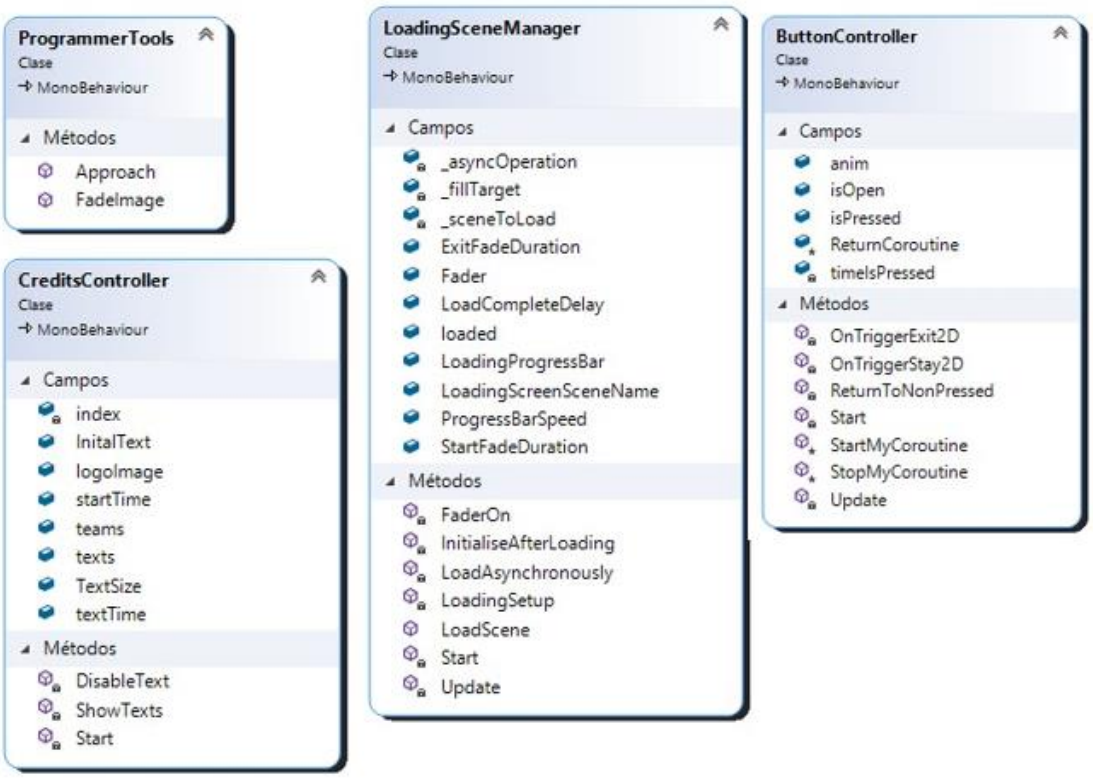


Ilustración 71: Diagrama de clases. Escenas

El trabajo realizado fue ButtonController que contiene el funcionamiento del botón para abrir la puerta de cada nivel.

Interacción Personaje

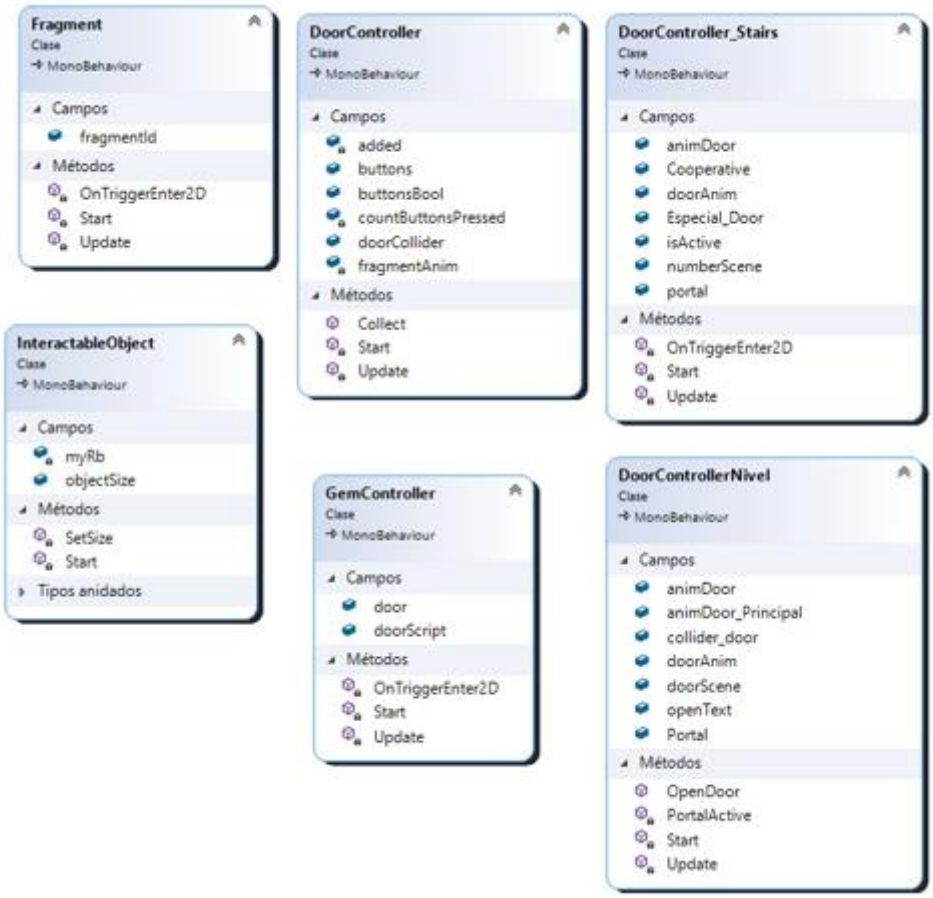


Ilustración 72: Diagrama de clases. Interacción Personaje

El trabajo realizado fue DoorController_Stairs que controla las puertas en la escena de las escaleras del juego, DoorControllerNivel que controla cuando se coge un fragmento para abrir la puerta. Por último, el DoorController que contiene el funcionamiento de los botones para que aparezca el fragmento.

Menú y selección

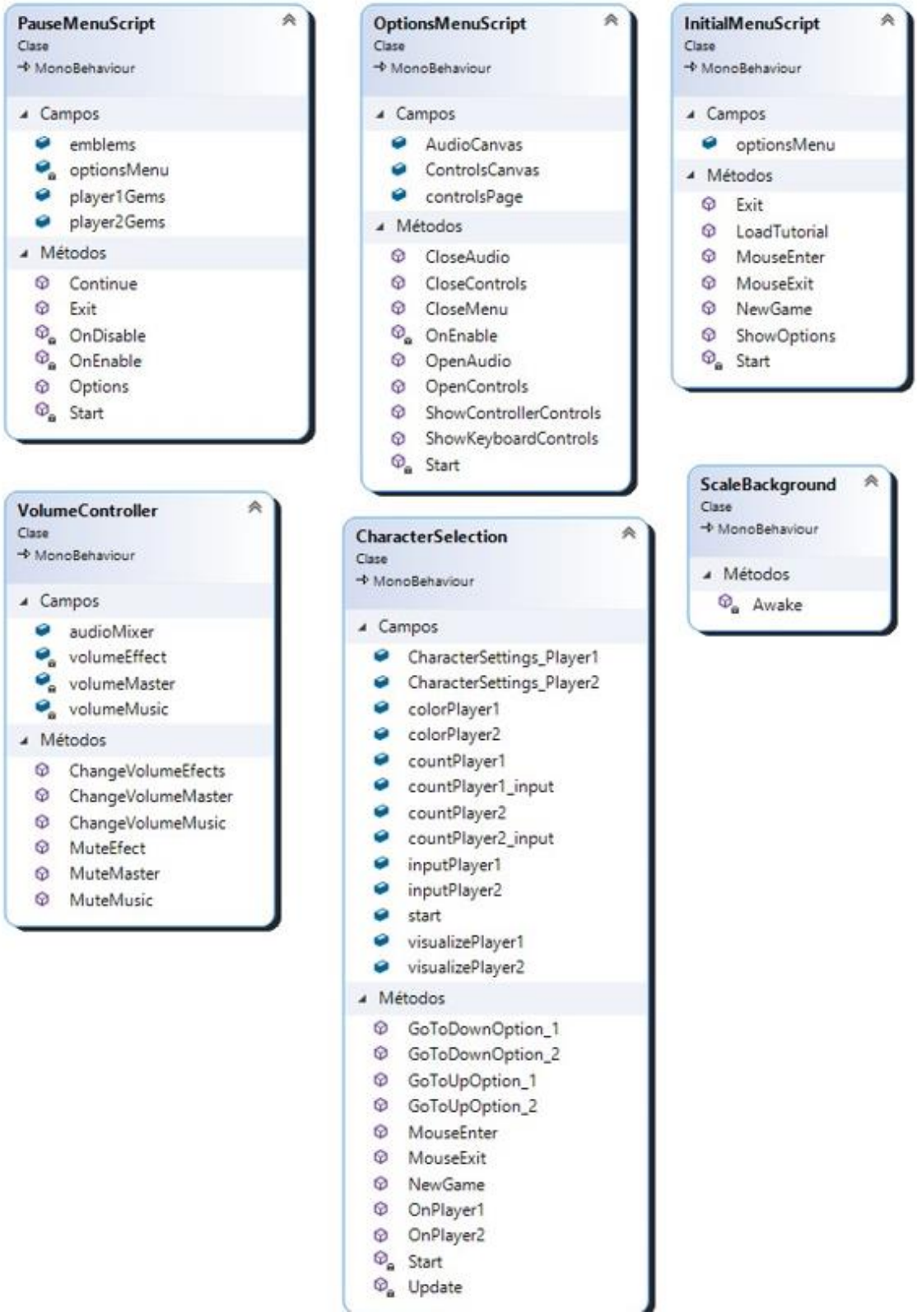


Ilustración 73: Diagrama de clases. Menú y selección

El trabajo realizado fue Character Selection que tiene todo el funcionamiento del menú de personalización del personaje, para que los jugadores puedan elegir el color y el periférico con el que jugar.

Skills del Personaje

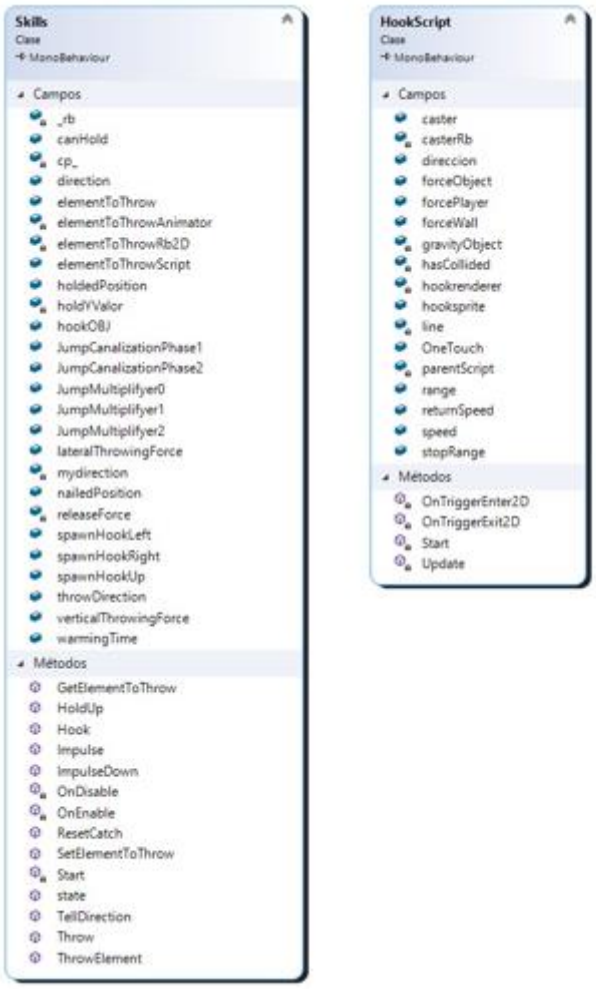


Ilustración 74: Diagrama de componentes. Skills

El trabajo realizado fue HookScript que contiene el funcionamiento de la habilidad gancho y Skills que contiene el impulso.

Trampas que no heredan

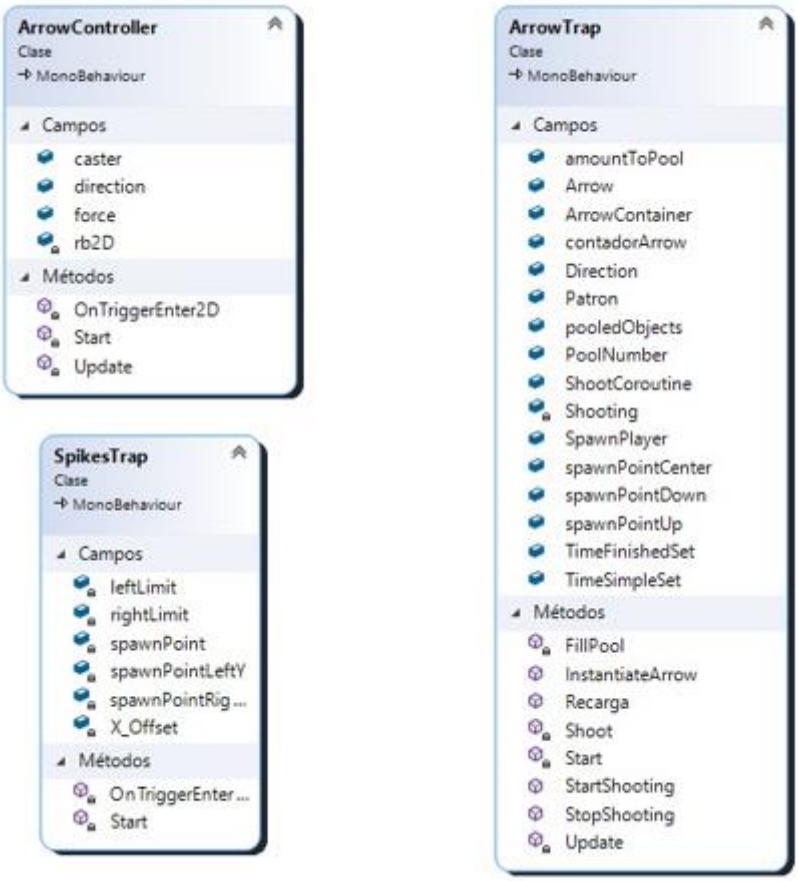


Ilustración 75. Diagrama de clases. Trampas que heredan

El trabajo realizado fue Arrow Trap que contiene el funcionamiento de la trampa de flechas que funciona con una pool de objetos. Y Arrow Controller, que controla el comportamiento de las flechas que salen de la trampa.

Trampas

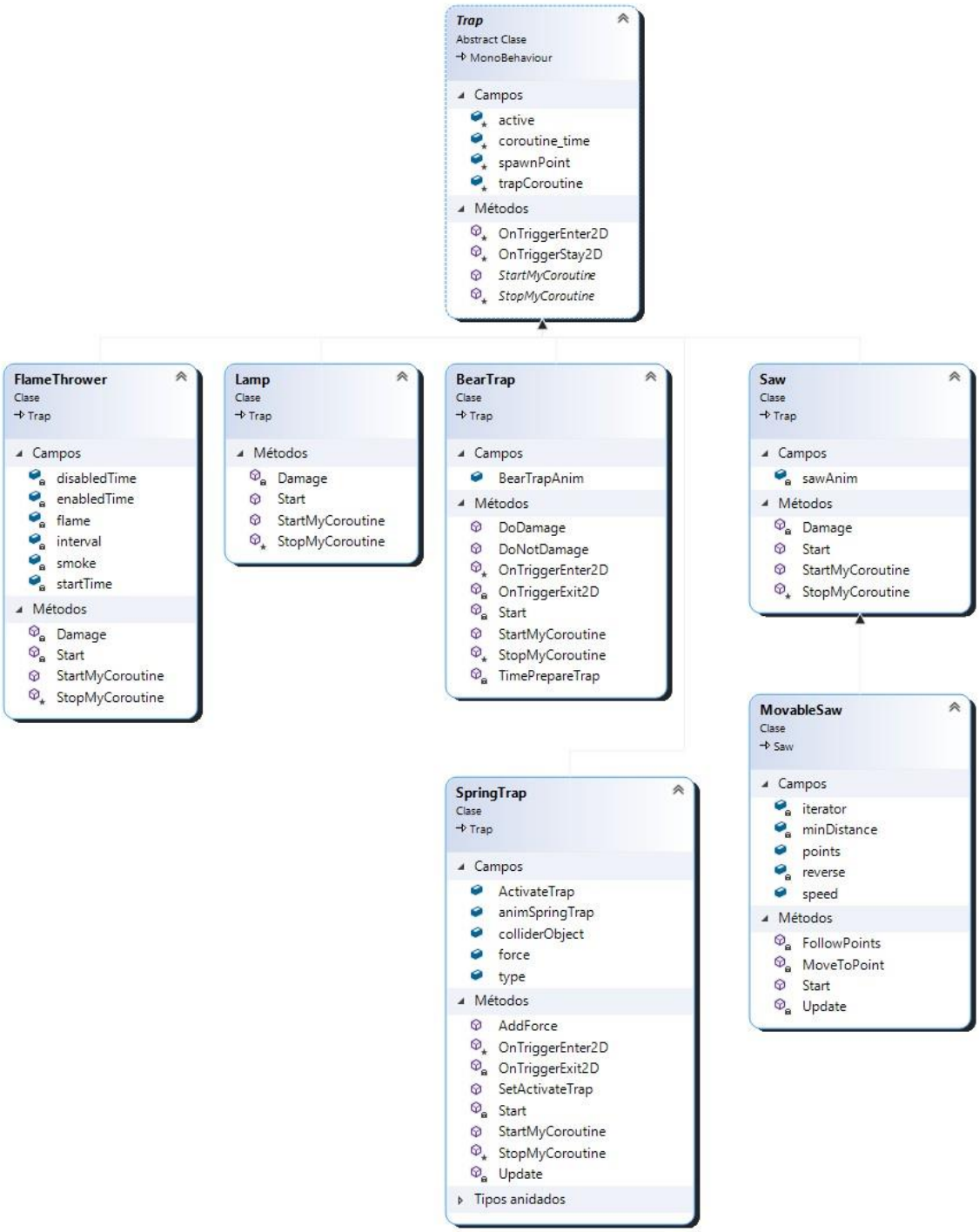


Ilustración 76: Diagrama de clases. Trampas

El trabajo realizado fue Spring Trap que contiene el funcionamiento de la trampa muelle, trampa que funciona por corrutinas y eventos de animación, y la Bear Trap que contiene el comportamiento de la trampa para osos.

Estructura del Proyecto

- Anima2D: Librería de terceros para animar por huesos.
- TextMeshPro: Librería de terceros para mejorar la edición y personalización de los textos.
- Animations: Recoge todas las animaciones del juego. Dentro cada elemento tiene su propia carpeta (Ej: Logo, HUD, Player)
- Font/Tipografías: Almacena las tipografías usadas en el proyecto.
- Gizmos: Recursos visuales.
- Scenes: Contiene las distintas escenas del proyecto.
- Sound: Contiene los recursos usados para la implementación del audio y sonido.
- Input: Carpeta que contiene los archivos relacionados con el nuevo Input System tales como el Input Actions (contiene todas las acciones del Player) y el script de Virtual Keyboard que sirve para que dos jugadores usen un mismo Keyboard, de esta manera, se soluciona el hecho de que dos jugadores estén pulsando al mismo tiempo.
- Materials: Materiales que se utilizan dentro del juego.
- Particles: Sistemas de partículas que se utilizan dentro del juego.
- Prefabs: Prefabs de los distintos elementos del juego.
- Scripts: Recoge los scripts que corresponden con cada elemento del juego, por ejemplo, la carpeta Player corresponde con los scripts del jugador.
- Shaders: Contiene los Shaders del juego.
- Sprites: Contiene los sprites que se utilizan como placeholder y los atlas que se utilizaran en el juego.
- Textures: Texturas del juego tales como los normal maps.
- Traps: Contiene lo relacionado con las trampas. En ella se encuentra el script Trap Group, que facilita el trabajo a diseño, así como la estructura de herencia de las trampas.
- URP: Contiene todos los archivos relacionados con el Universal Render Pipeline (Render usado en el juego).

BIBLIOGRAFÍA

[0] Brackeys. (2020, 5 julio). *EVERY Image Effect in Unity Explained - Post Processing v2 Tutorial* [Vídeo]. YouTube. https://www.youtube.com/watch?v=9tjYz6Ab0oc&ab_channel=Brackeys

[1] Channel Mixer. (2020). docs.unity3d.com. https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/Post-Processing-Channel-Mixer.html

[2] Code with k (7 de abril de 2020). Unity Post Processing with Universal Render Pipeline. Recuperado de https://medium.com/@yousafzai.kamran60/unity-post-processing-with-universal-render-pipeline-8782abd3f619#:~:text=Post%2Dprocessing%20applies%20full%2Dscreen,the%20image%20appears%20on%20screen

[3] Cooper, T (2018). The Lightweight Render Pipeline: Optimizing Real Time Performance. Unity Blog. Recuperado de: https://blogs.unity3d.com/es/2018/02/21/the-lightweight-render-pipeline-optimizing-real-time-performance/

[4] Corral Franco, P. (2020). *Desarrollo de un Videojuego Serio como herramienta de ayuda a la captación de población femenina en las carreras STEM* (Tesis de Fin de Máster). Universidad de Málaga, Málaga.

[5] Daitli. (2020, 5 marzo). *Overwatch Chain Hook Ability - Unity3D Tutorial* [Vídeo]. YouTube. https://www.youtube.com/watch?v=KBK8OsJa91Y&ab_channel=Daitli

[6] davidsanh. (2020, febrero). *OverwatchChainHook*. GitHub. https://github.com/davidsanh/OverwatchChainHook

[7] D.C. (2019, 5 marzo). *¿Qué es la aberración cromática y por qué se utiliza en videojuegos?* Generación XBOX. https://generacionxbox.com/aberracion-cromatica-videojuegos/

[8] DocFX. (2020). *Bloom*. Unity. https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/post-processing-bloom.html

[9] DocFX. (2020, 3 agosto). *Universal Render Pipeline Asset*. Unity. https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.0/manual/universalrp-asset.html?q=global%20light%202d

[10] Esoteric Software. (2020, 9 marzo). *Unity Universal Render Pipeline support*. Spine. http://es.esotericsoftware.com/blog/Unity-Universal-Render-Pipeline-support

[11] Lee, K. (2020, 21 octubre). *New Unity Input System: Getting Started*. raywenderlich.com. https://www.raywenderlich.com/9671886-new-unity-input-system-getting-started

[12] *Lente anamórfica*. (2020, 2 mayo). Wikipedia. https://es.wikipedia.org/wiki/Lente_anam%C3%B3rfica

[13] Lira, F (2018). Maximizing Rendering Efficiency [Sesión de conferencia]. MOVING MOBILE GRAPHICS SIGGRAPH 2018. British Columbia, Vancouver, Canadá

[14] *Luminancia*. (2021, 23 abril). Wikipedia. https://es.wikipedia.org/wiki/Luminancia

[15] Neel Bedekar (15 de noviembre de 2019). Getting Started with Unity's Lightweight Render Pipeline (LWRP). Recuperado de: https://developer.oculus.com/blog/getting-started-with-unitys-lightweight-render-pipeline-lwrp/?locale=es_ES

[16] McGrail, A. (2020, 10 febrero). *Achieve beautiful, scalable, and performant graphics with the Universal Render Pipeline*. Unity Blog. https://blog.unity.com/technology/achieve-beautiful-scalable-and-performant-graphics-with-the-universal-render-pipeline

[17] Patel, S. (2016, 9 marzo). *Unity - Lighting Effect on 2D sprite in unity*. TheAppGuruz. http://www.theappguruz.com/blog/unity-lighting-effect-on-2d-sprite-in-unity

[18] Pennock, J. (2012, 19 marzo). *Unity Automatic Documentation Generation (An Editor Plugin)*. http://www.jacobpennock.com/Blog/. http://www.jacobpennock.com/Blog/unity-automatic-documentation-generation-an-editor-plugin/

[19] Quirós Pérez, A. (2020). *Diseño de Módulo audio reactivo para la representación gráfica de audio para el proyecto Soundcool* (Tesis de Fin de Máster). Universitat Politècnica de València, Valencia.

[20] Unity Technologies. (2021). *2D Animation rápida y eficiente*. Unity. https://unity.com/es/features/2danimation

[21] Unity Technologies. (2021). *Facilitando cámaras para películas y videojuegos*. Unity. https://unity.com/es/unity/features/editor/art-and-design/cinemachine#unity-emmys

[22] Unity Technologies (2020). Feature Comparison Table. *Unity*. Recuperado de: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@10.2/manual/universalrp-builtin-feature-comparison.html

[23] Unity Technologies. (2021). *Introduction to Lighting and Rendering*. Unity Learn. https://learn.unity.com/tutorial/introduction-to-lighting-and-rendering#5c7f8528edbc2a002053b532

[24] Unity Technologies. (2021, junio 14). *TextMeshPro*. Unity DOCUMENTATION. https://docs.unity3d.com/Manual/com.unity.textmeshpro.html

[25] Unity Technologies (2020). Shaders and Materials. *Unity*. Recuperado de: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/shaders-in-universalrp.html#:~:text=The%20Universal%20Render%20Pipeline%20uses,new%20set%20of%20standard%20Shaders

[26] Unity Technologies. (2021, 14 junio). *Standard Shader*. Unity DOCUMENTATION. https://docs.unity3d.com/Manual/shader-StandardShader.html

[27] Unity Technologies. (2019, 7 mayo). *Visión general del post-procesamiento*. Unity DOCUMENTATION. https://docs.unity3d.com/es/2018.4/Manual/PostProcessingOverview.html

[28] Wikipedia (Sin fecha). Tubería de renderizado. Wikipedia. https://es.wikipedia.org/wiki/Tuber%C3%ADa_de_renderizado

[29] Wikipedia (Sin fecha). Graphics Pipeline. Wikipedia. https://en.wikipedia.org/wiki/Graphics_pipeline

[30] Zafra, D. (2020). *¿QUÉ ES LA EXPOSICIÓN EN FOTOGRAFÍA? AJUSTES BÁSICOS DE EXPOSICIÓN*. CAPTURE THE ATLAS. https://capturetheatlas.com/es/que-es-la-exposicion-en-fotografia/#definicion

Índice alfabético

[1] Muestro de alta calidad.

[2] La exposición en fotografía puede explicarse de forma simple como la cantidad de luz que recibe tu cámara.

[3] Según la Wikipedia, la luminancia es la densidad superficial de intensidad luminosa en una dirección dada.

[4] Según la Wikipedia, la lente anamórfica se refiere en óptica a una lente que aumenta de manera diferente el eje vertical y el horizontal la imagen. En el ámbito cinematográfico hace referencia al objetivo de la cámara que dispone de un dispositivo óptico compuesto por un sistema de lentes cilíndricas y prismáticas que permiten comprimir (anamorfizan) las imágenes en el eje horizontal adaptándola al ancho del fotograma para luego proyectarlas en pantalla ancha mediante ópticas que las descomprimen (desanamorfizan), devolviéndole su anchura normal.

[5] Son las llamadas de dibujo que suelen consumir muchos recursos, y la API de gráficos realiza un trabajo significativo para cada llamada de dibujo, lo que provoca una sobrecarga de rendimiento en el lado de la CPU.

[6] Según Deusens, la Realidad Extendida (Extended Reality) es una combinación de las denominadas tecnologías inmersivas: VR (Realidad Virtual) + AR (Realidad Aumentada) + MR (Realidad Mixta). Es decir, la Realidad Extendida, o XR, engloba todos los entornos, reales y virtuales, representados por gráficos de ordenador o dispositivos móviles.

[7] Tipo de algoritmo de selección de visibilidad que no renderiza objetos que están demasiado lejos de la vista del Player.