CATNIP

Trabajo de Fin de Grado

Alumno
Daniel Diz García

Directores
Antonio Seoane Nolasco Rocío Mihura López

comunicación audiovisual+3d+interactiva

UNIVERSIDADE DA CORUÑA

Índice

Anexos

Introducción

El proyecto presentado se denomina “Catnip”. Consiste en el diseño y desarrollo de un prototipo funcional de un videojuego que combina los géneros de plataformas y aventuras con desarrollo en *scroll* lateral. Tiene una trama detectivesca protagonizada por gatos en tono de humor, contando con numerosas referencias y parodias a elementos del cine, los videojuegos y otros referentes de cultura popular.

El objetivo del Trabajo de Fin de Grado es el diseño del videojuego y el desarrollo del prototipo del videojuego Catnip. Al ser un proyecto realizado por un solo estudiante, sin experiencia previa en desarrollo práctico de videojuegos, el aprendizaje y la investigación en áreas multidisciplinares fueron tareas constantes durante todas las fases de la producción.

Catnip es un proyecto que empezó en 2016, como proyecto para la asignatura de Videojuegos, aunque su base se formó tanto dentro como fuera del ámbito del Grado. Fue especialmente importante el apoyo de otros estudiantes interesados en el desarrollo de videojuegos, así como de familiares y amigos del entorno personal, para recibir ideas y *feedback* valioso que ayudó en gran medida a definir la identidad del proyecto.

El Trabajo de Fin de Grado se compone de una demostración de los primeros minutos del juego, una memoria que cubre los aspectos más importantes del desarrollo del proyecto, y el Documento de Diseño de Juego sobre el que se empezó a diseñar el proyecto. También se adjuntan otros anexos que permiten visualizar con mayor detalle elementos relativos a la memoria.

La labor de preproducción fue muy importante para definir las metas concretas del proyecto, y establecer un itinerario que concretase el camino a seguir, teniendo presentes las limitaciones ya mencionadas en cuanto a capital humano, conocimientos, medios y tiempo disponibles. Es importante destacar que, aunque se partió de un Documento de Diseño de Juego elaborado en la asignatura de Videojuegos en el curso 2017-2018, en el Trabajo de Fin de Grado se revisó, adaptó y completó el documento de diseño y se desarrolló el prototipo.

1. El videojuego “Catnip”

1.1. Sinopsis

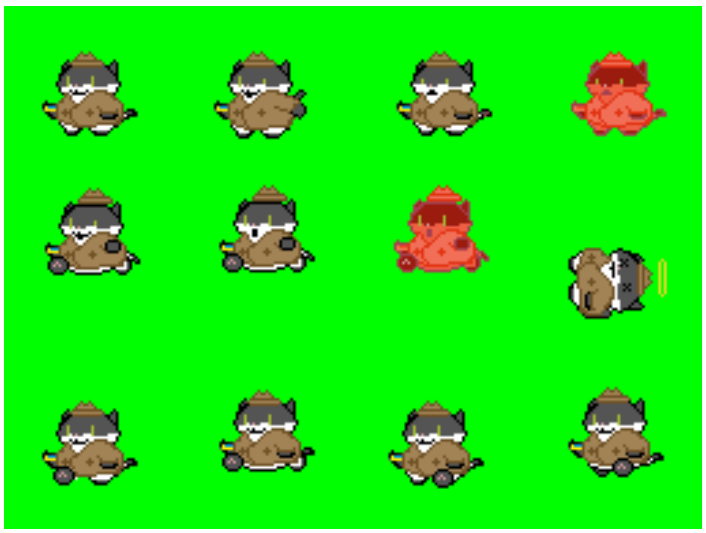
En Catnip, el jugador encarna a Neil Pawson, un gato detective de la ciudad de Michotown. Neil debe rescatar a su hermano Connor, que ha sido secuestrado por la organización criminal Gatuza que trafica con una sustancia llamada *catnip* que está tomando las calles. Para encontrar a su hermano y acabar con la Gatuza, Neil deberá seguir las pistas dejadas por Connor y embarcarse en un viaje lleno de peligrosos enemigos e inesperados aliados.

1.2. Descripción

Catnip es un videojuego que transcurre en la ciudad de Michotown, una gran urbe inspirada en Nueva York. Michotown está habitada por una sociedad de gatos antropomorfos y cuenta con elementos ambientales y estéticos que evocan a la sociedad estadounidense de la década de los 50. El contexto del juego es ficticio y sus lugares y eventos temporales no se corresponden con el mundo real, por lo que es habitual la presencia de elementos anacrónicos y tecnología imposible, tomando elementos de corrientes estéticas como el *retrofuturismo* o el *dieselpunk*. Por otra parte, existen paralelismos con figuras reconocibles de la cultura popular, especialmente de medios como los videojuegos, el cine o la televisión.

1.3. Orígenes

Catnip es una idea que surgió a finales de 2016 como proyecto personal dentro de la actividad de la asociación universitaria *Monkey Laboratory (MoLab).* Los encuentros de la asociación sirvieron para gestar la idea, que posteriormente sería desarrollada en la asignatura de Videojuegos durante el curso 2016-2017 y planteada en eventos como Carballo Interplay.



Spritesheet de Neil realizada en pixel art

La idea para el proyecto surge de elementos del día a día, al convivir con gatos y observarlos al pasar mucho disfrutando de su compañía. También estuvo influenciada por otras obras y referencias que se citarán más adelante, como el cómic *Blacksad* o películas como *Quien engañó a Roger Rabbit* o *Zootrópolis.*

El género de detectives también se pensó como un marco adecuado donde diseñar una historia que, a la hora de trasladar a un videojuego, reuniera elementos de exploración para motivar a conocer el mundo del juego, con sus entornos y personajes. Además, el género de detectives presenta unos arquetipos de personajes y situaciones habituales reconocibles por el público general, que se puede aprovechar para presentar bajo el estilo del proyecto. De este modo, se emplea un recurso de lugares comunes en los videojuegos, para facilitar la aceptación por parte de los jugadores.

1.4. Planteamiento

Desde el primer momento, el proyecto de Catnip se enfoca como un ejercicio para aplicar en la práctica la formación adquirida durante el grado, así como una oportunidad de aprender y adquirir competencias para una futura trayectoria en el sector del desarrollo de videojuegos.

Fue importante identificar pronto las limitaciones antes mencionadas y establecer hasta dónde se podía llegar, para plantear un proyecto realizable dentro de los plazos marcados, pero sin renunciar a que plasmase los valores y la identidad de la idea.

Uno de los mayores desafíos fue identificar y recopilar todos los elementos (*assets*) para construir el videojuego. Debido a la gran cantidad de *assets* necesarios, se tomó la decisión de elaborar unos elementos del juego y tomar otros de autoría externa.

La estética de Catnip es uno de sus elementos reconocibles. Al utilizar *assets* de varias fuentes, puede suceder que no todos tengan el mismo estilo. Esto no ocurre cuando los *assets* se producen de manera interna, porque se dispone de un mayor control creativo. Por otra parte, crear un *asset* nuevo supone más tiempo de producción que utilizar un *asset* ya preparado para utilizar. Los *assets* de terceros permiten agilizar la producción, pero también se ha invertido tiempo en su búsqueda. Fue necesario investigar cuales eran los *assets* más adecuados para el estilo del proyecto. Una vez seleccionados, se realizaron numerosas pruebas y ajustes para integrarlos dentro del juego.

1.5. Objetivos

Además del valor didáctico como ejercicio de Trabajo de Fin de Grado, con el proyecto Catnip se aspira a crear un videojuego de jugabilidad sencilla, que toma inspiración de numerosas obras de distintos medios, pero que quiere aportar un estilo propio. Para alcanzar este objetivo, el proyecto se sustenta sobre un aspecto gráfico atractivo, un estilo

y diseños identificables, y una sensación de juego que divierta al jugador y lo invite a continuar avanzando.

Se pretende conseguir crear un videojuego combinando *assets* internos y externos, con un método de desarrollo accesible pero efectivo, y priorizando obtener una estética y estilos reconocibles.

2. Preproducción

2.1 Planificación del trabajo

Para realizar la planificación, uno de los puntos de partida fue revisar el Documento de Diseño del Juego elaborado en la asignatura de Videojuegos para estructurar el desarrollo del proyecto.

En el Anexo IV que contiene el Documento de Diseño de Juego, Catnip estaba planteado como un juego completo diseñado para un circuito comercial. Esto significa que su complejidad era mayor de lo alcanzable para un único estudiante, con conocimientos básicos para afrontar el desarrollo, por lo que el Documento de Diseño de Juego sirvió para identificar los puntos fuertes del proyecto, simplificarlos y condensarlos en una demostración.

Una vez se decidió aquella parte que se iba a desarrollar, se descompuso el trabajo en listas de elementos, donde se determinaría el orden de desarrollo y las etapas donde se englobarían dichos elementos y el tiempo dedicado a cada parte. De este modo, pudieron elaborarse recursos a la producción como el Calendario de Producción y el Diagrama de Gantt.
Para controlar el progreso del proyecto, se utilizaron aplicaciones como Trello.

El proceso de planificación del trabajo se realizó durante el primer cuatrimestre. En el segundo cuatrimestre sobrevinieron circunstancias que afectaron notablemente al desarrollo del proyecto y alteró la planificación. Un periodo de enfermedad a mediados de enero paralizó el proyecto durante varias semanas. Del mismo modo, empecé a trabajar a media jornada desde abril, lo que se unió a dificultades para desarrollar el proyecto durante el inicio del confinamiento, ralentizando el desarrollo general respecto al primer cuatrimestre.

Como consecuencia, al haber planificado el trabajo pensando en una disponibilidad exclusiva para el proyecto, el tiempo dedicado a cada tarea tuvo que ser replanteado. Muchos elementos no esenciales de la idea inicial se simplificaron o suprimieron. A pesar del esfuerzo realizado, no se alcanzó el objetivo inicial de presentar en la primera convocatoria. Gracias al trabajo conjunto con los tutores, el proyecto pudo ajustarse a la nueva situación, reestructurando la planificación marcada en el calendario de producción y en el diagrama de Gantt.

2.1.1. Calendario de Producción

El calendario de producción permite ver el tiempo dedicado a cada tarea situado dentro de un calendario natural. Se han excluido los fines de semana.

El código de colores responde a distintas tareas, pero en aquellas que comprenden diferentes procesos se emplea un mismo color, especificando al inicio cada proceso y a qué parte corresponde. Por ejemplo, las tareas de diseño de personajes, modelado, texturizado y animación de personajes utilizan el color amarillo, ya que forman parte

de una misma categoría de “Personajes”. Del mismo modo, las tareas de color naranja engloban lo relacionado a la construcción del nivel, y las granate, a la programación.

Para ver el calendario de producción, consultar el Anexo I.

2.1.2. Diagrama de Gantt

El Diagrama de Gantt representa el contenido del calendario de producción, pero presentado de un modo que permite visualizar mejor la duración y la consecución de las tareas.

Para ver el diagrama de Gantt, consultar el Anexo II.

2.2. Investigación y referencias

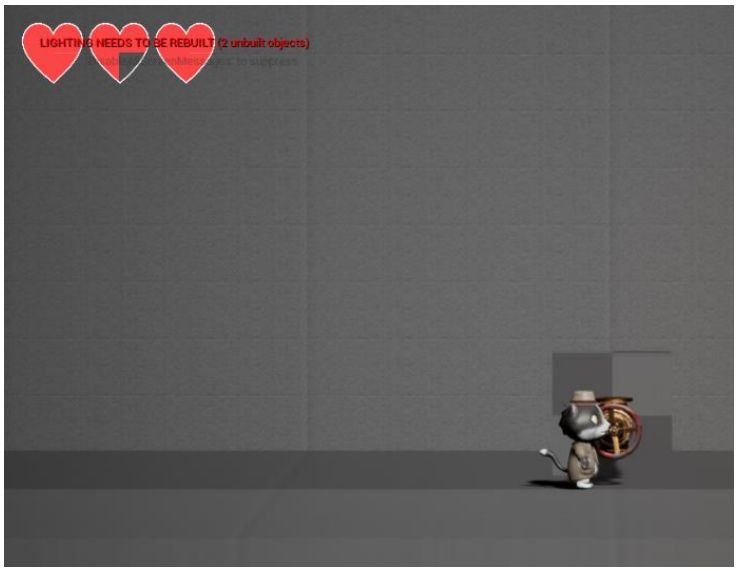
Durante el proceso de investigación se buscaron obras similares a *Catnip*, para obtener inspiración, identificar competidores y analizar la manera en la que se abordaron aspectos de la producción.

Al investigar se realizaba una labor de análisis activa, tomando nota de aspectos como los diseños de los personajes y entornos, la estructura de los niveles, la iluminación, la presentación de interfaces de usuario o los ritmos de juego.

La investigación también estuvo relacionada con la selección de *assets* a utilizar en el proyecto. Se buscó entre las librerías de *assets* 3D disponibles de plataformas como *Unreal Marketplace, Gamedev.tv, Free3D, UE4Resources, CGTrader, Free3D, Sketchfab o Turbosquid.* Otros recursos como imágenes, música, sonidos o iconos se obtuvieron de portales como *Freepik, Flaticon, Freiconshop* o *Taketones*. Se le dio preferencia a los recursos gratuitos o ya obtenidos con anterioridad al inicio del proyecto.

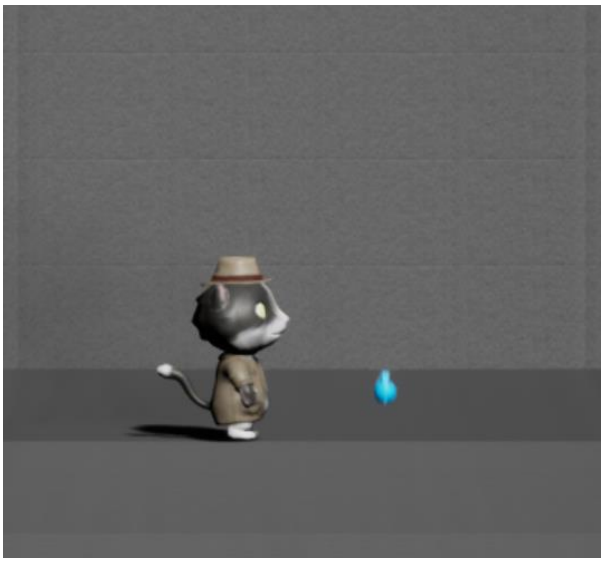
Los recursos descargados se integraban en niveles de prueba en un proyecto de Unreal Engine distinto, que actuaba como contenedor de assets. En ese proyecto se comprobaba como afectaba la iluminación, la escala del personaje o el rendimiento. Cuando se seleccionaban los *assets* a utilizar, se seleccionaban y se migraban al proyecto principal. De este modo, se evitaba saturar el proyecto principal con elementos innecesarios.

En ocasiones, la investigación llevaba a ideas que se trataron de implementar, pero que acabaron siendo desechadas o descartadas por causas diversas. Por ejemplo, se empezó a trabajar en un sistema de salud representado por corazones, que al recibir daño se iban vaciando a partir de cuartos, de manera similar a los videojuegos de la serie *Zelda.* Aunque se llegó a implementar y funcionaba a la hora de recibir daño, daba problemas al recuperar salud, por lo que, aunque ocupó varias horas de trabajo, se acabó descartando.



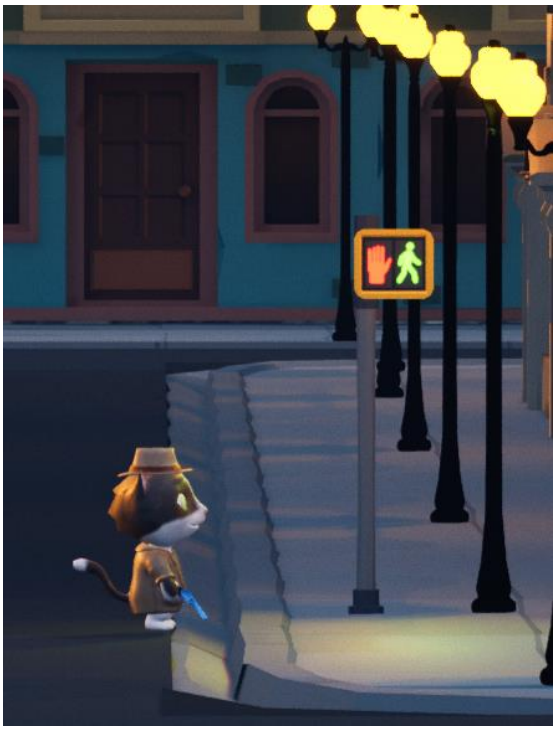
Sistema de salud defectuoso

Otro elemento descartado fue una mecánica de lanzamiento de granadas, pensado para dañar a los enemigos situados detrás de obstáculos. La mecánica consistía en que, al mantener pulsado el botón de lanzamiento de granada, se aumentaba la distancia a la que lanzaría en arco la granada. La granada explotaría tras dos segundos, causando daño en un área a los enemigos que se encontrasen dentro, y al jugador si era alcanzado. Se desechó porque el sistema utilizado funcionaba cuando se realizó en proyecto *FirstPersonShooter*, pero daba problemas al implementarse en un *SideScroller*, ya que las granadas no se impulsaban correctamente en un desarrollo 2D.



Neil lanzando una granada

Otras funcionalidades estaban previstas, pero no se llegaron a añadir, como integrar un *checkpoint* a mitad de nivel para reaparecer en caso de morir. Debido a la corta duración de la demostración del juego, se consideró una funcionalidad secundaria. Queda constancia en el nivel el lugar donde se iba a ubicar el checkpoint, y el elemento que lo representaría: un semáforo indicaría un peatón cruzando en verde, que se iluminaría al pasar el jugador por delante y que serviría el punto donde se reaparecería tras un *Game Over*.

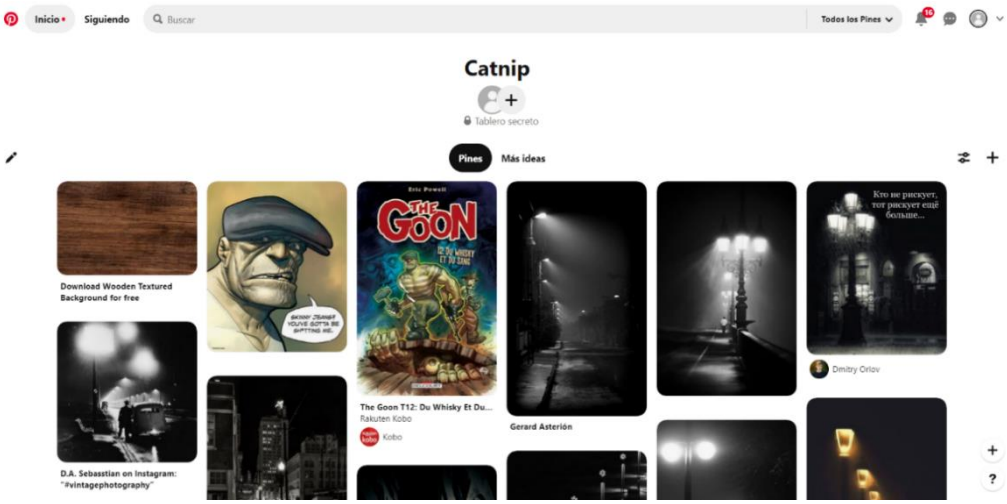


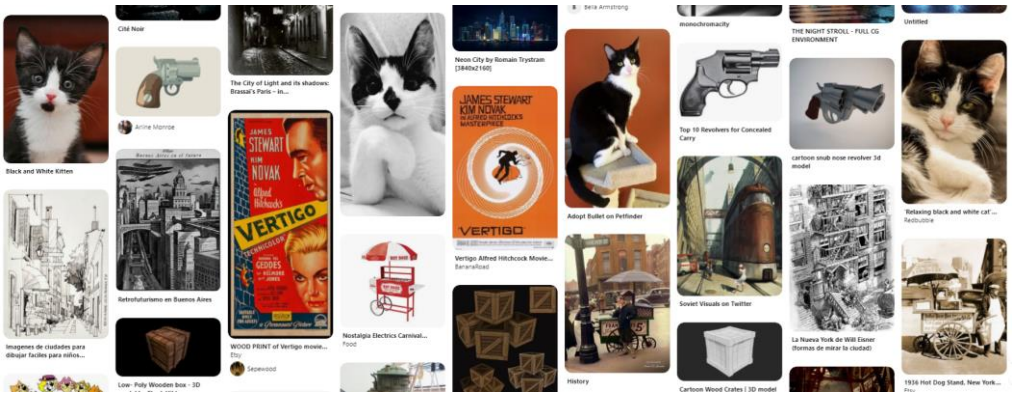
Representación de punto de guardado

Ya que definir un estilo característico era uno de los objetivos principales del proyecto, identificar y elegir las referencias adecuadas fue una tarea importante que ocupó gran parte del tiempo de la preproducción, y continuaron resultando de utilidad durante todo el proyecto.

Las referencias respondían a distintas necesidades del diseño de personajes, entornos, jugabilidad o ambientación. Mientras que unas ocasiones una misma referencia podía cubrir varios aspectos deseados, otras veces el valor de una referencia se encontraba en elementos muy concretos.

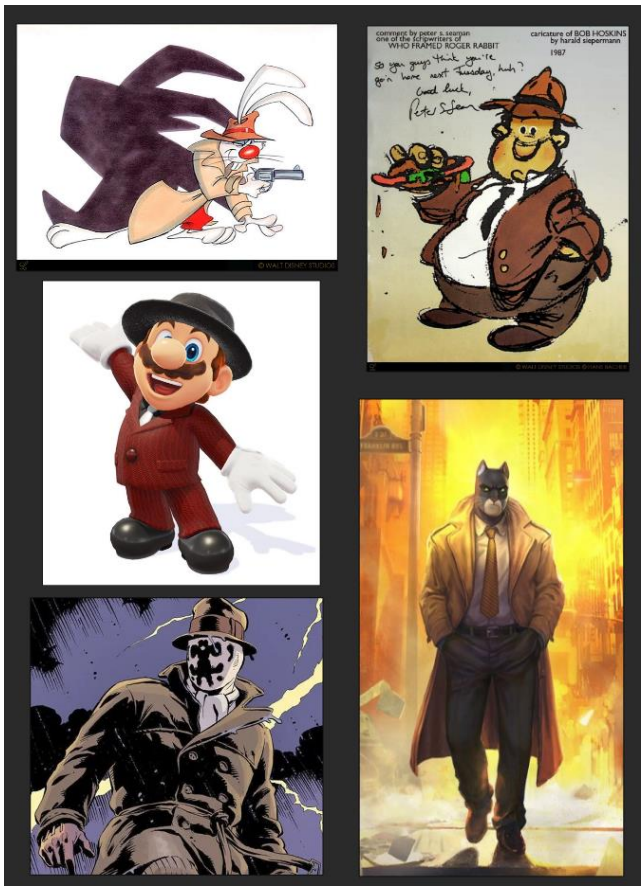
Para recopilar referencias se utilizaron buscadores de plataformas como *Google* y su herramienta *Google Image Search*, *Youtube*, o *Pinterest*. También fueron de gran utilidad las referencias aportadas por los tutores y por personas ajenas al proyecto.



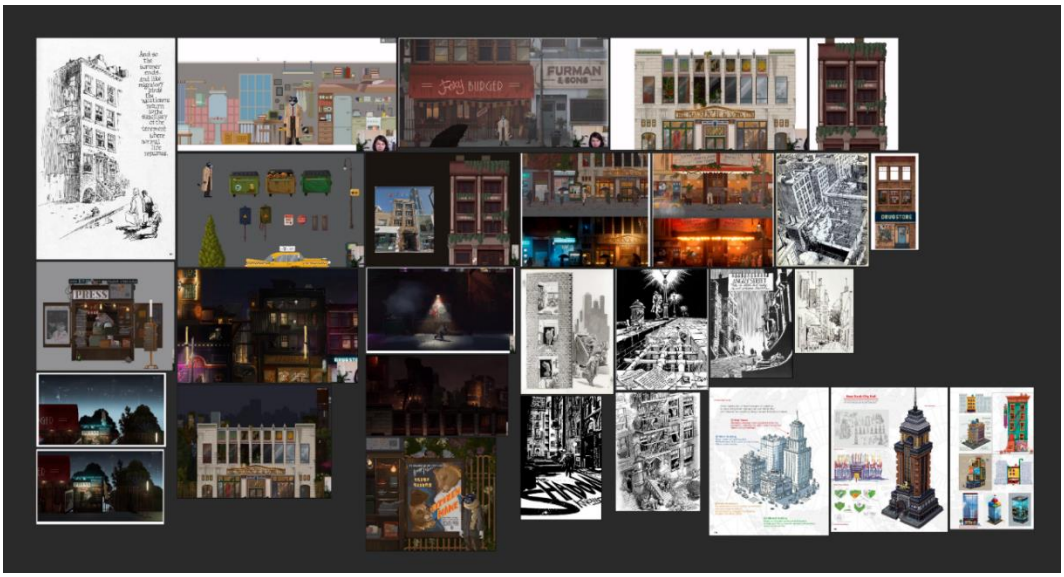


Tablón de Pinterest

Herramientas como *PureRef* permitieron componer las referencias en tablones, creando así visiones de conjunto. En base a las referencias también se pudieron crear "*moodboards*", de gran utilidad para definir aspectos como colores, proporciones o tonos.



Panel de FreeRef con referencias para el protagonista



Panel de FreeRef con referencias para la ciudad

Para diseñar la jugabilidad y el aspecto *ingame* de los elementos se tomaron referencias de varios juegos. Se procuró jugar directamente a los títulos para estudiarlos desde la perspectiva del jugador. Cuando no fue posible, se observaron vídeos de los juegos.

A continuación, se mencionan algunos de los juegos más influyentes para *Catnip:*

- *Guns, Gore and Cannoli (Rogueside – Claeysbrothers, 2015)*
  Videojuego que combina mecánicas de *shooter* y plataformas en *scroll* lateral, en un subgénero *Run and gun* que resultó una de las principales referencias del videojuego *Catnip.* Está ambientado a finales de los años 30 y presenta una construcción de escenarios en entornos urbanos, exteriores e interiores, poco habitual y que resultó de mucha utilidad para pensar en cómo crear los entornos del nivel de la demostración. También su estética resultó inspiradora a la hora de definir aspectos de personajes como los enemigos.
- *Detective Pikachu (Creatures, 2016)*
  De este videojuego se observó la manera en que se desarrolla el inicio del juego y las mecánicas de diálogo y recogida y análisis de pistas. Aunque estas mecánicas no se llegaron a introducir en el prototipo del videojuego Catnip, también se tomaron ideas como los toldos o las sombrillas sobre las que desplazarse, además de inspirar la parte del parque dentro del nivel.
- *Dex (Dreadlocks, 2015)*
  Este título combina elementos de muchos géneros, como aventura, plataformas, shooter o juego de rol, en un desarrollo lateral 2D y transcurriendo en un entorno urbano, por lo que sirvió de inspiración para diseñar el escenario y cómo moverse por él.
- *Serie Shantae (WayForward, 2002-2020)*
  Esta serie cuenta con numerosas entregas que presentan una fórmula similar., con una estructura por niveles explorables, pero contenidos, donde conseguir llaves u objetos que permitan avanzar de manera no lineal. Además, cuenta con varios personajes entrañables y un tono de humor presente durante toda la aventura, rompiendo la cuarta pared en ocasiones.
- *Sam & Max: The Devil's Playhouse (Telltale Games, 2010)*

- *Serié clásica de aventuras gráficas protagonizadas por una pareja de animales detectives que en su salto al 3D adoptó un estilo gráfico muy identificable.*
- *Blacksad: Under the Skin (Pendulo Studios, 2019)*
  Adaptación al videojuego de la serie de cómics Blacksad, una de las principales influencias de Catnip.
- *Donald Duck: Goin' Quackers (Ubisoft – Disney Interactive, 2000)*
  Uno de los niveles de este plataformas 3D transcurre en una ciudad, por lo que se tomó como inspiración.
- *Super Lucky's Tale (2017, Playful Corp.)*
  Los antagonistas de este juego son un grupo de gatos malvados con diseños muy variopintos.

2.3. Formación y aprendizaje

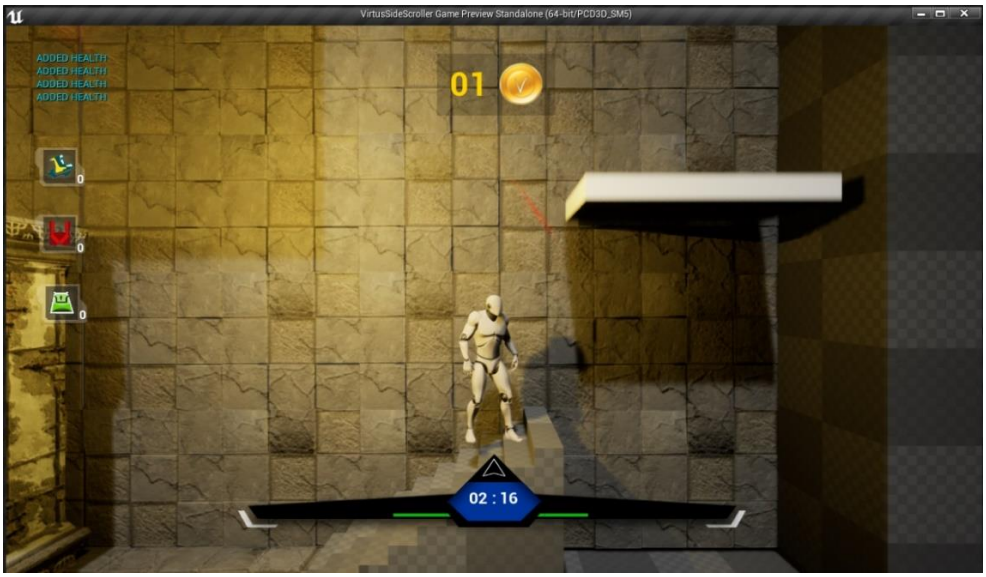
A la hora de elegir un videojuego como Trabajo de Fin de Grado, la formación adquirida durante las asignaturas del grado resultó de gran valor como introducción a las herramientas y conceptos necesarios para afrontar una producción así.

Sin embargo, la naturaleza del proyecto implicó adquirir conocimientos más allá de lo visto en clase, por lo que hubo que dedicar tiempo y esfuerzo a formarse mediante realización de proyectos previos, para conocer mejor las herramientas a utilizar y los procesos de trabajo para desarrollar videojuegos.

Sidescroller Virtus

En esta serie de vídeos se revisaron conceptos básicos de *Unreal Engine* durante el desarrollo de un proyecto de un videojuego de *scroll* lateral. Enseña a poner en marcha los sistemas de salud y munición integrados en un *HUD* funcional con gráficos, sistemas de menús, habilidades, *pickups* y elementos del entorno como obstáculos, plataformas móviles y peligros que pueden dañar al jugador.

El curso tiene un enfoque introductorio y mucho de su contenido ya se aprendió con anterioridad en la asignatura de Interacción 3D. Aun así, resultó de utilidad, ya que su formato de vídeos cortos y su propuesta de trabajar siempre dentro de un mismo nivel me permitió volver a recuperar las bases de Unreal Engine y aprender conceptos nuevos que se expandirían durante el proyecto.



Captura del proyecto Virtus Side Scroller

Creación de Videojuegos en Unreal Engine para principiantes con Horacio Meza

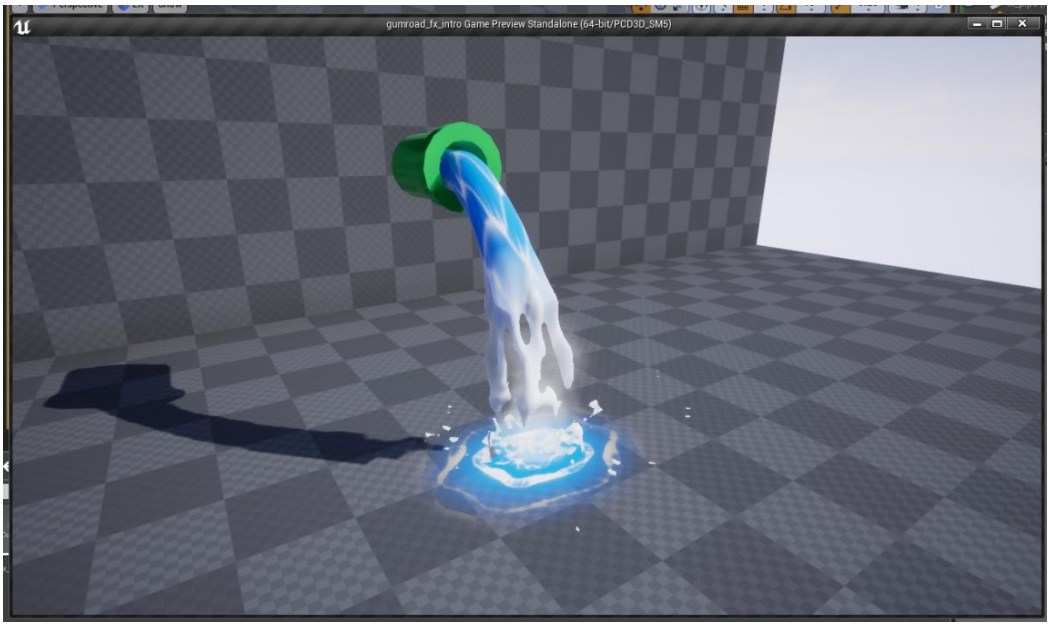
En este curso, recomendado por los tutores, se aprende a crear videojuegos sencillos a la vez que se introducen los fundamentos de Unreal Engine, intercalando lecciones de teoría con ejercicios prácticos. Resultó muy valioso para aprender más sobre el motor y comprender mejor sobre el funcionamiento de la programación en Blueprints, al explicar algunos de sus elementos más importantes, como los sistemas de eventos o el uso de variables, funciones y macros, de una manera clara y comprensible para desarrolladores sin conocimientos de programación.

Captura de un minijuego "Endless Runner" creado en el curso de Horacio Meza

Introduction to Visual Effects for Games in Unreal

Este curso de la plataforma *The Gnomon Workshop* está impartido por Jeremy Griffith, artista *VFX senior*. El curso parte desde la etapa de esbozo del efecto a realizar hasta su fase final dentro de Unreal Engine y sirve para introducirse a la creación de efectos para videojuegos. Además de impartir conocimiento técnico específico, incide en la observación y estudio de la naturaleza y de los fenómenos

físicos para comprender cómo funciona en el mundo real aquello que se quiere recrear.
Se entrará en detalle sobre el contenido del curso cuando se explique el proceso de creación del efecto del chorro de agua en el Anexo III.



Chorro de agua funcionando dentro del proyecto donde se desarrolló

BP 3rd Person Game: Introduction
Serie de vídeos del canal oficial de Unreal Engine impartidos por Wes Bunn, *Technical Writer* en *Epic Games.* Introduce elementos importantes para la creación de un juego en tercera persona, como la configuración de *inputs* (incluyendo el soporte de control para mando), conceptos de animación importantes como el uso y configuración de *Blendspaces*, *Animation Blueprints* y *State Machines*, *Skeleton Retargeting*, *Slot Nodes* y *Animation Notifies*. Todos estos puntos y más se aplicaron en la animación del personaje principal y en la de los enemigos, como se explicará más adelante en el punto correspondiente.

Foros de Unreal
El portal de los foros de la comunidad de usuarios y desarrolladores de Unreal Engine también fue un sitio web recurrente para resolver dudas, problemas y obtener inspiración para implementar ideas.

2.4 *Concept art* y *mockups*

Se realizaron *mockups* rápidos utilizando *collages* para diseñar elementos como la interfaz, los escenarios o los niveles. Al igual que los *moodboards*, fueron útiles para representar rápidamente ideas y transmitir conceptos de manera visual.

Concept art para representar cómo atacaría Neil dentro del juego

Composición realizada a partir de concept art y capturas de un nivel de demostración en Unreal Engine

3. Producción

En este proyecto de Trabajo de Fin de Grado consideramos que la etapa de producción se inicia en el momento en el que ya se ha concretado el trabajo a realizar y se ha definido un estilo al que atenerse.

3.1 Personajes

Los personajes son uno de los pilares sobre los que se sustenta el estilo de Catnip. Era muy importante que fuesen carismáticos y reconocibles por el público, despertando simpatía y que, con solo verlos, invitasen al público a querer jugar al título. Aunque el género de los gatos antropomorfos se ha explotado en multitud de ocasiones (siendo el cómic *Blacksad* uno de los principales referentes del proyecto), siguen siendo aceptados por la audiencia, especialmente en obras de animación combinadas con humor.

3.1.1 Referencias

A la hora de crear los personajes para ser implementados en el videojuego, las referencias fueron muy útiles para iniciar todo el proceso de creación del personaje.

Las referencias fueron importantes para tratar la anatomía de los personajes con proporciones no realistas.

Era importante lograr que el diseño de los personajes transmitiese expresividad.

Las cabezas grandes permiten destacar las caras sobre el resto del cuerpo.

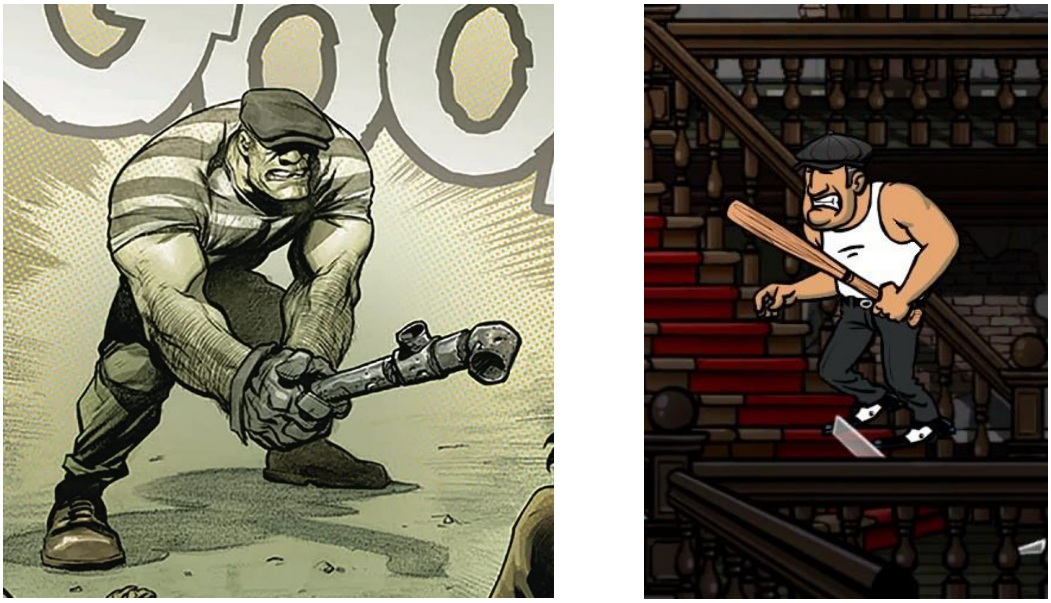
Las referencias de vestimenta sirvieron tanto para ambientar el mundo como para diseñar a los personajes.

Neil está inspirado en Blacksad, al ser un gato detective. Pero también toma elementos de Mario, como las proporciones. El patrón de manchas negras sobre pelo blanco está inspirado en el personaje de *Watchmen* Rorschach. La idea de que Neil fuera un detective bajito y regordete también toma inspiración del personaje de Eddie Valiant, de la película *¿Quién engaño a Roger Rabbit?*

La sociedad de gatos toma elementos de obras como *Fritz the Cat* o *Dick Tracy*.

Para los enemigos, una de las bases fue el protagonista del cómic *The Goon*, pero también se tomó inspiración de los enemigos de *Mafia 2* o *Tom & Jerry*.

Referencias de Neil, el protagonista

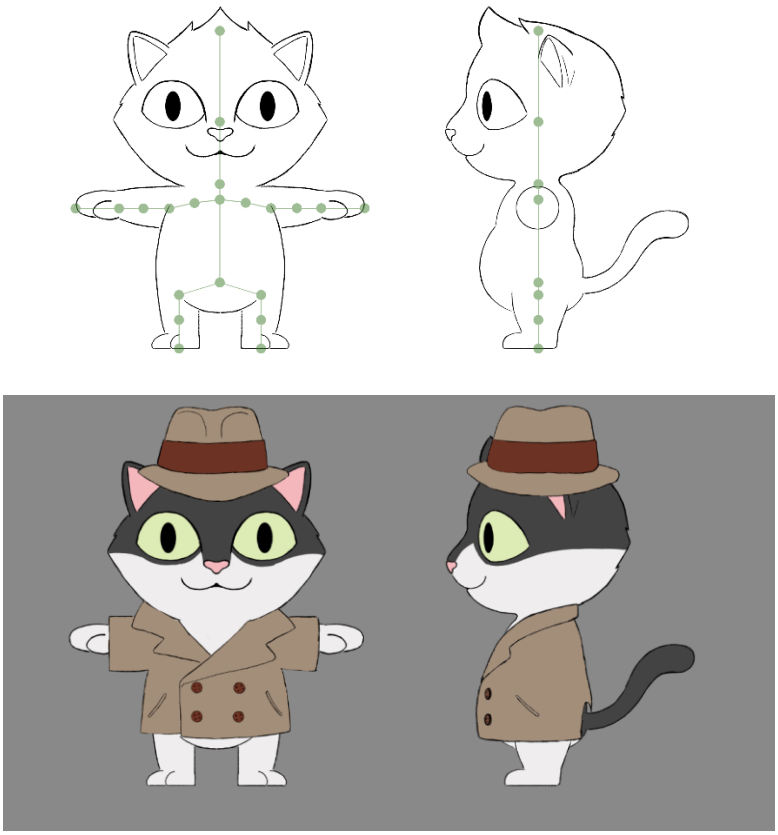


Referencias del enemigo matón

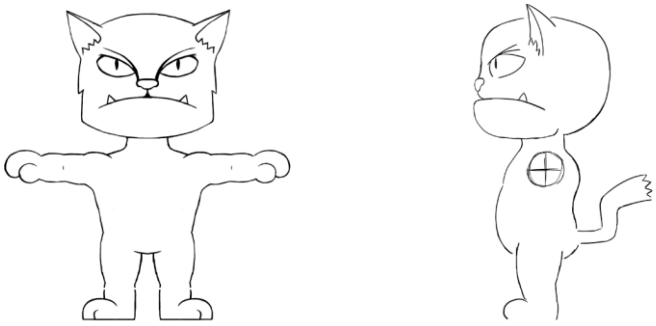
3.1.2 Modelado

Referencias de modelado

El arte conceptual sirvió para crear unas referencias de modelado que funcionasen, pues al no contar con una formación de dibujo artístico, errores en la perspectiva o las proporciones se traducirían en problemas en las siguientes etapas. En base a lo aprendido sobre cánones de proporciones en la asignatura de Animación 3D-1 y tras algunas pruebas en papel, se adoptó un canon de dos cabezas y media para el protagonista y de tres cabezas para los enemigos. Es habitual que el personaje protagonista sea más pequeño que la mayoría de sus enemigos, pues inspira una sensación de vulnerabilidad y desafío en los jugadores, al intimidarlos con el tamaño y la escala del mundo hostil que los rodea y por el que deben avanzar.



Referencias de modelado de Neil



Referencias de modelado del enemigo

El modelo de los personajes se realizó modelando la cabeza, el cuerpo y las extremidades por separado. Después se unieron y se ajustó la geometría en las zonas de las articulaciones donde se unían.

La ropa de los personajes se modeló a partir de las caras del cuerpo, eliminando la geometría del cuerpo cubierta por la ropa para ahorrar polígonos y optimizar los recursos. Sin embargo, esto resultó ser un problema en la etapa de animación, al no prever que podían quedar expuestas zonas sin geometría al deformarse con el *skin* del personaje.

Accesorios como el sombrero del personaje o la gorra de los enemigos se modelaron en escenas independientes y luego se añadieron al modelo de los

personajes, combinando todas las mallas en una sola para ser animada y exportada.

En el modelado se encontraron problemas que no se previeron en la fase de concepto, que llevaron a cambios en el diseño de los personajes.

La gabardina de Neil estaba cerrada en los diseños iniciales del personaje, pero el resultado del modelo 3D no funcionaba igual que los conceptos. Se optó por abrir la gabardina, pero al abrirla se veía un área vacía en la zona del cuello, el pecho y la barriga, por lo que se decidió añadir una corbata. Añadir una corbata presentaba otros problemas, ya que al ir suelta habría sido necesario configurar un tejido para que respondiera de manera dinámica a las animaciones del personaje. Para evitarlo, se siguió experimentando con los patrones de manchas del personaje protagonista y añadir la forma de la corbata como parte de su pelaje, lo cual causó una impresión positiva al mostrar el proyecto a terceros, por lo que se aceptó como la solución más eficaz.

El diseño de Neil y del pistolero enemigo eran similares, al contar los dos con un sombrero *fedora* y una pistola. Por otra parte, dentro del universo del videojuego, y como convención en los videojuegos en los que aparecen enemigos relacionados con el *hampa*, es que los enemigos visten mejor a medida que aumenta su cargo dentro de la organización criminal. Crear un nuevo diseño supondría volver a etapas ya cerradas y retrasaría el avance del proyecto. Para solucionar estos problemas, el pistolero pasó a compartir modelo con el matón, diferenciándose en los colores.

Diseño original del enemigo pistolero

Un principio de diseño que se siguió durante el desarrollo de los personajes fue que su silueta fuera distinguible para que se diferenciaran entre sí y con el entorno. Para probar la silueta, se utilizaban las vistas ortográficas de maya con la opción de visualización *Use All Lights,* que permitían ver una silueta negra en vista lateral del personaje y comprobar su contorno. En base a la silueta se modificaron partes del diseño durante la fase de modelado, como el pelo del protagonista o la joroba de los enemigos.

Silueta de Neil

Silueta del enemigo

3.1.3 Texturizado y materiales

Para el texturizado de los personajes se utilizó Photoshop, dibujando a mano sobre los mapas de UVs generados en PSD Networks desde Maya. Se utilizaron

pinceles personalizados para las distintas texturas, como el pelaje o la ropa de los enemigos, y se tomaron referencias para estudiar el pelaje de los gatos.

Textura del pelaje de Neil

Textura del pelaje del Enemigo

Textura de la gabardina de Neil

Textura del jersey del enemigo

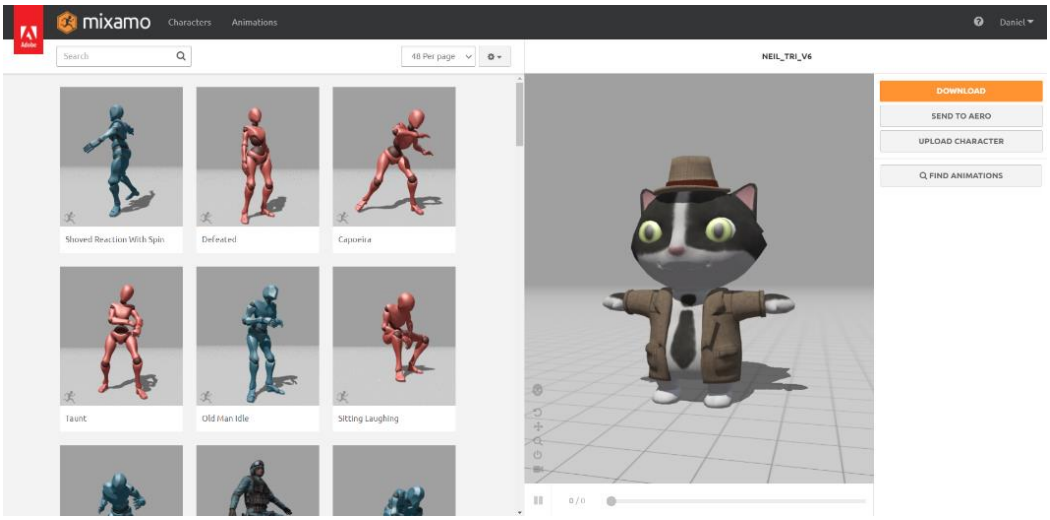
Para añadir el tinte rojo cuando los personajes reciben daño se utilizaron las mismas texturas, pero con variaciones en los materiales de Unreal.

Al añadir los modelos en FBX a Unreal se crearon automáticamente materiales a partir de las texturas asociadas, pero fue necesario ajustar sus valores para que presentaran la apariencia deseada.

3.1.4 Animación

Mixamo

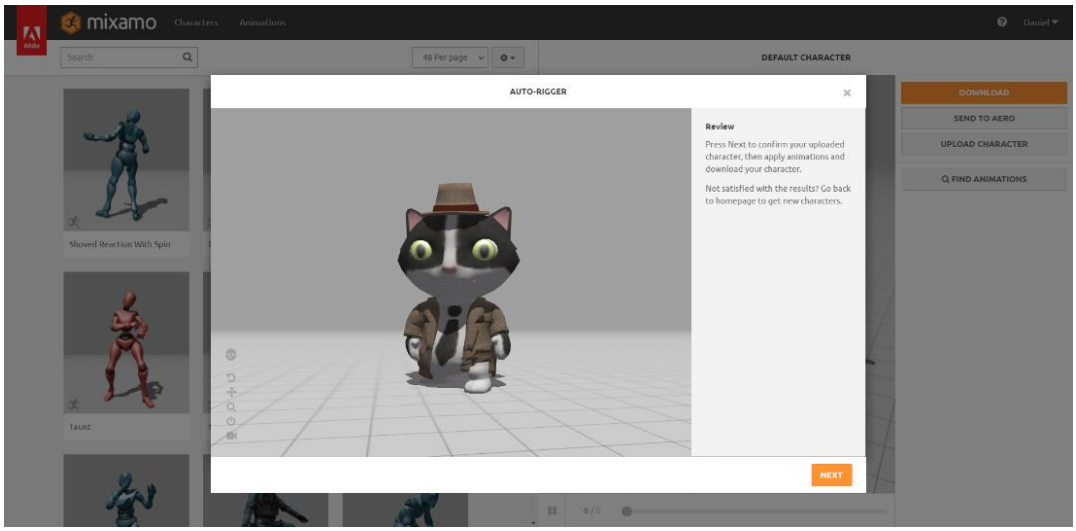
Para realizar las animaciones se utilizó el catálogo de animaciones de la plataforma *Mixamo*. Una vez modelado el personaje, se subió el modelo como un archivo FBX, y Mixamo le asignó un esqueleto utilizando su herramienta automática de *rig*.



El modelo de Neil cargado en Mixamo

Esta herramienta permitió ahorrar mucho tiempo en comparación a tener que realizar todo el proceso de *setup* del personaje (creando el esqueleto, skin y rig con sus controles), además de las animaciones correspondientes.

Sin embargo, hubo que invertir más tiempo del esperado en ajustar los pesos de los vértices del *skin* del modelo, ya que se deformaban incorrectamente en las animaciones, especialmente en las zonas de las articulaciones. Las animaciones de Mixamo están diseñadas para modelos humanoides con proporciones más realistas.

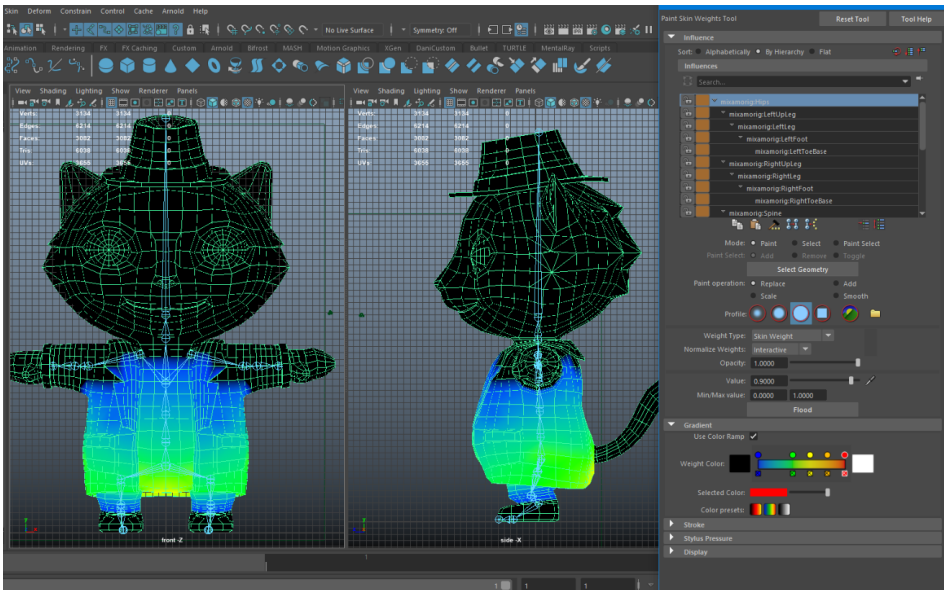


Problemas de deformaciones en Mixamo

Maya

Para corregir estos errores, se añadían las animaciones a la línea de tiempo de Maya, y se comprobaban fotograma a fotograma. En el proceso de corrección se trabajaba desde lo general a lo específico, siguiendo la jerarquía de los *joints* del esqueleto y bloqueando aquellos que quedaban resueltos.

Primero se utilizaban los pinceles para eliminar la influencia de los *joints* en zonas donde era totalmente incorrecta. Por ejemplo, una norma a seguir era que las articulaciones del lado derecho no podían ejercer ninguna influencia sobre las del lado izquierdo, y viceversa, por lo que se subsanaba tan pronto se detectaba.



Corrección de los pesos en Maya

A continuación, se pintaban las influencias de los *joints* con valores que fueran coherentes a la anatomía, pero también teniendo en cuenta el resultado en pantalla. Para las correcciones finales, se utilizaban herramientas como *Hammer*

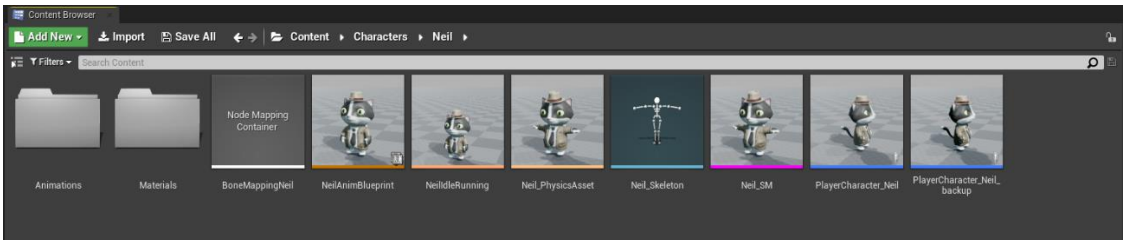
Skin Weight, que corregía vértices tomando los valores de los adyacentes. Era una manera eficiente de corregir vértices que despuntaban respecto a sus vecinos. Por último, aquellos vértices que presentaban errores que no se lograban subsanar en los procesos anteriores se revisaban y corregían desde el panel *Component Editor* de Maya.

Una vez se resolvieron todos los vértices de uno de los lados del modelo, se realizó un *mirror* de los valores de los pesos. De este modo, se ahorraba tiempo al no tener que duplicar todos los procesos, uno para cada lado, aunque se realizó una comprobación general final de las animaciones para verificar que la operación de *mirror* se había efectuado correctamente. La última operación de ajuste de los pesos antes de la exportación fue aplicar la herramienta *"Prune Small Weights",* para eliminar influencias residuales, ya que Unreal tiene problemas para reconocer modelos con valores de pesos inferiores a 0.015.

El personaje se exportó desde Maya a Unreal como archivo FBX, siguiendo las indicaciones de las clases de la asignatura de Interacción 3D. Se exportó al personaje en la T-Pose sin animaciones asociadas, mientras que las animaciones se exportaron una a una en diferentes rangos de la línea de tiempo de Maya.

Unreal

En Maya, se importó al personaje con la geometría en *T-Pose* como referencia principal desde donde se crearon los componentes principales del personaje: la *Static Mesh*, el *Skeleton*, el *Physics Assets* y el *Animation Blueprint*. Cada pose se importó asociándola al *Skeleton* del personaje.

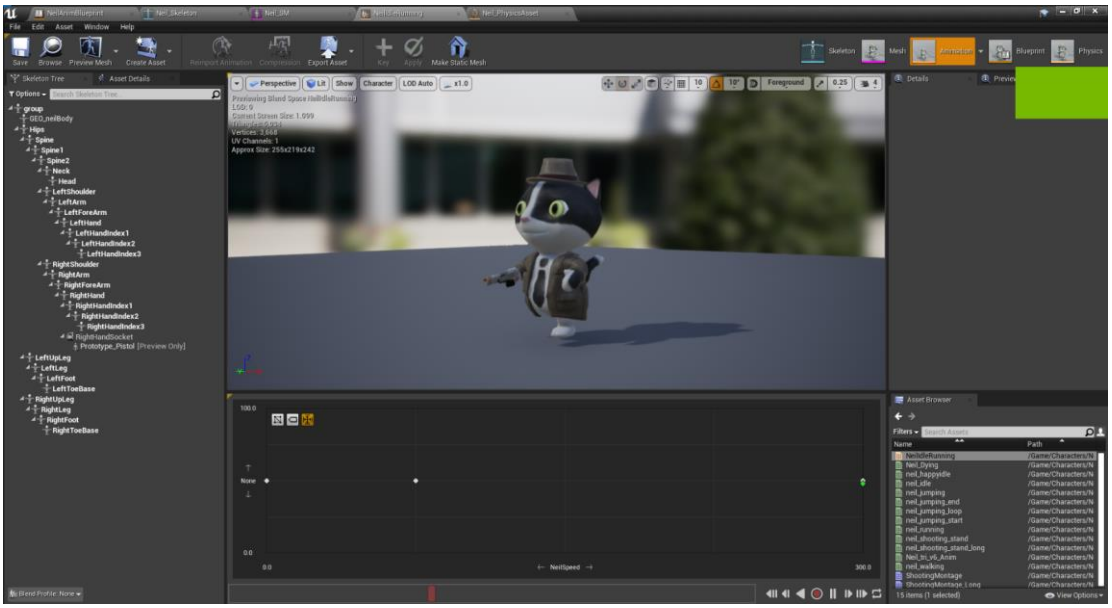


Personaje en Unreal y sus componentes

Una vez importadas las animaciones, se procedió a construir la programación del personaje. Una de las primeras tareas de programación de la animación fue la del desplazamiento del personaje.

Primero se construyó un *blendspace 1D,* donde se incluyeron tres animaciones: la animación *idle* o en reposo, que correspondería a la que ejecuta el personaje cuando no está realizando ninguna otra acción; la animación de caminar y la animación de correr. El *blendspace* permite realizar una transición de una a otra en función del valor del eje horizontal X, que correspondería al desplazamiento del personaje por las superficies del entorno. Aunque la construcción del *blendspace* es un proceso sencillo, requiere pruebas para encontrar los valores adecuados para implementar las transiciones entre animación, y también se

modificaron los valores originales de *playback speed* de las animaciones de caminar y de correr para evitar que los pies del personaje patinasen.

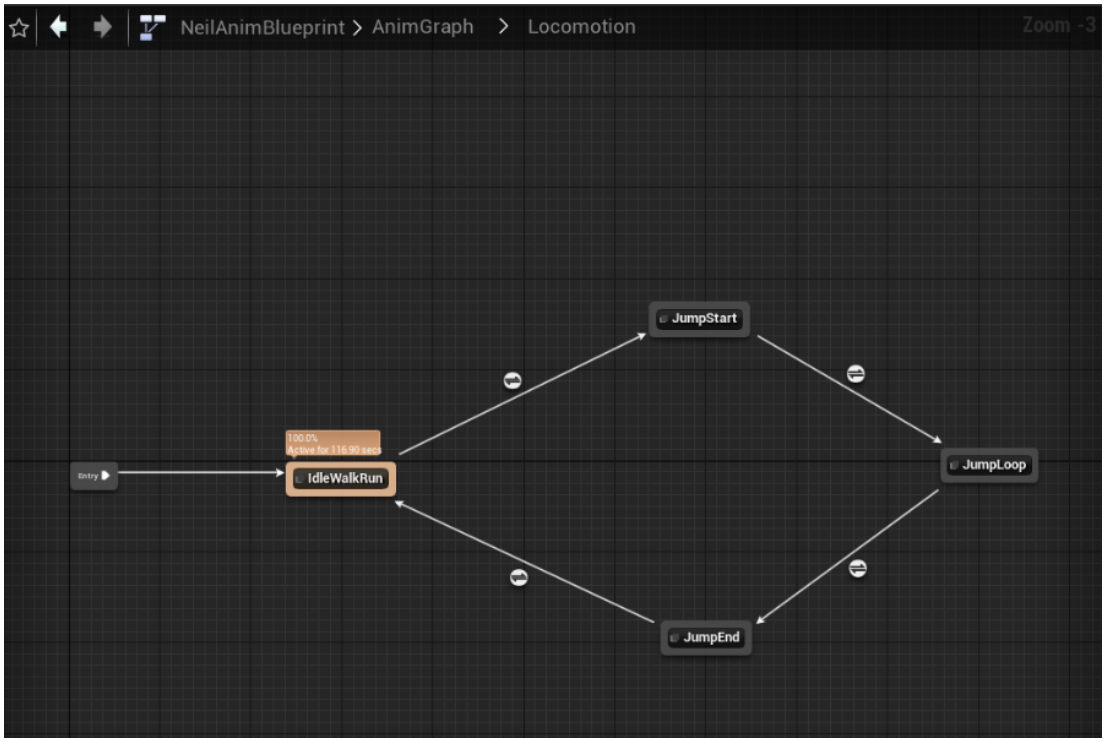


Creación del Blendspace para el movimiento de Neil

Una vez construido el *blendspace*, se procedió a trabajar con la animación de salto. La animación de salto que se había seleccionado en Mixamo presentaba un inconveniente, ya que no permitía ser ejecutada en bucle mientras el personaje estuviera en el aire, al estar pensada para ejecutarse con un inicio y un final con una distancia determinada. Para poder aprovechar la animación, se realizaron tres duplicados y se recortaron los fotogramas para descomponerla en tres animaciones diferentes: salto desde el suelo (*neil_jump_start),* suspensión en al aire (*neil_jumping_loop*) y aterrizaje (*neil_jumping_end*).

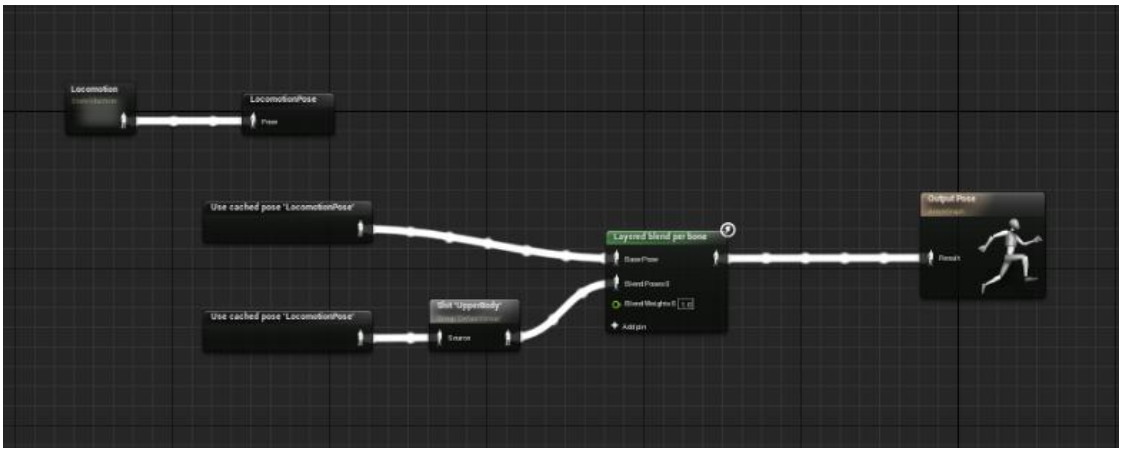
Fragmentación de la animación de salto

Con estas animaciones, se pudo construir un sistema de *Locomotion* que incorporaba un sistema de reglas que se regía por una variable booleana, que determinaba la animación a ejecutar según el estado del personaje: si estaba saltando, si estaba en el aire, o si estaba aterrizando. Con este sistema, el personaje ya no realiza la animación de caminar mientras se encuentra en el aire.



Sistema de Locomotion de Neil

Este sistema de *Locomotion* también sirvió como base para implementar que el personaje pudiera realizar la animación de disparar mientras corría y saltaba. Utilizando el nodo *Layered blend per bone* dentro del Animation Blueprint del personaje, es posible conectar dos poses a dicho nodo. Una de las poses actúa como una máscara que solo afecta a un subconjunto específico de huesos dentro del esqueleto. De este modo, se pudo tomar el *joint Spine1* como el punto del esqueleto donde se realizaría la transición de una pose a otra. Así, al ejecutar la animación de disparo, solamente se utilizaría la parte del esqueleto desde el *joint* Spine1 hacia arriba, permitiendo correr y saltar con la parte inferior del cuerpo.



Combinación de dos animaciones

3.1. Nivel

3.1.1. Descripción

El nivel entregado con el proyecto transcurre en las calles del barrio donde se encuentra el despacho de los hermanos detectives Connor y Neil Pawson. En la demostración se recreó la primera parte del primer nivel, que correspondería a Neil avanzando por las calles para llegar hasta el edificio donde se encuentra su hermano Connor.

Llegar hasta el despacho de los Pawson no será una tarea fácil. Las calles han sido tomadas por la Gatuza y los vecinos se han refugiado en sus casas. Los matones armados con porras y los pistoleros campan a sus anchas, atacando a cualquier que se cruce en su camino. Los secuaces de la Gatuza se encuentran por todas partes, desde las aceras hasta los tejados, por lo que Neil no estará a salvo en ninguna parte.

La invasión de los Gatuza afecta a la logística del pequeño comercio de Michotown

El barrio está sumido en el caos, con cajas abandonadas en las aceras y puestos de perritos calientes sin dueño que los vigile. La explosión de una tubería de gas ha provocado violentos incendios que cortan el paso a los viandantes. Para evitar que nadie entre en el edificio de los Pawson, la Gatuza ha saboteado un depósito

de agua que descarga directamente sobre el portal. Para poder cortar el agua Neil deberá encontrar y colocar la llave de paso en la azotea donde se encuentra el depósito.

Los tejados están vigilados por pistoleros de puntería certera

Para la estética del nivel se ha buscado la idea de edificio neoyorkino *vintage* de ladrillos y cemento, pero con fachadas coloridas en tonos pastel. Nos encontramos en un barrio humilde, de clase trabajadora. Los edificios tienen pocas alturas, y están construidos muy cercanos entre sí o adosados. Las uniones entre edificios crean callejones estrechos, ideales para ocultar la basura y sufrir emboscadas y atracos.

El parque es un pequeño refugio de vegetación en un entorno urbano y permite dividir el nivel en tres etapas. En el parque nos encontramos con un gran número de enemigos y no tenemos donde escondernos, por lo que puede ser aconsejable huir hacia las alturas de los balcones, utilizando el puesto de perritos calientes como trampolín y evitando así el fuego que corta el paso.
En los balcones hay que tener cuidado al saltar a los aparatos de aire acondicionado, ya que algunos están mal sujetos y pueden desprenderse al vacío, donde se encuentran llamas abrasadoras.

El parque está lleno de enemigos. Es recomendable huir..

Neil avanza entre los balcones

Una vez superados los balcones, Neil se encontrará con más gatos pistoleros en las azoteas. Deberá coordinar sus saltos con los disparos si quiere acabar con ellos con seguridad. También habrá momentos para trepidantes saltos entre azotea y azotea. Si Neil cae, no se hará daño, ya que los gatos siempre caen de pie, pero deberá volver a subir, pues en el tejado se encuentra la cañería que corta el paso del agua del depósito agujereado por la Gatuza. Una vez se cierra el agua, Neil puede pasar al portal de su edificio.

3.1.2. Referencias

La ciudad de Michotown se inspira tanto en ciudades reales como ficticias. La ciudad de Gotham donde transcurren las historias de Batman es una referencia, por la idea de ser un nido de maleantes. Para el barrio de los Pawson, se aspira a mostrar parte del ambiente de barrios como Brooklyn, tal y cómo aparece representada en los cómics de Will Eisner, con barrios humildes pero llenos de

gente. Este ambiente de historias de detectives en cómics en ciudades también se encuentra en clásicos como The Spirit, Sin City o Dick Tracy.

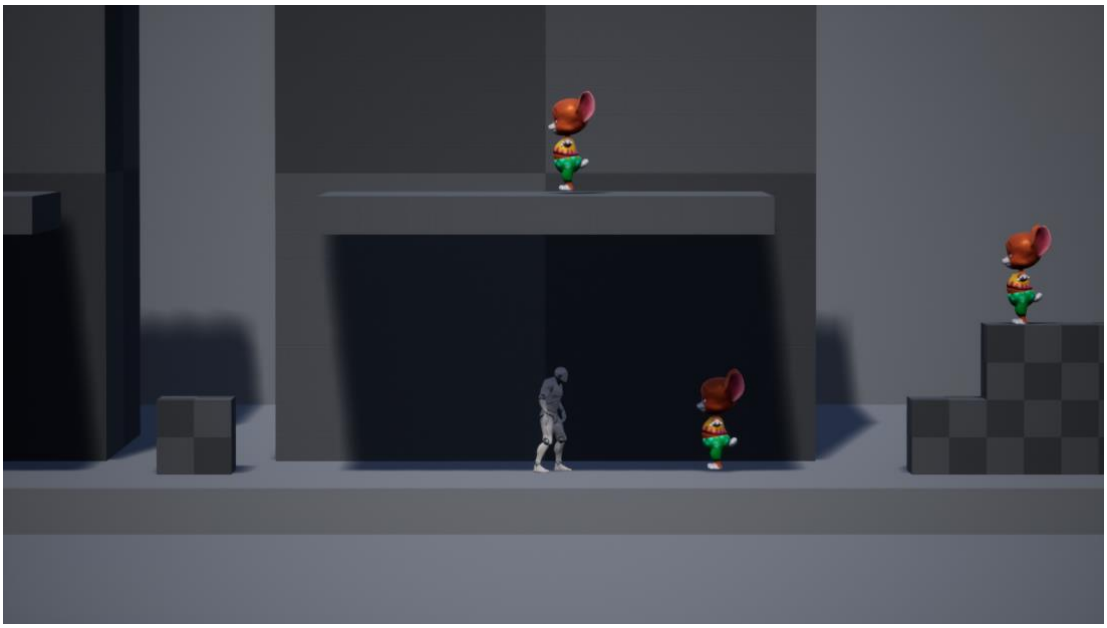
A la hora de estudiar referencias en videojuegos para diseñar el entorno, no se encontraron tantos ejemplos válidos de plataformas modernos en *scroll* lateral que transcurran en niveles urbanos. Los niveles urbanos presentan una serie de características propias, a la hora de pensar en su distribución, que los hace más complicados para distribuir elementos como las plataformas o las rutas por las que circular. En el caso de *Catnip*, se intentó que los elementos estuvieran colocados de una manera coherente, evitando elementos como plataformas o superficies flotantes, y manteniendo una escala de los elementos acorde a los personajes. Para recrear estos entornos, juegos como Dex o Detective Pikachu fueron buenas referencias. Otra referencia importante fue el nivel de *New Donk City* en *Super Mario Odyssey*, ya que presenta un entorno urbano que recorrer libremente, que a su vez es colorido y vibrante, resultando en una estética muy particular.

Captura de DEX

Captura de Super Mario Odyssey

3.1.3. Blocking

La fase de *blocking* se realizó utilizando como referencia diseños esquemáticos en papel del nivel y la herramienta de BSPs de Unreal, junto a algunos *assets* que cumplían una función de *placeholder*. Fue de mucha utilidad para establecer las dimensiones de la escala de los elementos, el campo de visión de la cámara y parámetros como la velocidad de movimiento o la altura del salto. A la hora de probar la jugabilidad se detectaron carencias que llevaron a los primeros cambios en el *layout* básico.



Captura del nivel de blocking

3.1.4. Construcción

La construcción del nivel se realizó sobre el *blocking*. Tras numerosas pruebas con diferentes paquetes de *assets*, se eligió el paquete *Polygon - City Pack*, que se ofreció de manera gratuita en la plataforma *Unreal Marketplace*. Este paquete presentaba un estilo similar al que se quería aspirar, numerosos elementos para construir los entornos y otros elementos del juego, era sencillo de usar, causaba un impacto bajo en el rendimiento y permitía múltiples opciones de configuración. Además, al tratarse de *assets* ya optimizados para su uso dentro de Unreal Engine, no era necesario preparar los materiales o la orientación de los ejes, como sí pueden requerir otros *assets* importados.

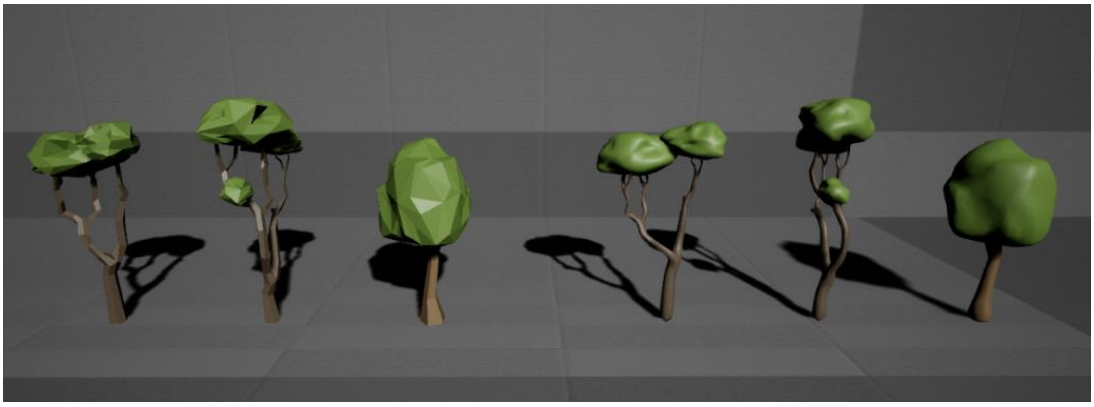
Mapa de ejemplo de Polygon - City Pack

Escaparate de assets de Polygon - City Pack

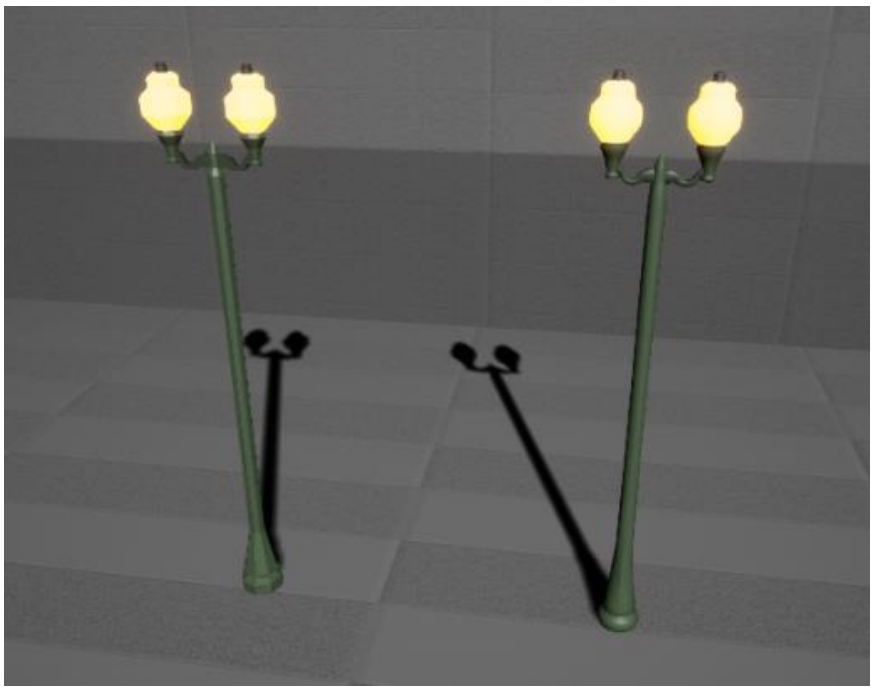
Incluso con estas facilidades, fue necesario dedicar tiempo a realizar modificaciones en algunos de los elementos de este paquete. Al tratarse de

assets con estilo *low-poly*, la carga poligonal es baja. En elementos inorgánicos, como los edificios, aceras u otros elementos decorativos de las fachas y las calles, esta característica es una ventaja, ya que permite un estilo visual claro y sencillo, que facilita la lectura del entorno y no obstaculiza la jugabilidad. Pero en elementos orgánicos, como los árboles o las rocas, o en superficies que requieren superficies curvas, este estilo desentona con el aspecto general del juego.

Para unificar estos elementos sin renunciar a las ventajas de usar estos *assets*, se optó por exportar estos *assets* desde Unreal a Maya y, mediante la herramienta de *Smooth*, se suavizó la geometría de los modelos, aumentando la carga poligonal y el detalle en un proceso automatizado. Una vez realizado el suavizado, se volvieron a importar los modelos en Maya, asignándoles los materiales originales y sustituyendo los modelos previos por los nuevos a través de Unreal, sin necesidad de volver a colocarlos, ahorrando así tiempo.

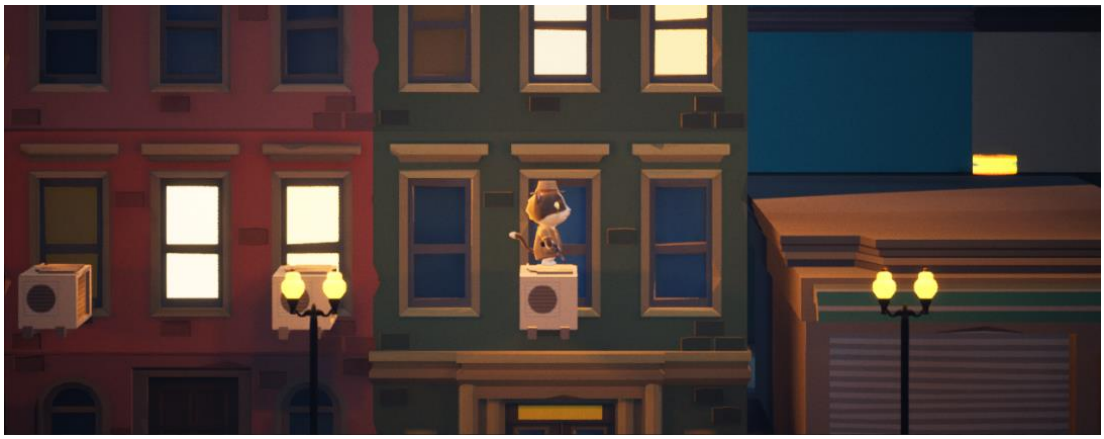


Comparativa de árboles originales con su versión suavizada



Comparativa de farola original con su versión suavizada

Otro inconveniente que se presentó al usar *assets* creados fue que las estructuras no estaban diseñadas para ser utilizadas en un videojuego de plataformas. Para conseguir que el personaje pudiera recorrer el escenario, hubo que modificar la escala de varios elementos y utilizar volúmenes de colisión invisibles para que el personaje pudiera posarse sobre ellos. Para evitar que estos ajustes fueran perceptibles, se aumentó la distancia focal de la cámara del personaje para reducir la profundidad y lograr así engañar al ojo del jugador. Pero este ajuste de la cámara supuso también tener que aumentar la distancia de los elementos y construir este escenario con este ajuste en mente.



Ejemplo de perspectiva 1: Escenario en la cámara del personaje

Ejemplo de perspectiva 1: Escenario en el viewport de Unreal Engine

Para crear profundidad, se colocaron edificios por detrás del plano principal de la calle donde transcurre la acción. Para el horizonte, se utilizaron edificios 2D como forillos extraídos del proyecto *Platformer Game* ofrecido gratuitamente por Epic Games para formación.
Para iluminar las ventanas de esos forillos de edificios, se crearon superficies rectangulares con un material emisivo que, en la distancia, simulan ser ventanas.

3.1.5. Iluminación

En un primer momento se buscaba la inspiración en la iluminación característica del cine *noir*, con contrastes duros, sombras muy marcadas y numerosas zonas en penumbra, pero resultaba poco práctica para jugabilidad, ya que en un videojuego de plataformas hay que poder ver hacia donde nos movemos.

Se trabajó en establecer varios planos de iluminación. El plano principal engloba al personaje, los elementos sobre los que se mueve y puede interactuar. Para determinar estos planos, se utilizaron los canales de Unreal. Se tomaba el canal más bajo para lo más lejano, y a partir de esos valores, se añadía iluminación en cada canal.

Para la construcción de la iluminación se utilizaron luces puntuales (*point lights*) en los elementos que emitían luz directa y se podían identificar como fuentes de luz principal, como las farolas. No se desestimó la importancia de la iluminación indirecta, generada en gran medida por el Blueprint del cielo junto a una luz direccional.

Para los carteles y los grupos de ventanas, se utilizaron *Rect Lights*, que permiten emitir luz desde un área con forma rectangular. Esto permitía contrastar los contornos de los personajes cuando se encontraban cerca de edificios.
Otros recursos que afectan a la iluminación son los tipos de niebla y las esferas de reflejos.

Ejemplo de Rect Light en un cartel y su influencia sobre un personaje

Se utilizaron materiales emisivos para superficies incandescentes que simulan emitir luz, como las farolas, pero también se emplearon para resaltar elementos jugables. Por ejemplo, los disparos tanto del personaje como de los enemigos tienen un canal emisivo para permitir que sean vistos incluso cuando la luz no les afecta. Del mismo modo, cuando los enemigos o el jugador reciben daño, cambian temporalmente a un color rojizo que también es emisivo, para que el jugador reciba la notificación de daño.

Proyectil con material emisivo

Enemigo recibiendo daño

3.1.6. Efectos

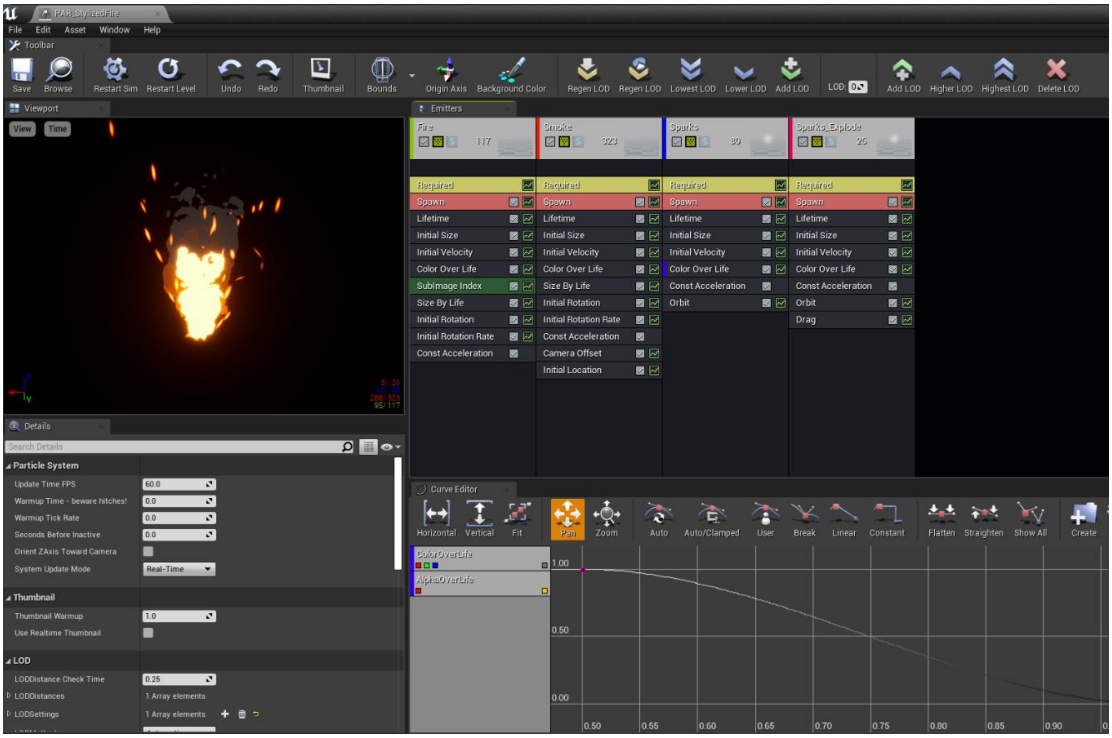
Chorro de agua

El chorro de agua está compuesto por cuatro partes:

- El chorro de agua
- La superficie donde impacta el agua
- Las salpicaduras
- El vapor

Ver Anexo III para más detalles sobre la construcción del material del chorro de agua.

Fuego

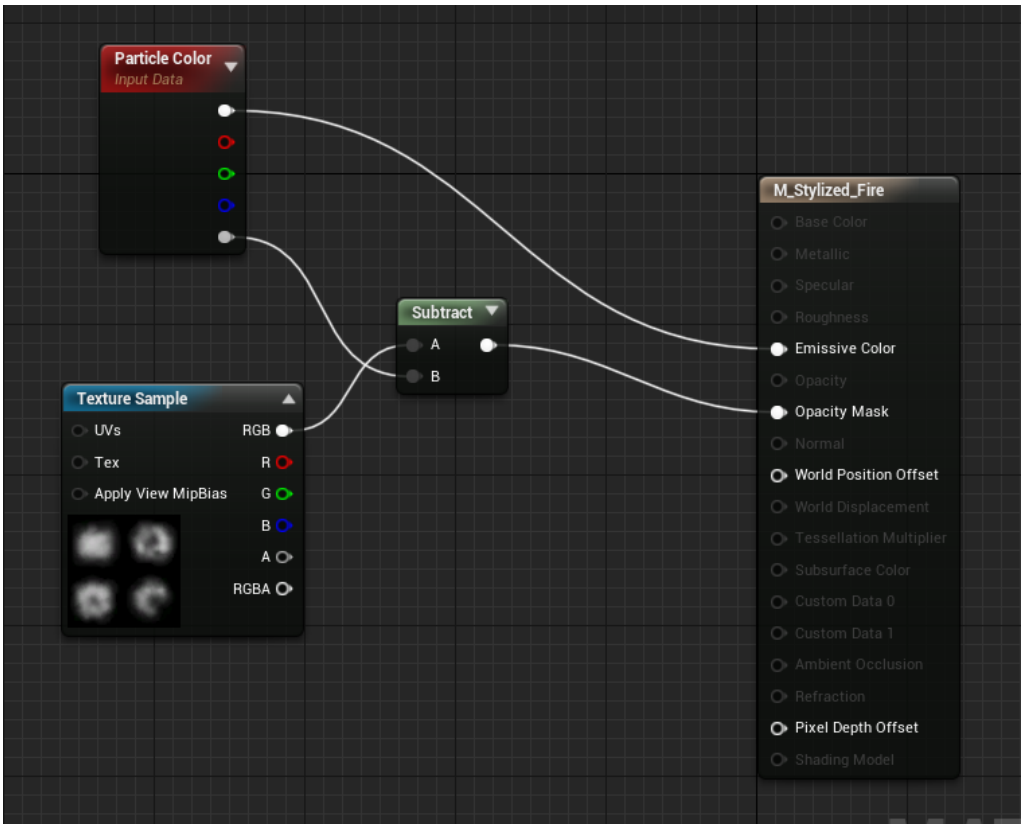


Sistema de partículas de fuego en el editor Cascade

El fuego es un sistema de partículas compuesto por cuatro *emitters*.
El primer *emitter* recrea las llamas. El volumen de las llamas se crea en un archivo de Photoshop. Sobre una capa con fondo negro, se crea una forma utilizando varios degradados circulares de color blanco, y a continuación se define utilizando la herramienta Borrador con un ajuste de dureza al 0%. Por último, se aplica un desenfoque gaussiano. El objetivo es conseguir una forma que evoque humo, con contornos que se desvanezcan, pero dejando suficiente margen en los bordes del archivo.

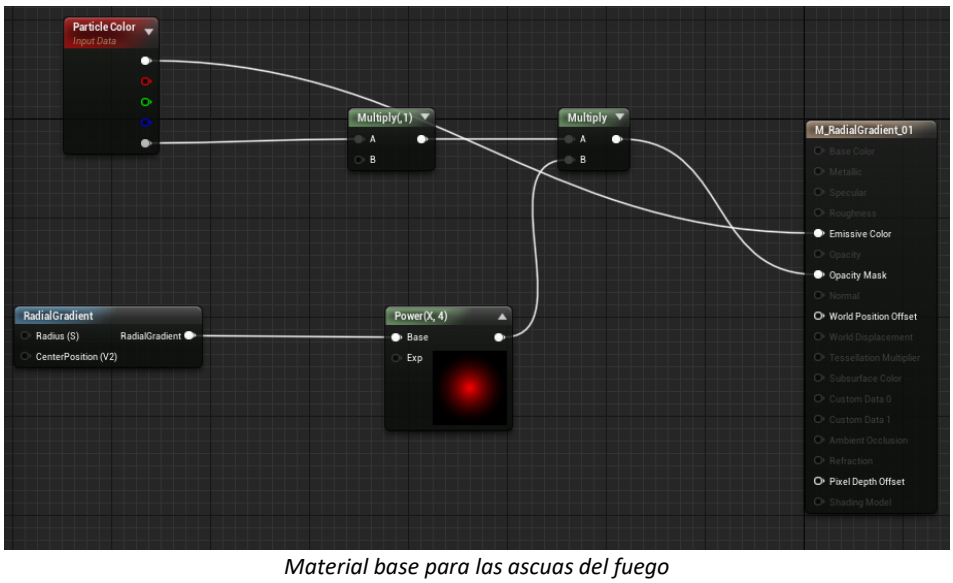
Se crean tres formas más utilizando el mismo procedimiento. A continuación, se componen las cuatro formas en un solo archivo, creando un mosaico de 2x2. El resultado se exporta como fichero TGA a Unreal.

En Unreal, se crea un material con las propiedades de *Material Domain - Surface*, con *Blend Mode - Masked* y *Shading Model - Unlit*. Se importa el archivo TGA con el mosaico de las cuatro formas como textura. Se crea un nodo *Particle Color*, que será el color de las partículas del fuego. Mediante un nodo Substract, conectamos la textura al canal A y el el canal Alpha del color al B. De esta manera, obtenemos las formas originales con el color de partículas que hemos creado. Conectamos el resultado al canal de *Opacity Mask* de la textura. Asignamos también la salida RGBA del color al canal de *Emissive Color*.



Material base para la forma de las llamas y del humo

Creamos otro material donde añadimos la función *RadialGradient* del *Engine Content* de Unreal. Conectamos su salida a un nodo Power ^4 y lo multiplicamos por el canal Alpha del nodo *ParticleColor*. El resultado se conecta al material por el canal de *Opactiy Mask.*



Material base para las ascuas del fuego

Con estos materiales creados, se procede a crear el sistema de partículas. El primer *emitter* generará el fuego. Utilizaremos el material de las formas de humo que hemos creado previamente. Mediante la propiedad de *Sub UV*, podemos

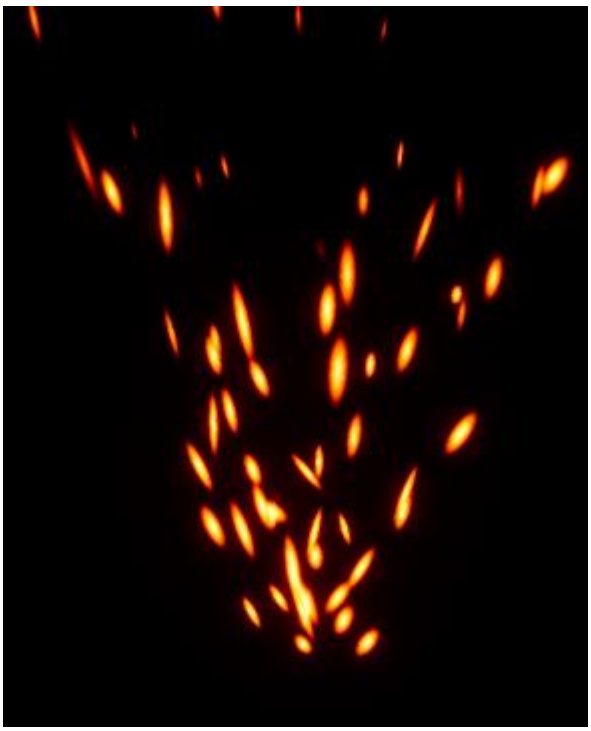
hacer que tome de manera aleatoria una de las cuatro casillas de la textura de mosaico con las formas. Así no hay que cargar varios *sprites* individuales. Ajustamos los valores de *Initial Life, Size By Life* e *Initial Velocity* para que las llamas se generen muy grandes en la base del fuego y se agiten mucho hacia los bordes, y a medida que ascienden encogen hasta que desaparecen. En el módulo de *Color Over Life* asignamos un color naranja intenso, que gracias a las propiedades del material de las partículas se mostrará emisivo.

Emitter de las llamas

Creamos un segundo *emitter* para recrear el humo. Tomamos la base del *emitter* del fuego, duplicándolo. En *Color Over Life*, creamos tres claves para que el humo empiece en un tono rojo oscuro cuando se genera a partir del fuego, para pasar a marrón y, por último, a gris oscuro antes de desaparecer. El humo también es más ancho y alto que las llamas, y debe generarse más arriba que el fuego. Es importante añadir un módulo Camera Offset al *emitter* para que el humo se genere siempre detrás del fuego, ya que si no el resultado es extraño, al solaparse las partículas del fuego y del humo.

Emitter del humo

El tercer y cuarto *emitter* sirven para crear chispas de fuego que flotan y estallan en el aire, respectivamente. Se utiliza el material creado previamente a partir de la función *RadialGradient* previamente, ya que es un método para hacer que las chispas incandescentes desaparezcan a medida que se apagan. Para hacer que las chispas floten alrededor del humo y del fuego se utiliza un módulo *Orbit*. El *componente Drag* permite generar explosiones intermitentes de chispas.

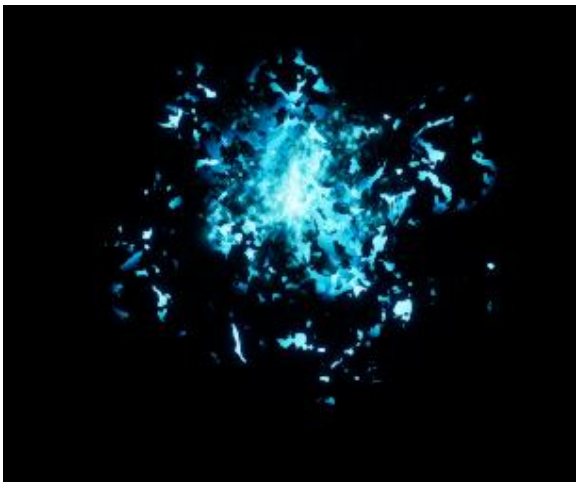


Emitter de las chispas

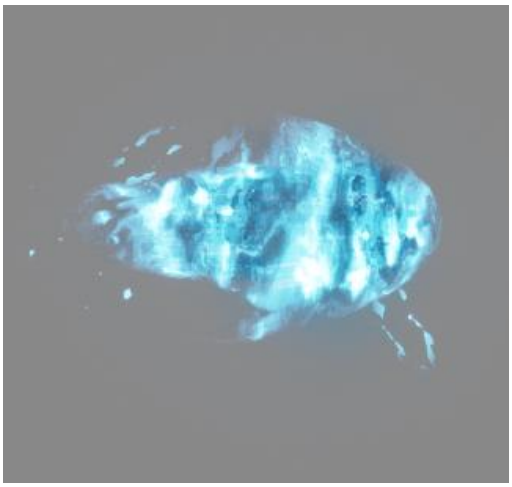
Resultado final del efecto del fuego

Los efectos de los proyectiles de agua, tanto cuando son lanzados como cuando impactan contra enemigos o superficies, provienen del paquete del Unreal Marketplace ***Advanced Magic FX 12.***

El efecto del impacto del agua se genera en el lugar en el que impacta un proyectil, bien sea un enemigo, el personaje o una superficie. El efecto del proyectil del agua forma parte del actor de los proyectiles.

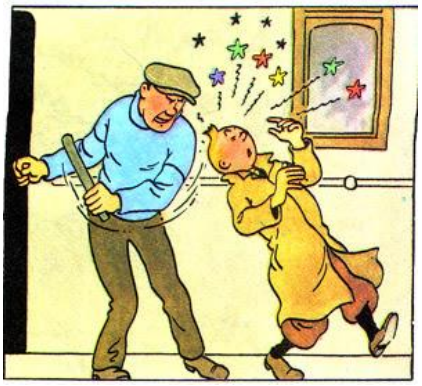


Efecto de impacto del proyectil

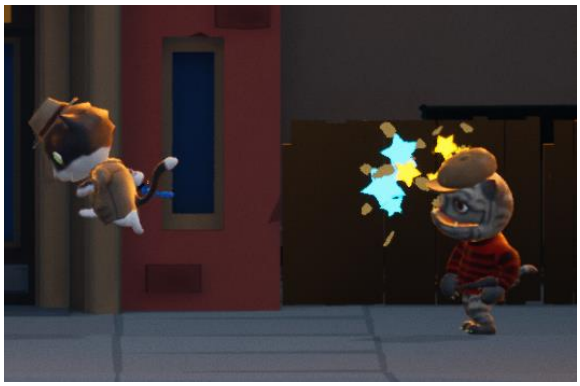


Efecto del proyectil de agua

El efecto de estrellas que se genera cuando el enemigo cuerpo a cuerpo golpea a Neil, así como el efecto de nube polvo que se genera cuando desaparecen los enemigos, se crearon a partir de efectos del paquete ***Luos's Particle Toolkit Vol. 1***. Se buscaba crear efectos con aspecto cómico, de estilo *cartoon,* que recordasen a una serie de dibujos animados.



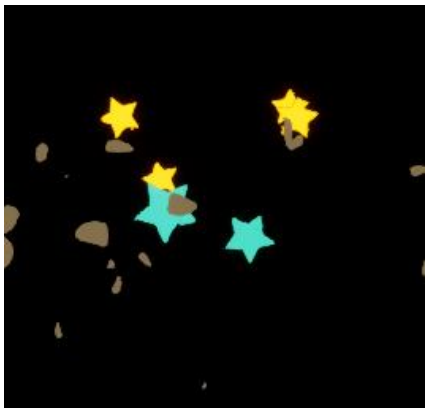
Referencia del efecto de golpe



Efecto del golpe en el juego

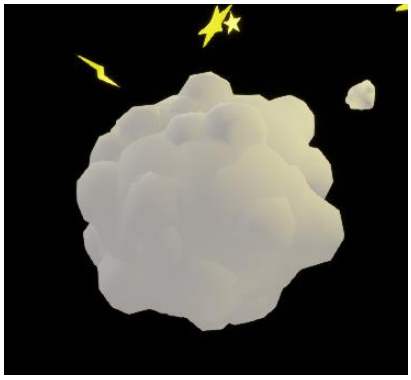
El efecto original del golpe soltaba estrellas, en una explosión, junto a una nube de polvo que se extendía en la periferia. Para ajustar el efecto del golpe, se modificó el efecto para generar estrellas en el momento del golpe, ocultando los demás emisores de partículas.

Efecto de golpe original

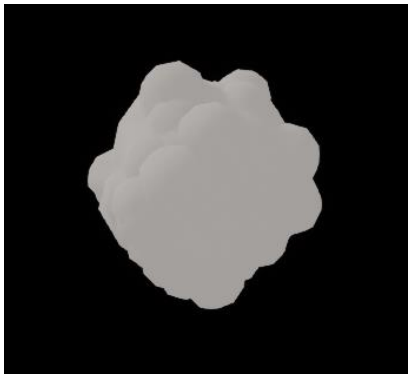


Efecto de golpe ajustado

El efecto original de desaparición del enemigo generaba una nube de polvo que emanaba estrellas durante varios segundos y luego desaparecía. Para el efecto de desapareción de los enemigos, se ocultaron las estrellas y se dejó la nube de polvo, pero generándose únicamente durante unos instantes.



Efecto de desaparición original



Efecto de desaparición ajustado

4. Jugabilidad

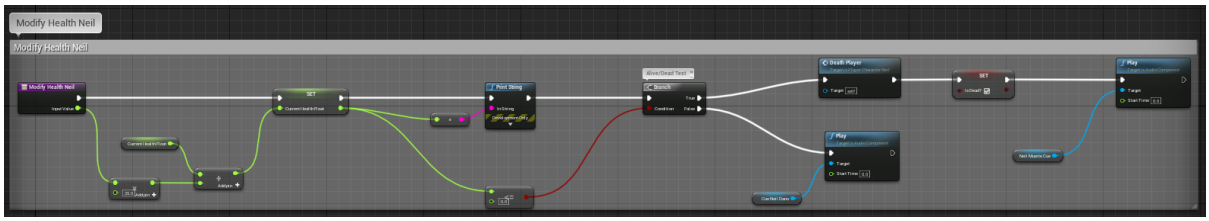
4.1. Programación

4.1.1. Salud

El sistema de salud del personaje del jugador se encuentra dentro del *Blueprint* del personaje. Consiste en una variable de tipo *float* con un valor por defecto de 200.

Para modificar la salud del personaje se creó una función con un valor *Input* que permite tanto sumar como restar salud, según si le hacen daño o si se el personaje se cura.

Cuando se llama a la función, se introduce un valor que se multiplica por 25 y se suma o resta a la salud actual, dependiendo de si el valor del *input* es positivo o negativo.



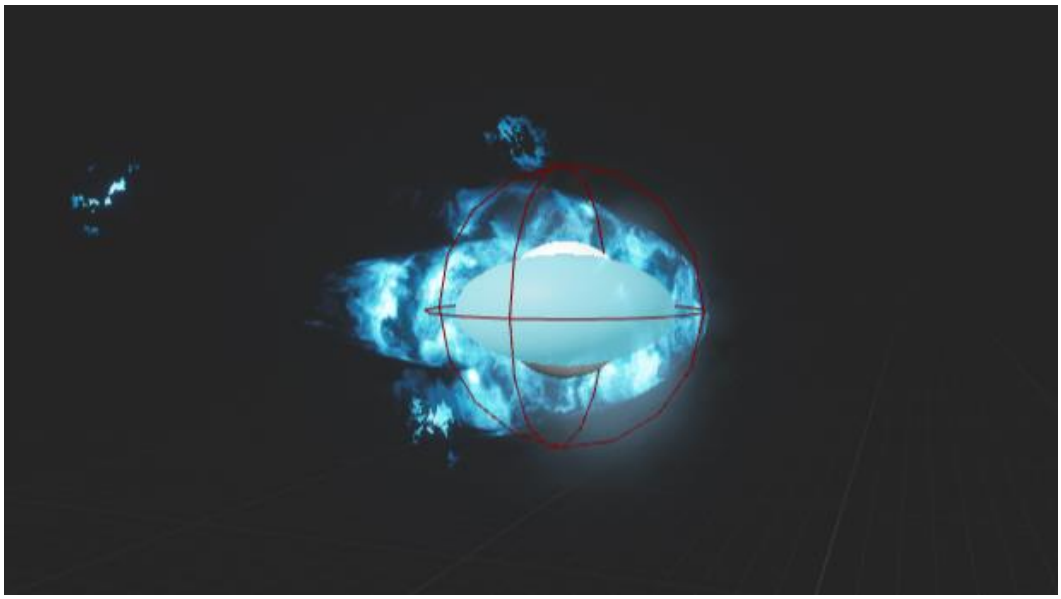
Modificación de salud del personaje

Se creó un pickup de salud que está representado por una lata de atún brillante. Este pickup restaura 25 puntos de salud de Neil. Si Neil tiene la salud completa, se puede recoger la lata, pero no tendrá efecto visible.

Pickup de salud

4.1.2 Proyectiles

Los proyectiles son actores generados por el jugador o por los personajes cuando realizan la acción de disparar. Están compuestos por una *mesh*, un sistema de partículas y una esfera de colisión.



Componentes del proyectil

Los proyectiles fueron uno de los elementos que más problemas causaron durante la programación de las mecánicas, ya que era necesario poder generar proyectiles que dañasen solo al personaje cuando fueran disparados por los enemigos, y que dañasen solo a los enemigos cuando fueran disparados por el personaje. Tras varias pruebas, se optó por generar dos tipos de proyectil: uno disparado por el personaje y otro disparado por los enemigos.

La base del funcionamiento de ambos proyectiles es idéntica. Cuando el personaje (o el enemigo) dispara, se ejecuta un nodo *SpawnActor BP* originado en un componente del personaje llamado *Projectile Spawn Location,* situado a la altura del cañón del arma cuando se realiza la animación de disparar. El nodo SpawnActor BP genera el proyectil, al que se le añade impulso tomando la dirección del vector del componente *Projectile Spawn Location,* para que salga disparado en la dirección a la que mira el personaje.

Dentro del Blueprint del proyectil, se establece con un nodo *Event BeginPlay* un valor de tipo *float* para determinar la *edad* del proyectil, que se incrementa cuando es disparado. Cuando dicha edad supera 1 (1 segundo), el actor es destruido.

Las diferencias entre los dos proyectiles son las siguientes:

- El proyectil que dispara el jugador comprueba, al colisionar, que el actor contra el que colisiona tiene la etiqueta “Enemy”. En caso afirmativo, ejecuta la secuencia de nodos que implican dañar al enemigo y destruirse. Existe también una etiqueta “Block”, que permite que el proyectil se destruya cuando impacta contra elementos del escenario que posean esa etiqueta y no los atraviese.

- El proyectil que dispara el enemigo realiza un *CastTo* al Blueprint de Neil y ejecuta la función *Modify Health Neil*.

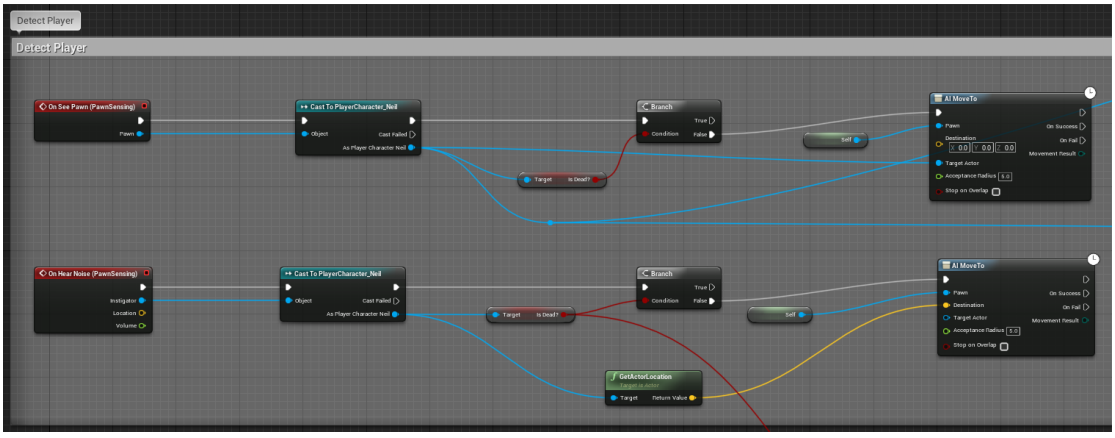
4.1.3 Enemigos

La programación de los enemigos está diseñada para atacar al jugador cuando se encuentra dentro de su cono de visión. En el *Blueprint* del personaje enemigo, se añade un componente *PawnSensing*, donde se ajusta el rango de visión y audición del personaje. A partir de ese componente se crea el evento *OnSeePawn (PawnSensing)*, y a continuación se realiza un *CastTo* al personaje del jugador y se comprueba, en primer lugar, con un nodo Branch, si está vivo. En caso afirmativo, la acción a realizar depende del tipo de enemigo.

El enemigo matón se acerca hasta Neil y le ataca cuerpo a cuerpo con su porra. El enemigo matón, al igual que el personaje protagonista, tiene asignado un *Blendspace 1D* para pasar de la animación *Idle* a la animación en carrera. También posee un *Animation Blueprint* con un sistema *Locomotion*. Dentro de *Locomotion*, se crea una *State Machine* y dentro de ella, se asigna el estado *Idle/Run*, donde se conecta el BlendSpace1D creado previamente conectado a la variable de velocidad del enemigo.

Dentro del *Event Graph* del *Animation Blueprint* del enemigo, se parte del evento *"Event Blueprint Update Animation",* desde donde se ejecuta una comprobación con el nodo *Is Valid* para comprobar que el nodo *Try Get Pawn Owner* es válido y a su vez, obtener el valor de la variable de velocidad del enemigo. A partir de ese valor, se calcula el *VectorLenght* y el resultado se conecta a un nodo *Set Speed* para definir la velocidad del personaje.

Utilizando el evento *PawnSensing* el enemigo se dirige al jugador. Para que le ataque al alcanzarlo, al *Blueprint* del enemigo, añadimos un volumen *Box.* Utilizaremos este volumen para que, cuando el personaje del jugador se encuentre dentro, el enemigo realice una animación de atacar.

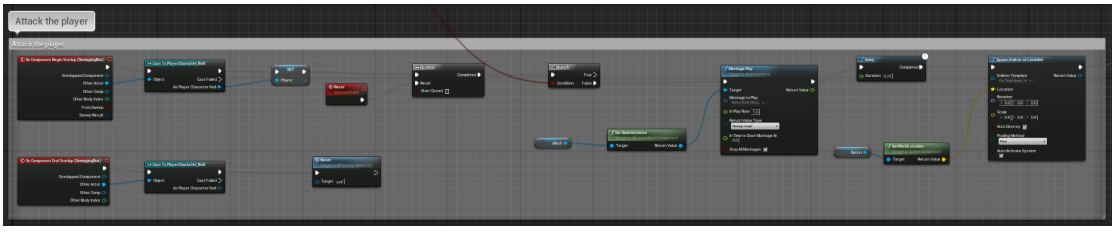


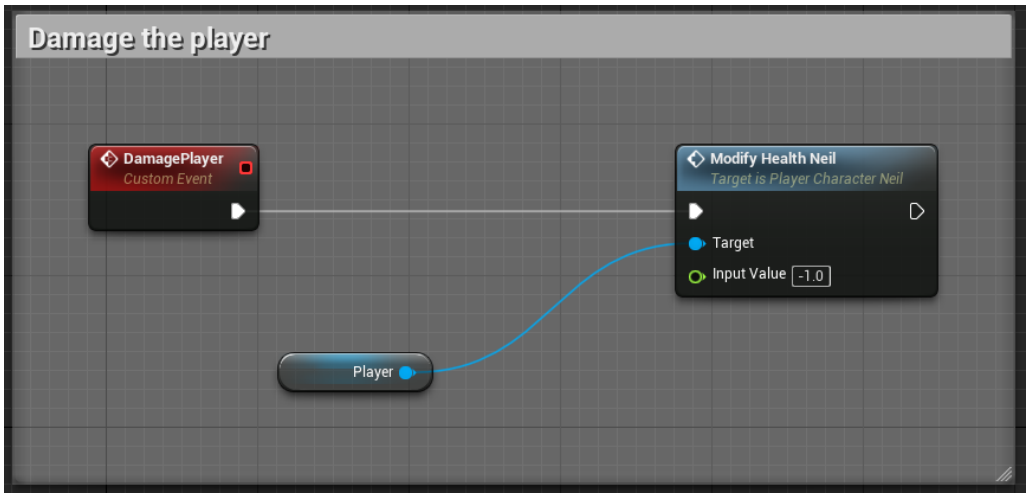
Detección y movimiento hacia el jugador

La animación de atacar parte de un *AnimMontage*, donde asignamos una animación de ataque con la porra. Usando los sistemas de *Notify*, añadimos un *Notify* cuando comienza el ataque y otro cuando termina.

Uso de Notify para atacar al jugador

El Notify de inicio del ataque permite ejecutar el *Custom Event Damage Player*, que dentro del *Blueprint* del enemigo ejecuta la función *Modify Health Neil*, que es la función a la que llaman todos los elementos que modifiquen la salud del personaje. Se añade un nodo *Delay* de 0,5 segundos para darle tiempo al enemigo a ejecutar la animación de ataque. El *Notify* de final del ataque llama al *Custom Event Reset*, que se conecta al nodo *Do Once* que inicia la secuencia de ataque, para evitar que el enemigo ataque solo una vez. Si no existiera el nodo *Do Once*, el enemigo no pararía de atacar.





Ataque y daño al jugador

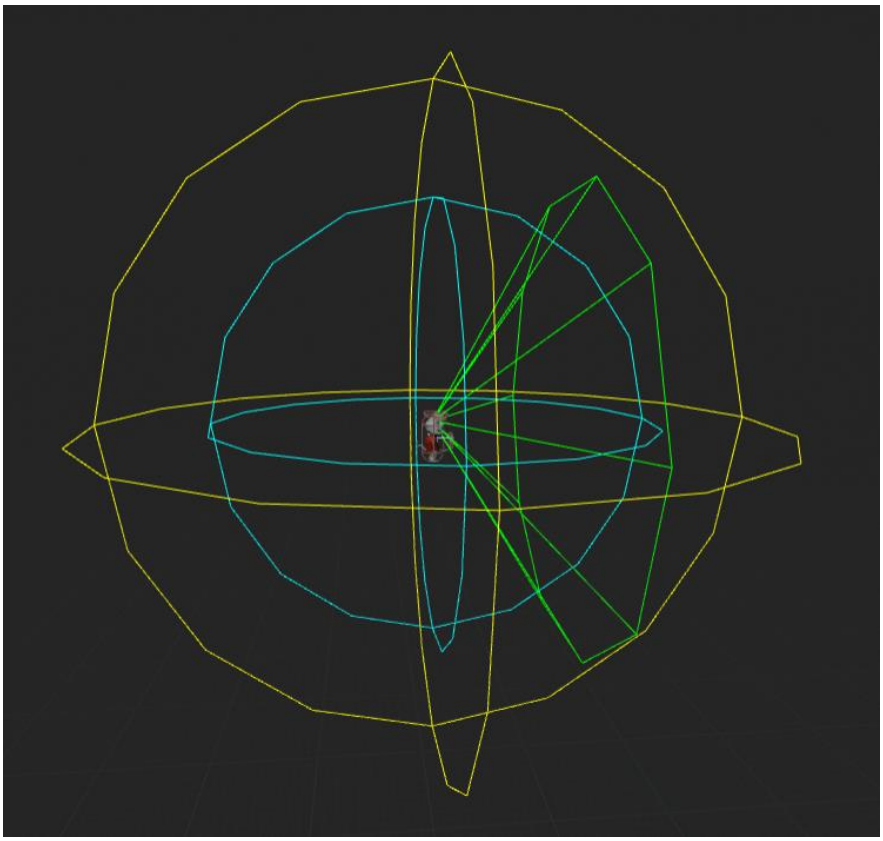
Además, en el momento en el que se daña al personaje, se le lanza usando el mismo método que con el fuego.

Para que el enemigo pudiera correr y atacar a la vez, se utilizó el mismo sistema que con el personaje protagonista, utilizando el nodo *Layered blend per bone* y *cached poses* para poder combinar dos animaciones en diferentes partes del esqueleto.

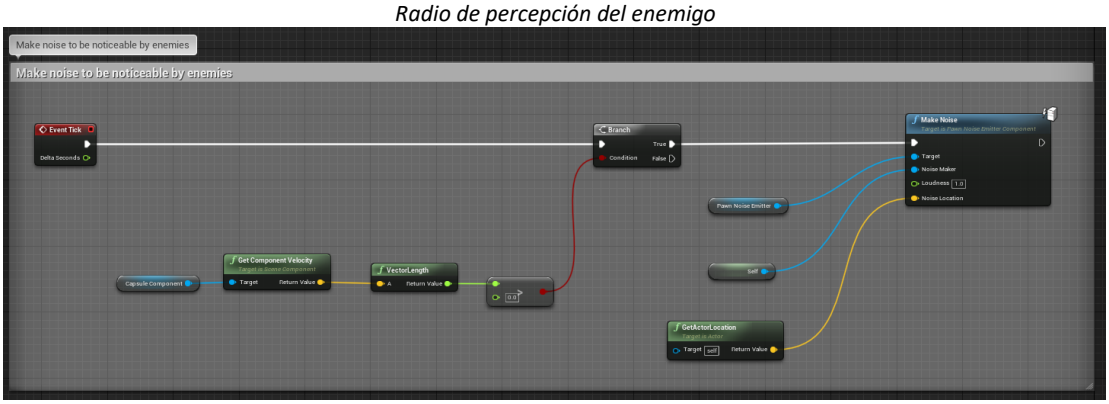
El enemigo pistolero no se desplaza, pero dispara proyectiles que dañan al personaje a distancia. Su rotación se ajusta según donde se encuentre el jugador.

Las balas disparadas por los enemigos son un tipo de actor distinto a las que dispara el personaje, para evitar que puedan dañar a sus aliados. En el apartado de creación de los disparos se explica el funcionamiento de los proyectiles.

Uno de los problemas que se detectó al comenzar las pruebas era que los enemigos no seguían al jugador cuando saltaba por encima de ellos. Tras varias pruebas, se optó por la solución de que el personaje del jugador generase ruido constantemente con la función *Make Noise*. De este modo, añadiendo un evento *On Hear Noise* al componente *PawnSensing* de los enemigos, los enemigos pueden oír al jugador en un radio reducido pero cuyos límites se encuentran dentro del rango de salto del jugador. De este modo, cuando el jugador salte por encima de ellos, ellos girarán con él.



Radio de percepción del enemigo



Emisión de ruido del jugador

Se puede acabar con los enemigos de dos maneras: disparándoles o saltando encima de ellos. Lo más efectivo es disparar, ya que es más seguro, permite un mayor rango y hace más daño. Sin embargo, en ocasiones puede ser útil saltar encima de los enemigos para ganar distancia cuando nos rodean, o cuando no se tiene un buen ángulo de tiro.

El sistema de daño de los enemigos funciona distinto según la acción que se ejecute. Cuando el enemigo es disparado, se lanza un evento *Event Any Damage*, ya que es daño causado por el actor de la bala lanzada por el jugador. Cuando se le salta encima, se comprueba en el volumen que lleva el enemigo en la cabeza si el actor que causa el *Overlap* tiene la etiqueta "*Neil*" con el nodo *"Actor Has Tag".* Así, se evita que el enemigo se dañe al entrar en contacto con la cabeza con cualquier otro elemento del escenario o del juego, como las balas de otros enemigos. Debido a que las balas ya tienen un valor de daño asociado, para

restar salud cuando se salta en la cabeza se crea una variable específica para el daño causado por los saltos.

4.1.4 Entornos

Fuego

El fuego cumple la función de obstáculo. Es dañino para el jugador. Si el jugador entra en contacto con el fuego, sufrirá 25 unidades de daño y saldrá despedido. El jugador no puede saltar por encima del fuego desde el nivel del suelo. Debe dar un rodeo y saltar entre los balcones del hotel para superarlo.
Originalmente, el fuego iba a hacer daño por segundo al permanecer sobre él, pero durante las pruebas de jugabilidad se detectó que podía ser más ventajoso atravesarlo aún a costa de sufrir daño. Empujando al jugador en el sentido contrario al que avanza y causándole daño se inculca mejor la idea de que el fuego debe evitarse.

Puesto de perritos calientes

El puesto de perritos calientes funciona como trampolín y permite impulsarse a lugares elevados. Cuando el jugador toca el volumen de colisión asignado a la sombrilla del puesto, sale disparado en la dirección del eje Z.

Aire acondicionado suelto

Los aparatos de aire acondicionados sirven como plataformas entre ventanas. Los que están sueltos presentan una textura más oscura y tiemblan al pisarlos, para después caerse.
La animación de balanceo previa al derrumbe se creó utilizando una *timeline* donde se creó una línea ondulante en el eje X mediante fotogramas clave alternando entre valores positivos y negativos. El resultado de esta *timeline* se conectó a un nodo *SetRelativeTransform*.

Manivela y chorro de agua

El chorro de agua se encuentra al final del nivel y bloquea la entrada al edificio de los hermanos Pawson. Neil debe encontrar la manivela y colocarla en la tubería para cerrar el paso del agua.
La manivela es un *pickup* cuyo Blueprint activa una variable Booleana en el personaje que indica que tiene la manivela. La manivela tiene un material pulsante emisivo que hace que destaque e indica que es un objeto importante que se debe recoger. Solo Neil puede ver que brilla, ya que su instinto detectivesco está tan agudizado que percibe la realidad de una manera distinta a los demás felinos.

En las tuberías cercanas al depósito se colocó un volumen de tipo *TriggerBox*. Este volumen comprueba si Neil tiene la manivela. Si no la tiene, un mensaje indica que se debe ir a buscarla. Si la tiene, se activa una cinemática en la que se corta el agua y se abre el paso.

5 Interfaz

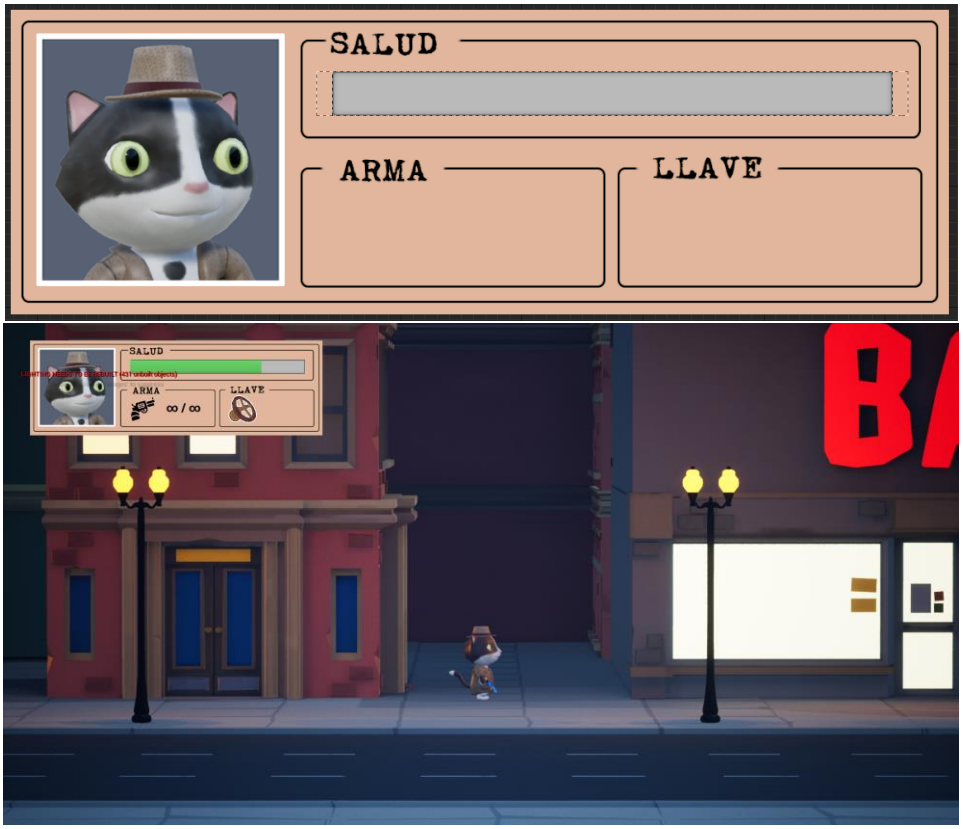
5.1 HUD (Heads Up Display)

El interfaz del HUD se sitúa en la parte superior izquierda de la pantalla. Presenta un retrato del personaje, una barra de salud y un icono con el arma que se está usando en ese momento, que muestra su cargador y la munición total. En la demostración del juego muestra un símbolo de infinito, ya que el sistema de munición no está implementado.

La barra de salud representa la salud actual del personaje. Es una barra de progreso conectada a la variable de salud del personaje. Se vacía a medida que sufre daño y se rellena al recuperar salud.

También incluye un apartado “Llave”, donde se añade la válvula cuando se recoge y que desaparece al colocarla.

Los elementos de la interfaz se superponen unos sobre otros. Los iconos de la pistola y de la llave se superponen al HUD con el retrato y la salud del personaje, mediante el parámetro *ZOrder*, que determina el orden de renderizado. Todos los elementos están anclados a la esquina superior izquierda de la pantalla, para que no se descoloquen en caso de maximizar la ventana del juego.



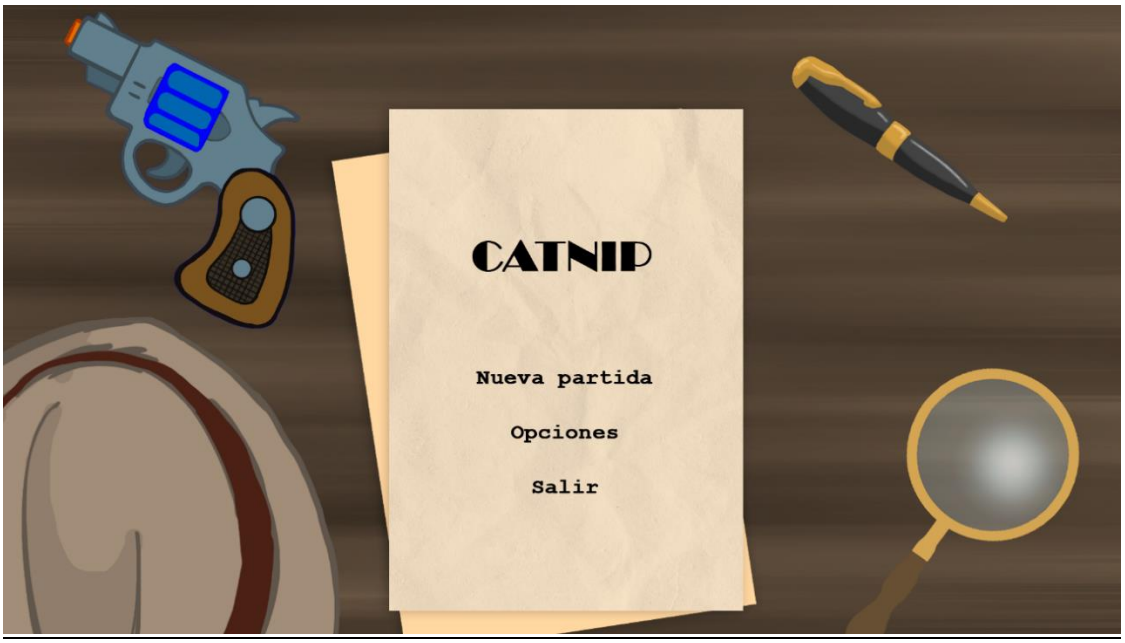
Diseño del HUD principal y su representación durante el juego

5.2 Menú principal

El menú principal representa la mesa del despacho de los Pawson. Se ha buscado un diseño diegético integrado con los elementos del juego.

Para realizar el menú, se ha utilizado un proyecto gratuito para formación de Epic Games del curso “**Build a Detective's Office Game Environment”**

Primero se realizó una composición en Photoshop, a modo de mockup, para esbozar la idea. Luego, utilizando el proyecto mencionado, se compuso una escena similar, utilizando elementos como el sombrero del protagonista o su pistola. Se colocó la cámara apuntando hacia el escritorio desde una perspectiva en picado y se realizó una captura de pantalla a alta resolución. Después, en Photoshop se ajustó la imagen y se añadió el logotipo del proyecto, junto a las opciones del menú principal. La imagen se exportó de nuevo a Unreal, donde se añadió una capa de funcionalidad a través de botones.



Mockup del menú principal

Segunda versión del menú principal

Versión final del menú principal

Cuadros de diálogo

Al realizar ciertas acciones, aparecen cuadros de diálogo emergentes sobre la cabeza del personaje protagonista. Su función es informar al jugador sobre el contexto de la acción realizada. La mayoría de ellos están narrados en primera persona por el protagonista, que aparece indicado como "NEIL" en el marco de la esquina superior izquierda. Los cuadros de texto están anclados para ocupar toda la pantalla, para que siempre parezca que se generan encima del personaje.

Diálogo emergente en la entrada al metro

El funcionamiento de estos diálogos está regulado por elementos de tipo *Box Volume* dentro del nivel. Igualmente, su programación está realizada dentro del Level Blueprint. El principio de su funcionamiento es similar, aunque varía según el diálogo.

El primer diálogo que puede ver el jugador se encuentra si se dirige a la izquierda de la pantalla, hacia la entrada a la estación de metro. Cuando el jugador está dentro del volumen asociado al mensaje, se añade el cuadro de diálogo a su pantalla. Cuando sale del volumen, el mensaje desaparece tras un breve lapso.

En la parte final del nivel, donde Neil debe cortar el paso del agua para llegar hasta el despacho, se presentan más diálogos con funcionamientos propios.

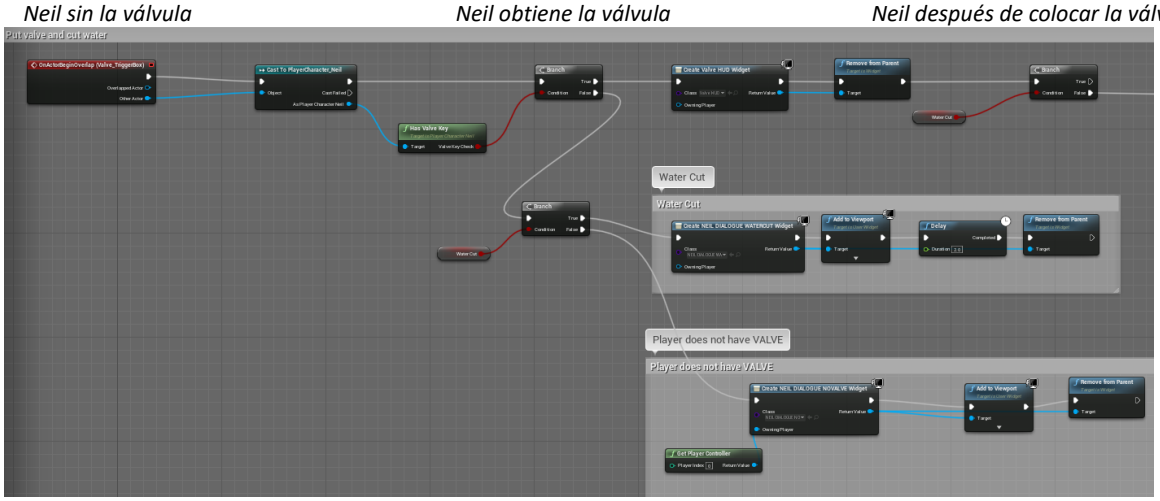
En la azotea donde está la tubería donde Neil debe colocar la válvula, el mensaje que aparecerá está condicionado por las siguientes variables:

- Si Neil tiene o no la válvula.
- Si el agua está cortada.

Neil sin la válvula

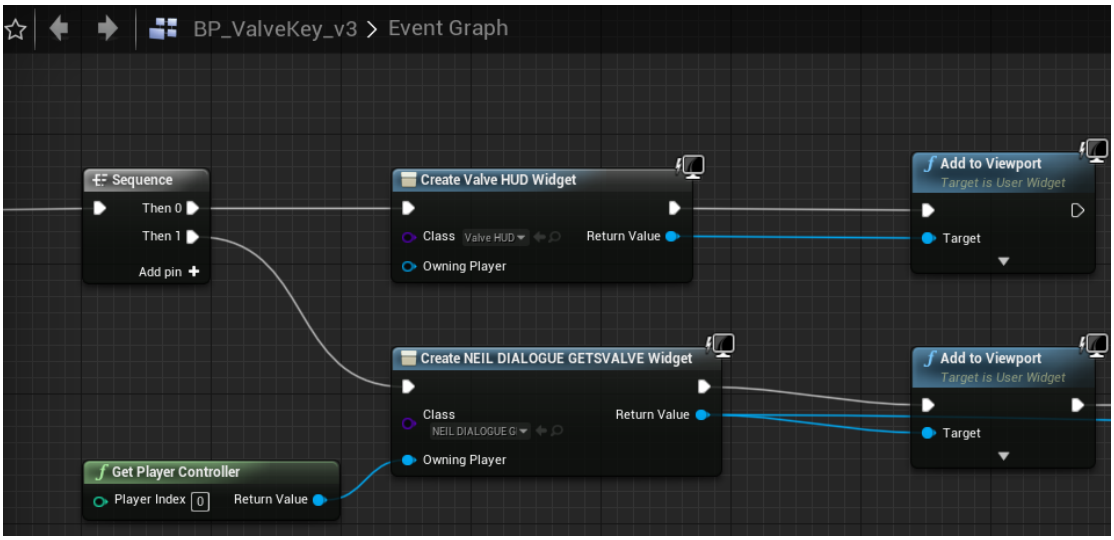
Neil obtiene la válvula

Neil después de colocar la válvula



Red de nodos que regula el diálogo a mostrar en la tubería

Cuando recoge la válvula, el icono de la válvula aparece en el HUD, y Neil muestra un diálogo. Estos eventos se recogen dentro del *Blueprint* de la válvula.



El Blueprint de la válvula añade el icono al HUD y el cuadro de diálogo

5.3 Otros cuadros de diálogo

Muestran información dirigida desde el juego al jugador, como si fueran relatados por un narrador extradiegético.

Pickup de la pistola

En este caso, el cuadro de texto es un material, aplicado sobre un plano dentro del actor del pickup del arma. No se añade al *viewport* del personaje, como el resto de los elementos de la interfaz del juego.

Ejemplo de un cuadro de texto dentro de un pickup y cómo un elemento del escenario se superpone

Game Over

Menú de game over

Este cuadro de texto está ligado a la función de muerte del jugador. Cuando el jugador muere, se arrebata el control al jugador y aparecen dos botones que el jugador debe pulsar con el cursor del ratón. Una opción reinicia el nivel y la otra quita el juego. Al pasar el cursor sobre los botones, se tintan de un color.

Pantalla de nivel completado

Menú de game over

Este cuadro de texto aparece cuando el jugador llega al portal del edificio de los Pawson, al final del nivel. Se le felicita por haber llegado hasta el final la demostración, pero a diferencia de la pantalla de Game Over, solo hay una opción para quitar el juego.

Bibliografía

- Virtus Dev Squad | Free GameDev Courses
  Recuperado de https://www.virtushub.com/
- Horacio Meza: Programación y Videojuegos
- http://www.horaciomezadev.com/
  Recuperado de https://www.thegnomonworkshop.com/tutorials/introduction-to-visual-effects-for-games-in-unreal
- Unreal Engine 4 3rd person tutorial Daniel Stan
  (Lista de reproducción de Youtube) Recuperado de
  https://www.youtube.com/watch?v=HCpuchWXSGY&list=PLAZLBDxdRhE4NhwxgGFU8emsLTslFoP83&index=33&t=261s&ab_channel=balaganvfx
- Polygon City Pack
  Recuperado de https://syntystore.com/products/polygon-city-pack
- Platformer Game Sample (Documentación de Epic Games)
  Recuperado de https://docs.unrealengine.com/en-US/Resources/SampleGames/PlatformerGame/index.html
- Coordinates Expressions (Documentación de Epic Games)
  Recuperado de https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/ExpressionReference/Coordinates/index.html
- UE4 - AI Chase/ Melee Attack Player (Canal de Joseph Whitworth)
  Recuperado de https://youtu.be/Zq4fUqGWsk8
- Build a Detective's Office Game Environment
  Recuperado de https://www.unrealengine.com/en-US/onlinelearning-courses/build-a-detectives-office-game-environment

Bancos de recursos

- Free Sound Library | Big Sound Bank
  Recuperado de https://bigsoundbank.com/
- OpenGameArt.org
  Recuperado de https://opengameart.org/
- CGTrader | Download 3D Models
  Recuperado de https://www.cgtrader.com/
- TakeTones | ROYALTY FREE MUSIC FOR VIDEO
  Recuperado de https://taketones.com/
- TurboSquid | 3D Models for professionals
  Recuperado de https://www.turbosquid.com/

Música

- Menú principal
  "Me & You", por Jam Morgan.
  Recuperado de https://taketones.com/track/total-fun
- Nivel
  "Total Fun", por Jam Morgan.
  Recuperado de https://taketones.com/track/total-fun