

TRABAJO FIN DE MÁSTER

MÁSTER EN PROGRAMACIÓN DE VIDEOJUEGOS

“Monster Heroes: Time Travelling Odyssey”



Autoría:

Baena Toquero, Manuel José
Díaz-Estébanez Arias, Jesús
Domínguez Gómez, Manuel Jesús
Sánchez Sinisterra, Irene

Supervisión:

Aniol Batallé Pararols

Tres Cantos, Julio de 2023.

Confirmación de originalidad del trabajo presentado

D/D.^a Manuel José Baena Toquero, con DNI/Pasaporte nº 75159134X estudiante del Máster en Programación de videojuegos. DECLARO Y GARANTIZO que el Trabajo Fin de Máster titulado Monster Heroes: A Time travelling Odyssey, es exclusivamente resultado de mi esfuerzo intelectual y, por tanto, que el contenido es original, que no es copia, traducción ni cualquier tipo de transformación, alteración, versión o modificación de ninguna clase de trabajo, obra o creación perteneciente a un tercero, y que no vulnera ninguna previsión legal, contrato, derecho o propiedad de tercero ni constituye un acto de competencia desleal.

En todo caso, me hago responsable delante de THE CORE y, por tanto, eximo a esta, de todas las cargas pecuniarias a favor de terceros, que se pudieran derivar para THE CORE con motivo de acciones, reclamaciones o conflictos derivados del incumplimiento de las manifestaciones y obligaciones contenidas en el párrafo anterior por mi parte.

Además, confirmo específicamente que las fuentes consultadas que he utilizado para la realización de dicho trabajo, si las hubiera, están correctamente referenciadas en el cuerpo del texto, en forma de cita, y en la bibliografía final.

Asimismo, declaro conocer y aceptar que de acuerdo con la Normativa académica vigente en THE CORE, el plagio del Trabajo Fin de Máster entendido como la presentación de un trabajo ajeno o la reproducción total o parcial de textos sin citar su procedencia y considerándolos como de elaboración propia, conllevará automáticamente la calificación de “suspenso” (0) tanto en convocatoria ordinaria como en extraordinaria. El estudiante deberá matricular la asignatura en el siguiente curso académico. Lo dispuesto en esta normativa se aplicará sin perjuicio de las responsabilidades disciplinarias en las que pudiera incurrir el estudiante por el plagio de conformidad con lo previsto en el Reglamento Disciplinario de la Escuela (con la consideración de falta muy grave)

Fecha y firma: 09/07/2023

Manuel José Baena Toquero

Confirmación de originalidad del trabajo presentado

D/D.^a Jesús Díaz-Estébanez Arias, con DNI/Pasaporte nº51490070Q estudiante del Máster en Programación de videojuegos DECLARO Y GARANTIZO que el Trabajo Fin de Máster titulado Monster Heroes: A Time travelling Odyssey, es exclusivamente resultado de mi esfuerzo intelectual y, por tanto, que el contenido es original, que no es copia, traducción ni cualquier tipo de transformación, alteración, versión o modificación de ninguna clase de trabajo, obra o creación perteneciente a un tercero, y que no vulnera ninguna previsión legal, contrato, derecho o propiedad de tercero ni constituye un acto de competencia desleal.

En todo caso, me hago responsable delante de THE CORE y, por tanto, eximo a esta, de todas las cargas pecuniarias a favor de terceros, que se pudieran derivar para THE CORE con motivo de acciones, reclamaciones o conflictos derivados del incumplimiento de las manifestaciones y obligaciones contenidas en el párrafo anterior por mi parte.

Además, confirmo específicamente que las fuentes consultadas que he utilizado para la realización de dicho trabajo, si las hubiera, están correctamente referenciadas en el cuerpo del texto, en forma de cita, y en la bibliografía final.

Asimismo, declaro conocer y aceptar que de acuerdo con la Normativa académica vigente en THE CORE, el plagio del Trabajo Fin de Máster entendido como la presentación de un trabajo ajeno o la reproducción total o parcial de textos sin citar su procedencia y considerándolos como de elaboración propia, conllevará automáticamente la calificación de “suspenso” (0) tanto en convocatoria ordinaria como en extraordinaria. El estudiante deberá matricular la asignatura en el siguiente curso académico. Lo dispuesto en esta normativa se aplicará sin perjuicio de las responsabilidades disciplinarias en las que pudiera incurrir el estudiante por el plagio de conformidad con lo previsto en el Reglamento Disciplinario de la Escuela (con la consideración de falta muy grave)

Fecha y firma:

Fecha y firma: 09/07/2023

Jesús Díaz-Estébanez Arias

Confirmación de originalidad del trabajo presentado

D/D.^a Manuel Jesús Domínguez Gómez, con DNI/Pasaporte nº47547326R estudiante del Máster en Programación de videojuegos DECLARO Y GARANTIZO que el Trabajo Fin de Máster titulado Monster Heroes: A Time travelling Odyssey es exclusivamente resultado de mi esfuerzo intelectual y, por tanto, , que el contenido es original, que no es copia, traducción ni cualquier tipo de transformación, alteración, versión o modificación de ninguna clase de trabajo, obra o creación perteneciente a un tercero, y que no vulnera ninguna previsión legal, contrato, derecho o propiedad de tercero ni constituye un acto de competencia desleal.

En todo caso, me hago responsable delante de THE CORE y, por tanto, eximo a esta, de todas las cargas pecuniarias a favor de terceros, que se pudieran derivar para THE CORE con motivo de acciones, reclamaciones o conflictos derivados del incumplimiento de las manifestaciones y obligaciones contenidas en el párrafo anterior por mi parte.

Además, confirmo específicamente que las fuentes consultadas que he utilizado para la realización de dicho trabajo, si las hubiera, están correctamente referenciadas en el cuerpo del texto, en forma de cita, y en la bibliografía final.

Asimismo, declaro conocer y aceptar que de acuerdo con la Normativa académica vigente en THE CORE, el plagio del Trabajo Fin de Máster entendido como la presentación de un trabajo ajeno o la reproducción total o parcial de textos sin citar su procedencia y considerándolos como de elaboración propia, conllevará automáticamente la calificación de “suspenso” (0) tanto en convocatoria ordinaria como en extraordinaria. El estudiante deberá matricular la asignatura en el siguiente curso académico. Lo dispuesto en esta normativa se aplicará sin perjuicio de las responsabilidades disciplinarias en las que pudiera incurrir el estudiante por el plagio de conformidad con lo previsto en el Reglamento Disciplinario de la Escuela (con la consideración de falta muy grave

Fecha y firma:09/07/2023

Manuel Jesús Domínguez Gómez

Confirmación de originalidad del trabajo presentado

D/D.^a Irene Sàncchez Sinisterra, con DNI/Pasaporte nº 47570638Z estudiante del Máster en Programación de Videojuegos DECLARO Y GARANTIZO que el Trabajo Fin de Máster titulado Monster Heroes: Time Travelling Odyssey es exclusivamente resultado de mi esfuerzo intelectual y, por tanto, , que el contenido es original, que no es copia, traducción ni cualquier tipo de transformación, alteración, versión o modificación de ninguna clase de trabajo, obra o creación perteneciente a un tercero, y que no vulnera ninguna previsión legal, contrato, derecho o propiedad de tercero ni constituye un acto de competencia desleal.

En todo caso, me hago responsable delante de THE CORE y, por tanto, eximo a esta, de todas las cargas pecuniarias a favor de terceros, que se pudieran derivar para THE CORE con motivo de acciones, reclamaciones o conflictos derivados del incumplimiento de las manifestaciones y obligaciones contenidas en el párrafo anterior por mi parte.

Además, confirmo específicamente que las fuentes consultadas que he utilizado para la realización de dicho trabajo, si las hubiera, están correctamente referenciadas en el cuerpo del texto, en forma de cita, y en la bibliografía final.

Asimismo, declaro conocer y aceptar que de acuerdo con la Normativa académica vigente en THE CORE, el plagio del Trabajo Fin de Máster entendido como la presentación de un trabajo ajeno o la reproducción total o parcial de textos sin citar su procedencia y considerándolos como de elaboración propia, conllevará automáticamente la calificación de “suspenso” (0) tanto en convocatoria ordinaria como en extraordinaria. El estudiante deberá matricular la asignatura en el siguiente curso académico. Lo dispuesto en esta normativa se aplicará sin perjuicio de las responsabilidades disciplinarias en las que pudiera incurrir el estudiante por el plagio de conformidad con lo previsto en el Reglamento Disciplinario de la Escuela (con la consideración de falta muy grave

Fecha y firma: 09/07/2023

Irene Sàncchez Sinisterra

ÍNDICE

AGRADECIMIENTOS	1
RESUMEN DEL PROYECTO	3
ABSTRACT	4
PARTE I. PRESENTACIÓN DEL PROYECTO	5
El equipo	5
Motivación personal	7
Oportunidad	9
Objetivos	10
Plan de desarrollo	11
Reflexión crítica	13
PARTE II. DESARROLLO DEL VIDEOJUEGO	15
2.1. Análisis previo	17
2.1.1. Análisis inicial y estado del arte	17
2.1.2 Hitch y Engagement	24
2.1.3 Conclusiones	26
2.2 Estética	28
2.2.1 Shader de personaje y entorno	28
2.2.2 Texturas	31
2.3 Coleccionables y trampas	39
2.3.1 Huevos	39
2.3.2 Trampas	40
2.4 Diseño de niveles	49
2.4.1 Estructura	49
2.4.2 Niveles	50
2.5 Definición del Gameplay	61
2.5.1 Sistema de interacción	61
2.5.2 Acciones del jugador	64

2.5.3 Control del personaje	67
2.5.4 Animación del personaje	71
2.5.5 Control de cámara	73
2.6 Interfaz de Usuario (UI)	75
2.6.1 Splash Screen	75
2.4.2 Botones	81
2.4.3 InGame UI	88
2.6.1 Planteamiento general del Sistema de Eventos	106
2.6.2 Ejemplo 1: Desactivación de bloqueos	108
2.6.3 Carga y descarga dinámica de secciones de nivel	110
2.7 NPCs e Inteligencia artificial	114
2.7.1 Definición de personajes	114
2.7.2 Feedback de personajes	115
2.7.3 Capabilities - Sistema de capacidades	117
2.7.4 Inverse Kinematics	124
2.7.5 Definición del agente inteligente	128
2.8 Pruebas de Calidad	151
BIBLIOGRAFÍA	152
ANEXOS	154

AGRADECIMIENTOS

Baena Toquero, Manuel José

Quisiera agradecer el apoyo de mi mujer y mi hija. Ellas son mi motivación para seguir trabajando cada día, seguir estudiando, formándome y creciendo como persona y profesional. Sin su apoyo, cariño y sacrificio diario, no habría sido capaz de terminar este máster.

También quiero agradecer al resto de miembros de este equipo, y a Aniol, por todo su trabajo, colaboración y por los buenos momentos que hemos pasado trabajando y aprendiendo juntos. Ha sido un placer colaborar con tan magnífico grupo de personas.

Díaz-Estébanez Arias, Jesús

En primer lugar, me gustaría agradecer tanto a mis padres como a mis hermanas por toda su dedicación y paciencia que han tenido conmigo a lo largo de este máster.

A su vez, también me gustaría agradecer a mis compañeros de máster como a los profesores por su constancia y profesionalidad a lo largo de este año académico.

Y por último quiero felicitar a mis compañeros de equipo por realizar este esfuerzo sobrehumano con este proyecto de fin de grado, finalmente el esfuerzo ha dado sus frutos.

Dominguez Gómez, Manuel Jesús

Quiero agradecer el apoyo incondicional de mis padres, sin ellos nunca habría podido llegar donde estoy ahora y son mi motivación para seguir trabajando y estudiando duro.

Agradecer a mis compañeros de trabajo, que sin ellos este trabajo no hubiera sido ni la mitad de lo que es ahora y a Aniol, que nos ha brindado su experiencia en la creación de videojuegos y siempre ha estado ahí.

Sánchez Sinisterra, Irene

Quiero agradecer el apoyo incondicional y ayuda brindados por mi hermana mayor, que ha estado ahí en todos los altibajos y siempre ha creído en mí como persona y en mis capacidades personales y profesionales.

Agradecer también a mis compañeros de vida animales Tita, Milito, Grisi, Mià y Nebla por quedarse a mi lado siempre que he estado trabajando, a Titeuf un perro que aunque ahora no convivamos en el mismo espacio como antes sé que me acompaña, e incluso a aquellos que ya no están: Tor, Coco, Mi, Crunchi, Tarzán.

Otro de mis grandes agradecimientos va dirigido a Manuel José Baena por estar siempre disponible para ayudarme con mis tropiezos y enseñarme tanto sobre programación.

Finalmente, quisiera agradecer al equipo de trabajo haber sido el más comunicativo y cohesionado en el que he trabajado para un proyecto de videojuegos, y cómo siempre hemos podido hablar sinceramente y sin tapujos ni prejuicios.

RESUMEN DEL PROYECTO

El proyecto “Monster Heroes: Time Travelling Odyssey” tiene como resultado un juego de plataformas en 2.5D para dispositivos móviles iOS y Android principalmente, pero también disponible para PC.

Este proyecto es un juego de estilo casual¹, en el cual el jugador/a tomará el control de un monstruo que debe superar niveles de plataformas variados. El juego transcurre en una nave espacial capaz de viajar en el tiempo, con gráficos simples de estilo cartoon² en el que las partidas están organizadas en secciones cortas, pero lineales, de modo que el jugador puede retomar la partida donde la dejó siempre que haya alcanzado un punto determinado. Esto se hace así para que, aún siendo un juego casual, el jugador pueda tener una sensación de evolución y progreso dentro del juego.

El juego se desarrolla en un entorno tridimensional, donde la jugabilidad se basa únicamente en la interacción en un plano en 2D. El jugador, inicialmente, controla a un monstruo por defecto, el cual podrá cambiar por otros modelos a desbloquear conforme avanza en la partida, a través de un sistema de monedas colecciónables.

El monstruo seleccionado para jugar aparece en el nivel, generalmente, por la izquierda del mismo, y su objetivo consistirá, básicamente, en atravesar el nivel de izquierda a derecha ascendiendo o descendiendo por el nivel, hasta encontrar la salida resolviendo los puzzles, superando los obstáculos y enemigos.

El monstruo dispone de un total de tres vidas para atravesar un conjunto de escenarios o, en otras palabras, sección. Cada escenario está delimitado por el punto de control o “checkpoint” de inicio y el de final. Éste último se corresponde con el “checkpoint” de

¹ Un juego casual es aquel “[...] juego sencillo, apto para todos los públicos y que se puede disfrutar de rato en rato, sin que haga falta ser constante [...] son perfectos para jugar fuera de casa, en el transporte público, en una cafetería o incluso andando por la calle.” (U-tad, 2023).

² Estilo de gráficos que imita ilustraciones o dibujos animados.

inicio del siguiente escenario. En cada escenario, el objetivo es alcanzar el “checkpoint” siguiente.

ABSTRACT

The project "Monster Heroes: Time Travelling Odyssey" results in a 2.5D platform game for iOS and Android mobile devices mainly, but also available for PC. This project is a casual-style game, in which the player controls a monster who must overcome different types of platforming levels. The game takes place in a spaceship capable of time travel, with simple cartoon-style graphics in which the games are organized in short, but linear sections, so that the player can pick up where he/she left off whenever he/she has reached a certain point. This is done so that, even being a casual game, the player can have a sense of evolution within the game.

Key words: Simple, fast, colorful, challenging.

PARTE I. PRESENTACIÓN DEL PROYECTO

El equipo

Baena Toquero, Manuel José



Senior Software Engineer en Real-Time Innovations e Indie Game Developer.

Fundador de BlackAnt Games.

Contacto profesional: manueljosebaena@gmail.com

Díaz-Estébanez Arias, Jesús



Técnico Superior en Animación 2D, 3D y Entornos Interactivos

Junior Environment Artist

Contacto profesional: jcdiazestebanez@gmail.com

Dominguez Gómez, Manuel Jesús



Junior Unity Developer y Junior Level Designer.

Estudiante The Core.

Contacto profesional: mjdomgom@gmail.com

Sánchez Sinisterra, Irene



Técnica Superior en Animación 2D, 3D y Entornos Interactivos.

Senior QA Tester y Junior Unity Developer.

Vicepresidenta de ARESOV y Responsable Técnica de la Comisión Edujoc ++.

Contacto profesional: is.sinisterra71@gmail.com

Motivación personal

Baena Toquero, Manuel José

El desarrollo de videojuegos se ha convertido en mi gran hobbie profesional. A lo largo de mi carrera he implementado y publicado videojuegos para realidad virtual móvil y he creado infinidad de prototipos. He tenido la suerte de poder presentar proyectos al equipo de dirección de sistemas de simulación de Unreal Engine Europa, así como a los directores del equipo de simulación de Unity Engine. Así mismo, he tenido la suerte de solicitar y obtener, con mi actual empresa, el Epic MegaGrant para la creación de un plugin de comunicación industrial en tiempo real basado en DDS.

Realizar este máster y este proyecto ha sido un paso más en mi especialización en motores gráficos, tanto en el apartado de desarrollo de videojuegos, mi gran pasión, como dentro de mi actual empresa.

Díaz-Estébanez Arias, Jesús

Uno de mis mayores pasatiempos desde que era un crío fueron los videojuegos, he crecido con ellos y han estado presentes en todas las etapas de mi vida desde que tengo memoria, incluso hoy día constituyen una parte importante que define lo que soy yo ahora.

Desde pequeño siempre he tenido pasión por la rama de las ciencias, en concreto las matemáticas y las físicas. Desde mi punto de vista, toda operación es un reto en sí, en el cual debes de seguir unos procedimientos exactos para hallar la respuesta válida.

Pero cuando descubrí la programación, y en concreto la que estaba centrada en los videojuegos, encontré un enorme potencial en la representación visual de las matemáticas al aplicarlas al mundo artístico.

Gracias a este máster he conseguido acceder y entender numerosas ramas técnicas y artísticas, que de otra manera no hubiera podido comprenderlas por mí mismo.

Dominguez Gómez, Manuel Jesús

Los videojuegos han formado parte de mi vida desde que mis padres me dieron mi primera GameBoy Advanced, con el juego de Pokemon Amarillo. Desde ese momento, mi vida ha cambiado en muchos aspectos, pero una de las constantes siempre ha sido mi pasión por los videojuegos. Gracias a esta pasión, me fue fácil escoger a que me quería dedicar desde pequeño, realizar estos videojuegos, esperando que mi videojuego se convirtiera en ese Pokemon Amarillo que me dieron a la edad de 5 años. Este proyecto me ha dado la oportunidad de ver cómo se realiza un videojuego y espero que esto sea mi entrada en el mercado laboral.

Sánchez Sinisterra, Irene

Los videojuegos me han acompañado a lo largo de toda mi vida. Recuerdo, de pequeña, las incontables horas que compartí con mi hermana jugando. Recuerdo, de adolescente, evadirme en ellos como forma de supervivencia a la dura vida que he sufrido. Poder participar de este proyecto es para mí la culminación de lo que siempre he deseado: programar videojuegos. Gracias a este proyecto he tenido la oportunidad de hacer una inmersión en el campo de la producción Indie, en un grupo pequeño y muy polifacético. Para mí este es el principio de un camino que espero me acompañe laboral y personalmente de por vida. Por otro lado, he querido reivindicar la presencia del género femenino en la producción de videojuegos. He querido hacer notar mi presencia como mujer y demostrar que sí, que nosotras también podemos y debemos crear.

Oportunidad

Este proyecto nos brinda, como equipo, la oportunidad de poner en práctica y mejorar las habilidades que hemos obtenido durante la realización del máster, a la vez que aplicar los conocimientos de nuestras anteriores formaciones con un enfoque distinto.

A lo largo del máster hemos adquirido conocimientos relacionados con diferentes aspectos del desarrollo de videojuegos y relacionados con distintos ámbitos del desarrollo, incluyendo gestión de animaciones, programación, inteligencia artificial, etc. Sin embargo, todos estos conocimientos se han adquirido de forma aislada y no como un conjunto global. En este proyecto, hemos podido aunar todos estos conocimientos para crear un proyecto que nos permita evaluar esos conocimientos adquiridos y asimilarlos de forma práctica para aplicarlos en el futuro en nuestra carrera profesional.

Así mismo, desde el primer momento en que planteamos este trabajo de fin de máster, lo consideramos como una oportunidad de crear un producto de valor que, incluso después del máster, pudiéramos seguir desarrollando, ampliando y, si finalmente lo consideramos oportuno, publicarlo, pudiendo compartir este producto con el público objetivo para el que lo hemos creado.

Por tanto, vemos este trabajo, no sólo como un producto académico, sino como un producto que algún día pueda llegar a estar disponible para el público general.

Objetivos

O.G.1 Mejorar y asimilar nuestros conocimientos profesionales relacionados con el desarrollo de videojuegos

 O.E.1.1 Mejorar nuestros conocimientos de programación

 O.E.1.2 Aprender sobre el funcionamiento y aplicación de técnicas de Inteligencia Artificial avanzadas en Unity

O.G.2 Crear un proyecto para nuestro portfolio

 O.E.2.1 Crear un proyecto válido y funcional que sirva como base para nuestro portfolio

 O.E.2.2 Disponer de un proyecto que nos permita demostrar nuestras habilidades en posibles entrevistas de trabajo

O.G.3 Crear un proyecto que nos permita finalizar de forma satisfactoria nuestros estudios de máster.

Plan de desarrollo

La organización del trabajo ha seguido una filosofía SCRUM modificada en base a las posibilidades y capacidades del equipo. Un modelo SCRUM puro habría resultado excesivo dada la naturaleza del proyecto, por lo que se optó por simplificarlo con sprints y reuniones semanales.

Siguiendo este modelo modificado de SCRUM, cada sprint comienza un lunes y termina con el análisis de resultados el lunes siguiente, permitiendo así alcanzar los diferentes hitos de desarrollo planteados para el proyecto.

Cada sprint quedó por tanto dividido en tres partes: planificación, implementación y análisis de resultados.

- **Planificación:** La planificación del sprint consistía en un análisis de las tareas pendientes, la planificación de objetivos semanales y la constitución de las tareas que debían ser implementadas a lo largo del sprint para alcanzar el objetivo del sprint.

Como resultado de la fase de planificación, cada miembro del equipo seleccionaba las tareas que realizaría a lo largo de este, de modo que al implementar todas las tareas por parte de cada miembro del equipo, se cumpliría el hito marcado para dicho sprint.

- **Implementación:** Una vez se completa la planificación del sprint, se procede con la implementación de la tarea que realizará cada miembro del equipo de forma individual. Así mismo, en caso que fuera necesario, se realizan reuniones bajo demanda para solucionar problemas, dudas o aclarar conceptos específicos del desarrollo en equipo, de forma que todos los miembros del mismo trabajan con un objetivo común.

- **Análisis de resultados:** La fase final de cada sprint consistió en unificar el trabajo realizado. Para ello, se utilizó un repositorio. Al finalizar la integración, resolución de conflictos y disponer así de una versión estable, se procedía con la preparación de una compilación del juego que fuera utilizable y que cada miembro del equipo pudiera testear.

Utilizando esta compilación, se analizó las fortalezas y debilidades del proyecto y esto permitía, desde fases tempranas del proyecto, pivotar en su definición y gameplay.

Reflexión crítica

Este proyecto nos ha permitido poner en práctica los conocimientos adquiridos durante el máster, así como expandir ese conocimiento y practicar con otros third parties disponibles para el motor gráfico Unity.

Así mismo, para algunos miembros del proyecto nos ha permitido adentrarnos en el desarrollo de videojuegos Indie. Este proyecto supone una primera entrada en este mercado y la posibilidad de conocer lo que supone experimentar con el desarrollo de videojuegos al mismo tiempo que trabajamos en otras empresas.

A la hora de evaluar los resultados, no estamos de acuerdo con el resultado final obtenido. Algunos miembros del equipo consideran que el juego queda relativamente lejos del planteamiento inicial, algo totalmente razonable teniendo en cuenta el poco tiempo que se ha tenido para su desarrollo, mientras que otros miembros del equipo consideran que el resultado ha resultado satisfactorio.

Durante la realización del proyecto nos encontramos diversas dificultades que tuvimos que ir superando. A continuación se citan algunas de las principales dificultades a las que nos hemos enfrentado:

- Dificultad para sincronizar el equipo: Las dificultades personales a las que se enfrentó cada miembro del equipo hicieron que, en determinados momentos de la realización del máster, la carga de trabajo quedase ligeramente desbalanceada, fuera difícil integrar el trabajo realizado al no estar presentes debido a condiciones personales, etc.

Esto podría ser perfectamente extrapolable a las situaciones que se encuentra un equipo de desarrollo Indie, puesto que, por lo general, muchos desarrolladores Indie tienen que compaginar trabajo, vida familiar y situación personal con el desarrollo de videojuegos.

- Falta de conocimientos del equipo en campos específicos: En determinados momentos se tuvieron que realizar tareas que, siendo del ámbito del desarrollo

de videojuegos, no pertenecían a los estudios propios de este máster.

En algunos de estos momentos, no se disponía de conocimiento suficiente para garantizar la calidad del contenido que se introducía en el proyecto. El mayor ejemplo de esto fue la creación de animaciones específicas.

Algunos personajes en el juego no tenían un esqueleto de tipo humanoide y esto requirió crear animaciones propias o, en su defecto, buscar alternativas.

Finalmente, se realizó una investigación sobre cómo aplicar técnicas de Inverse Kinematics y cómo utilizar el sistema de animaciones de Unity para sobreponer este problema.

- Falta de conocimiento previo sobre las necesidades de los recursos artísticos en fases tempranas de desarrollo: A la hora de crear los recursos artísticos, hasta que no se integraron en las fases finales del proyecto, no se pudo valorar si realmente funcionan con el estilo gráfico del juego y cohesionan de forma correcta.

El ejemplo más claro de esto fue la creación de las puertas que cerraron la transición entre las distintas secciones de los niveles. El planteamiento original y el resultado final mostraron que había un problema de perspectiva, y esto exigió que el equipo de arte tuviera que pivotar en múltiples ocasiones para lograr un resultado coherente con el estilo gráfico del juego y la perspectiva 2.5D.

Estas son sólo algunas de las dificultades a las que nos hemos tenido que enfrentar. Por supuesto, durante todo el desarrollo, aparecieron múltiples problemas que tuvieron que ser solventados conforme se detectaron.

El futuro de este proyecto, esperamos todos los miembros del equipo, será redefinirlo, rehacer el estilo gráfico y las mecánicas de juego para convertirlo en un producto comercializable que suponga una oportunidad para todos los miembros del equipo.

PARTE II. DESARROLLO DEL VIDEOJUEGO

Esta sección de la documentación estará destinada al análisis del desarrollo realizado del videojuego, abarcando los diferentes aspectos que conformaron el desarrollo, las decisiones que se tomaron y las principales dificultades que se encontraron durante su desarrollo.

Para exponer esta parte del trabajo, se mostrará inicialmente los principales resultados del análisis realizado sobre el estado del arte del sector y estilo específico del juego que pretendíamos realizar. En este se analizan otros juegos de la misma temática, así como los principales elementos que se tomaron como referencia y motivación para implementar este videojuego.

A continuación se detallarán las principales decisiones que motivaron el Look And Feel del proyecto, abarcando tanto la estética que se pretendía lograr con el juego como los principales recursos que se desarrollaron para dotar de contenido gráfico al juego.

Posteriormente se analizarán los niveles propuestos para el prototipo de juego entregado, así como una breve explicación de qué se pretendía conseguir con cada uno de estos niveles.

A partir de este momento, comenzarán a tratarse elementos más técnicos del desarrollo, comenzando por el Gameplay del mismo y cómo se superaron ciertas limitaciones técnicas que se tuvieron que tener en cuenta para lograr que el juego pudiera ejecutarse en toda la gama de dispositivos para los que fue desarrollado.

En la siguiente sección se cubrirá la interfaz de usuario y algunos conceptos relacionados con el almacenamiento de información en el dispositivo.

Finalmente, las dos últimas secciones cubrirán dos de los aspectos técnicos más representativos del videojuego y algunas de las cuestiones de diseño más importantes que se tomaron. Por una parte, se cubrirá la creación de un sistema de eventos que

permite detectar la ocurrencia de acciones dentro del juego y reaccionar ante estas, explicando la motivación de incluir este sistema en el juego y algunos de los beneficios que nos aportó, para terminar con el sistema de Inteligencia Artificial que nos permitió crear los agentes inteligentes asociados a los personajes no jugables del juego.

Cabe destacar que, a pesar de haber cubierto algunos de los aspectos más importantes de desarrollo del juego, sólo se han cubierto los aspectos de mayor interés académico y han quedado algunos elementos sin cubrir dada la extensión del desarrollo realizado.

2.1. Análisis previo

En esta sección se mostrará algunos de los resultados obtenidos del estudio inicial del proyecto y algunas consideraciones de gran importancia que se tuvieron en cuenta a la hora de seleccionar el proyecto, diseñarlo y desarrollarlo.

2.1.1. Análisis inicial y estado del arte

Monster Heroes: Time Travelling Odyssey es un juego de plataformas de estilo casual pensado para ejecutar, principalmente, en dispositivos de gama media o baja como terminales móviles Android/iOS, aunque dada la flexibilidad que el motor gráfico Unity nos daba, también tenía sentido exportar el juego para dispositivos de mayores prestaciones, como ordenadores personales.

Dentro de este mercado, en la actualidad, existe una enorme competencia y una gran variedad de títulos que compiten por posicionarse dentro del mercado móvil. Algunos de estos títulos han alcanzado un éxito considerable y, por ello, hemos querido utilizar esos títulos como base y motivación para la implementación de este juego.

A continuación mostraremos las tres referencias principales que se utilizaron dentro del mercado de los videojuegos, que se analizaron para crear nuestro propio proyecto para obtener ideas, inspiración y motivación. Dos de estos juegos pertenecen al mercado móvil y uno a dispositivos de gama superior como consolas y PC.

Uno de los títulos que mayor influencia adquirió durante el desarrollo fue *Crossy Road Castle*³. Este título cumplía con todos los requisitos que planteamos inicialmente para nuestro juego. Se trata de un juego rápido, de plataformas, con una estética cartoon minimalista, agradable de jugar y pensado principalmente para dispositivos móviles.

³ Página principal del juego Crossy Road Castle: <https://www.crossyroadcastle.com/>

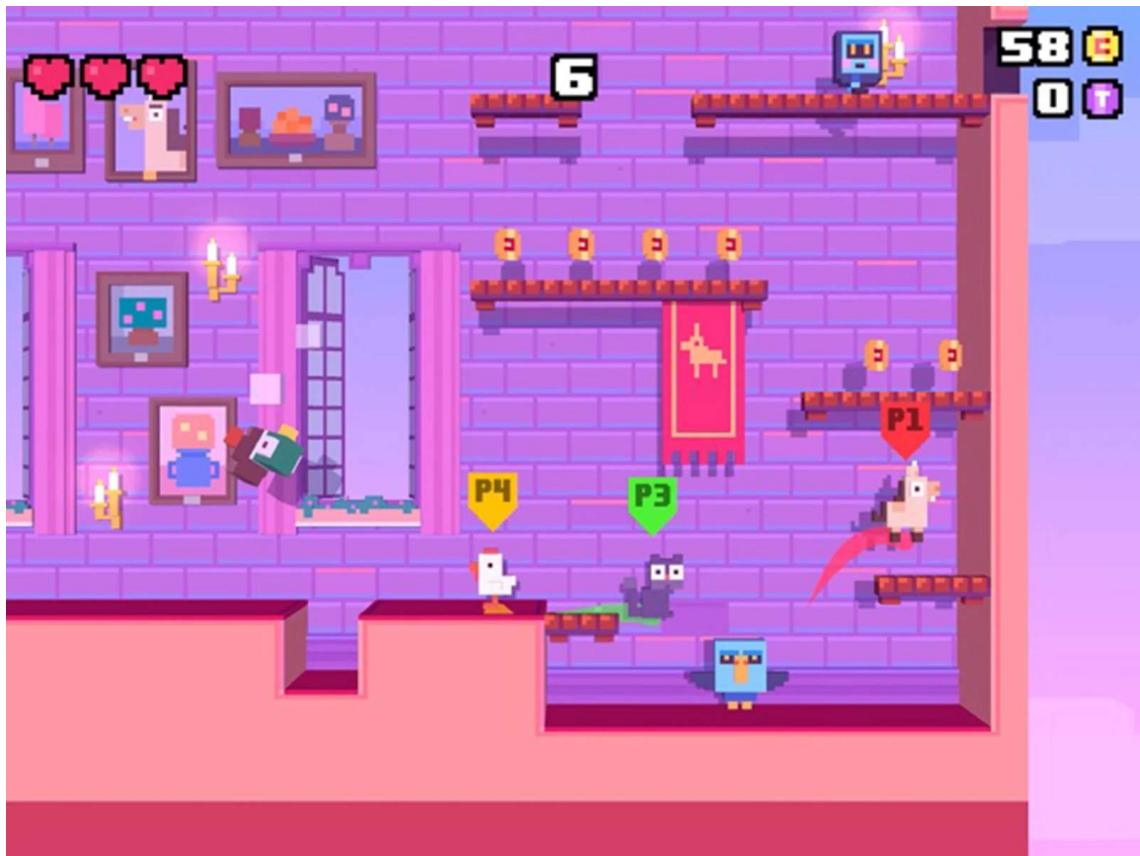


Imagen 1. Imagen del juego Crossy Road Castle obtenida de su página principal

Este juego, desarrollado por la empresa Hipster Whale y publicado por primera vez en 2020, además de cumplir con todos los estándares que habíamos definido para nuestro juego, es un ejemplo de juego bien asentado en un mercado móvil muy competitivo, ofrecido a los usuarios de iOS a través de Apple Arcade.

Se trata de un juego que ofrece una experiencia amigable y divertida, con una dificultad creciente en la que los jugadores tratan de avanzar a través de los niveles an partidas de entre 1 y 4 jugadores, pudiendo ejecutar tanto en dispositivos móviles como iPhone o iPad, como en dispositivos de televisión como AppleTV.

Así mismo, dentro de este mercado de dispositivos móviles, se consideraron también otras opciones que servían de gran motivación e inspiración. Una de los juegos que

también fueron analizados fue el título *Cookies Must Die*⁴, publicado por la empresa Rebel Twins Indie Games.

De este juego nos llamó principalmente la atención el éxito cosechado mediante la historia del juego. Este juego se basa en un humor absurdo y unos enemigos que, además de resultar poco creíbles (básicamente artículos en repostería), resultan agradables gráficamente y reaccionan ante el jugador, otorgando una sensación de inteligencia ante este que, sin dudas, nos gustaría replicar en nuestro juego.



Imagen 2. Imagen del juego Cookies Must Die obtenida de su página principal

Dentro del apartado de videojuegos que sirvieron como referencia, hay una referencia especial para dispositivos de mayores capacidades y este es *Rayman Legends*⁵, creado y distribuido por la empresa Ubisoft.

Este videojuego es, ciertamente, un referente dentro del género de plataformas. La amabilidad con la que se desarrolla el juego, la calidad de los gráficos, facilidad para jugar, historia minimalista y gráficos de estilo cartoon pero, realmente espectaculares, lo

⁴ Página principal del juego Cookies Must Die: <https://www.rebeltwins.com/cookies-must-die/>

⁵ Página principal del juego Rayman Legends: <https://www.ubisoft.com/es-es/game/rayman/legends>

convierten en un ejemplo de juego de plataformas para prácticamente cualquier desarrollador que se embarque en un desarrollo de estas características.



Imagen 3. Imagen del juego Rayman Legends obtenida de su página principal

Así mismo, aunque este juego se desarrolla principalmente en 2D, introduce componentes de 2.5D que resultan realmente espectaculares e integran con precisión y perfección dentro del juego. Un ejemplo de esto se puede observar en la finalización del nivel “Castle Rock” cuando, a pesar de que la totalidad del juego se ha realizado en un plano 2D, se cambia de plano y el jugador pasa a desplazarse en el plano de profundidad con elementos 3D rodeando al jugador.



Imagen 4. Imagen del juego Rayman Legends, nivel Castle Rock

Estos juegos, sin lugar a duda, han sido una gran motivación para seleccionar el proyecto que pretendíamos desarrollar. De todos ellos obtuvimos motivación e inspiración para la implementación de nuestro juego.

Crossy Road Castle mostraba el tipo de niveles que pretendíamos lograr, con secciones cortas pero desafiantes donde cada sección está bien delimitada, Cookies Must Die nos inspiró en la parte de humor absurdo que pretendíamos introducir en nuestro juego, donde decidimos que unos monstruos de aspecto poco creíble viajaran en el tiempo desde el futuro para rescatar a la humanidad, y Rayman Legends se utilizó como un ejemplo de excelencia dentro del campo de videojuegos en el que nos disponíamos a adentrarnos.

Aunque los videojuegos del mismo género se convirtieron por excelencia en una de las principales referencias que teníamos que tomar para el desarrollo de nuestro juego, no nos basamos únicamente en estos juegos para definir qué debía ser Monster Heroes: Time Travelling Odyssey.

Dentro del sector audiovisual encontrariamos también un gran número de referencias que nos resultan de gran interés para definir el tipo de proyecto que queríamos

desarrollar y dónde encajaría dentro del mercado, y no todas ellas pertenecen al sector de los videojuegos. De hecho, dos referencias del sector audiovisual resultaron especialmente interesantes a la hora de entender el tipo y el estilo del juego que queríamos desarrollar.

A continuación se muestran algunas de las referencias que se consideraron para definir algunos de los conceptos más importantes del juego y que no fueron cubiertas por los tres principales videojuegos del mismo sector:

- Bobobo (serie de televisión) y Los Rabbids (serie de televisión): Estas series de televisión de humor absurdo son una referencia importante puesto que en este juego se pretende disponer de un juego sin historia, pero con una alta dosis de elementos absurdos en los distintos niveles.



Imagen 5. Fotograma de la serie *Los Rabbids*

- Los Boxtrolls (película de animación): Esta referencia se utilizó principalmente para ayudar en la definición de los personajes que el jugador podrá controlar en el juego. Dado el carácter absurdo del juego, los personajes controlables por el jugador debían tener un aspecto absurdo, estos serían los denominados “Monster Box”. La idea era hacer que estos personajes fueran colecciónables y desbloqueables por parte del jugador durante la ejecución de la partida. Esto establecería al mismo tiempo, un mecanismo de adquisición/retención de jugadores, como una estrategia de monetización para el juego.



Imagen 6. Fotograma de la película *Los Boxtrolls*

- I wanna be the guy (videojuego indie): Este videojuego indie resulta especialmente interesante, puesto que es un juego que, a pesar de ser extremadamente difícil y frustrante, logró una gran popularidad.

Este juego sirvió de motivación a la hora de diseñar los niveles del juego. El juego, por su naturaleza absurda y, dadas sus mecánicas sencillas, debía resultar especialmente difícil en determinados momentos y, aunque se da la oportunidad al jugador de continuar por el punto en que había dejado la partida, frustrar al jugador en determinados momentos (sin resultar demasiado difícil) fue un elemento importante del diseño de los niveles del juego.



Imagen 7. Fotograma del videojuego *I Wanna Be The Guy*. Fuente: wikipedia⁶

⁶ Fuente del fotograma I Wanna Be The Guy: https://es.wikipedia.org/wiki/I_Wanna_Be_The_Guy

2.1.2 Hitch y Engagement

El Hitch define los elementos que pretenden garantizar que el jugador se involucre y se retenga en el proceso de vida del juego. Esta sección define los elementos que harán que el jugador no solo juegue, sino que además vuelva al juego con el tiempo, siendo un elemento principal en la retención y monetización del jugador.

Para este juego se plantearon los siguientes elementos que contribuyen al Hitch y Engagement del jugador:

- **Personajes coleccionables:** En este juego se pretende crear una base de personajes elevada, que permita al usuario colecciónarlos y elegirlos para jugar las partidas.

Los personajes coleccionables serán un elemento principal del hitch y engagement del jugador puesto que, desde el primer momento, se dará a entender al jugador que puede desbloquear personajes utilizando la moneda interna del juego que podrá obtener durante la partida.

Es por esto que un objetivo del juego es lograr que estos personajes despierten el interés y la curiosidad del jugador, mejorando el engagement y retorno del jugador.

- **Logros rápidos:** Para evitar una sensación de repetición y mejorar la satisfacción del jugador durante la partida, el diseño de los niveles debe garantizar que se tengan satisfacciones rápidas como resultado de jugar al juego, independientemente de los niveles que esté jugando.

Para lograr esto, el juego realizará un engagement en forma de checkpoints dentro de la sección de juego que esté jugando. Esto se logrará del siguiente modo:

- División del mundo de juego por niveles. Cada nivel dispondrá de un número determinado de secciones independientes.
- Al finalizar el nivel (momento en el que alcanza el final de la última sección), se notificará al jugador que ha alcanzado el fin de sección con diferentes animaciones en la UI de juego.

- Cada nivel (bloque con un determinado número de secciones) dispondrá de tres monedas especiales que el jugador podrá obtener una sola vez.
- Alcanzar esto será un “hito” dentro (checkpoint) del juego. Este momento será el momento principal para premiar al jugador y ofrecerle personajes adicionales a cambio de monedas de juego o con un descuento de monedas dentro del juego.

Este momento supondrá un hito con dos objetivos principales: Premiar al jugador con “descuentos exclusivos” y/o “personajes especiales”, y monetizar el juego utilizando dicho premio en forma de descuento exclusivo.

- **Rotura del modo de juego o gameplay principal:** Algunas secciones de juego consistirán en bosses específicos, que tendrán un minijuego diferente al gameplay habitual.

Estos minijuegos pretenden romper la línea de gameplay normal, proporcionando, momentáneamente, un gameplay totalmente diferente.

El objetivo de estos minijuegos será garantizar un interés por parte del jugador que mejore el engagement, precisamente, por despertar la curiosidad sobre cuál será el siguiente boss al que tendrá que enfrentarse.

NOTA: La rotura del modo de juego principal está planeada pero no se pudo implementar a tiempo para este prototipo.

2.1.3 Conclusiones

El análisis realizado nos llevó a obtener una serie de conclusiones de gran importancia que debíamos tener en cuenta a la hora de realizar nuestro videojuego. Estas son algunas de las más importantes que obtuvimos en esta primera fase de análisis:

- El mercado de los juegos de plataformas está muy bien asentado y los jugadores de este tipo de juegos son exigentes. Un juego con una estética y unas mecánicas aceptables no serían suficientes y debería prestarse mucha atención al detalle.
- El mercado móvil en el que encajaría nuestro juego resultaría extremadamente competitivo y requerirían de una campaña ASO⁷ bien estudiada que permitiera posicionarnos dentro de las distintas tiendas de aplicaciones móviles.
- Los elementos colecciónables y la capacidad de comprar artículos dentro del juego serían de gran importancia, por una parte, para mejorar la retención de jugadores y, por otra, para poder establecer una estrategia de monetización para el juego.
- El juego debería proponer partidas rápidas, con mecánicas sencillas y sin puzzles complejos tal y como proponen algunos juegos del mismo sector. La capacidad de mover al personaje y saltar debía ser más que suficiente para un juego de este género y estas características.
- El juego debía disponer de un componente absurdo. Esto mejorará la rápida retención de jugadores y la aceptación del juego.
- Los personajes principales, su animación, su movimiento y su apariencia serían clave para evitar el rechazo de los jugadores y poder conectar con ellos. El mejor

⁷ ASO proviene de las siglas App Store Optimization. Hace referencia a las estrategias y técnicas necesarias para el mejor posicionamiento de aplicaciones en el mercado de aplicaciones como Google Play Store o App Store.

ejemplo de esto es el juego Rayman Legends, que conecta muy rápido con el jugador.

- La muerte del personaje del jugador y los personajes no jugables, en ningún caso, debería ser traumática. En ninguna de las referencias que se han mostrado muestran una muerte seria, sino que los personajes explotan con un efecto de partículas estilo cartoon, se desintegran o simplemente desaparecen o se caen del nivel.

2.2 Estética

2.2.1 Shader⁸ de personaje y entorno

Durante la implementación del juego se requería de un efecto visual apropiado. El juego está pensado para que lo jueguen tanto niños como adultos y, por tanto, debe dar una sensación no agresiva y amigable en todo momento.

Con este objetivo, se realizó la implementación de un Shader que, originalmente, estaba pensado únicamente para los personajes, pero el resultado fué lo suficientemente satisfactorio como para utilizarlo también en los elementos de entorno.

Este Shader está basado en Cel Shading o sombreado plano, así como un efecto outline, que permite obtener una visualización del entorno exterior del personaje. Este efecto no se implementó desde cero, sino que se compró un recurso en la Unity Asset Store que se utilizó de base.

Nota: No se puede proporcionar enlace al recurso comprado en la Unity Asset Store puesto que se compró hace años y ha sido retirado de la asset store por el publicante. Sólo puede ser descargado por las personas que lo compraron antes de su retirada y no se dispone de un enlace válido para su visualización.

Este Shader permitía obtener una visualización tipos Cel Shading de los objetos de juego. Es el Shader utilizado, por ejemplo, en los elementos del entorno que no tienen outline.

⁸ Los shaders son conjuntos de instrucciones que calculan los niveles de luz, sombra y color durante la renderización.

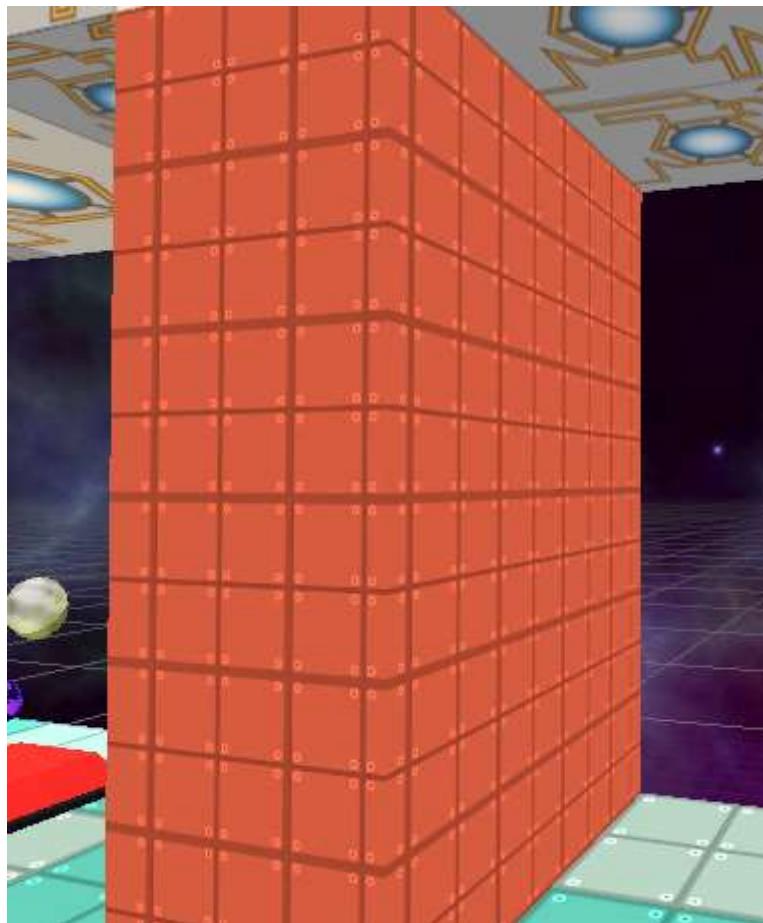


Imagen 8. Bloque con un estilo Cel Shading sin modificar

Sin embargo, este estilo no era suficiente para la correcta visualización del personaje, puesto que quedaba poco detallado y la muerte del personaje no resultaba amigable. Desactivar el modelo del personaje no ofrecía el resultado esperado.

Para lograr un efecto más espectacular, tanto en visualización como al morir el personaje, sin que resultara violento, se realizaron dos modificaciones sobre este shader:

- Efecto de disolución: Se agregó un efecto de disolución, basado en la modificación del canal alfa⁹ usando un filtro Single Noise¹⁰ y un valor de escala de grises. Un valor de transparencia determinaba el valor de gris a partir del cual se consideraba el personaje visible y cuál invisible.

También se agregó un color para delimitar el valor en el cual se considera que

⁹ Canal que controla la transparencia de la imagen.

¹⁰ Efecto que añade ruido, un granulado a la imagen.

está desapareciendo el personaje y servir de borde en la desaparición.

- Efecto de outline o dibujo del contorno del personaje: Modificando la posición de dibujado y haciendo que los bordes fueran de color totalmente negro, se logró agregar un outline al personaje. Este outline sirve para resaltar los rasgos del personaje y darle un estilo de dibujo a mano.

La siguiente imagen muestra el estado normal del personaje con outline y Cel Shading comparándolo con el material por defecto de Unity:



Imagen 9. Comparación de Cel Shading y outline (izquierda) frente al Shader por defecto de Unity Simple Lit (derecha).

La siguiente imagen muestra cómo se ve la disolución y el dibujo del contorno (outline) del personaje:

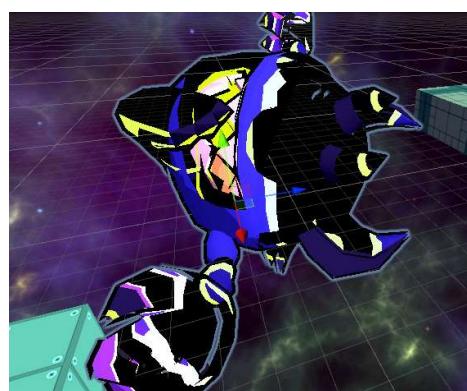


Imagen 10. Personaje principal de juego al morir, deshaciéndose tanto el outline como el material principal con un color amarillo para indicar por dónde progresó la disolución.

2.2.2 Texturas¹¹

Este apartado describe el proceso de diseño de las texturas creadas para el proyecto. Dichas texturas fueron creadas teniendo en cuenta que el juego transcurre en una nave espacial. El diseño de éstas también requería que la textura tuviera la capacidad de ser *tiled*¹².

Suelo

Tras realizar una búsqueda de referencias se decidió crear una textura usando grises y blancos en su diseño para denotar el aspecto metálico del suelo y, además, permitir un cambio de color una vez asignada la textura a un material.

La primera prueba se realizó con colores planos y sin detalles a modo de plantilla para hacerse una idea del aspecto visual que cobraría el suelo.

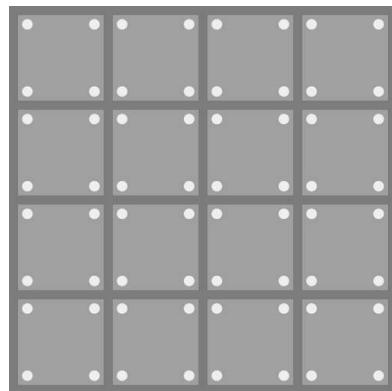


Imagen 11. Textura de prueba suelo..

¹¹ Las texturas son las imágenes “que vinculan la información de cada píxel de la imagen a la superficie de los modelos tridimensionales.” (Animum Creativity Advanced School, 2022)

¹² Una textura tileable es aquella que se puede repetir consecutivamente sin que se noten los cortes de la imagen.

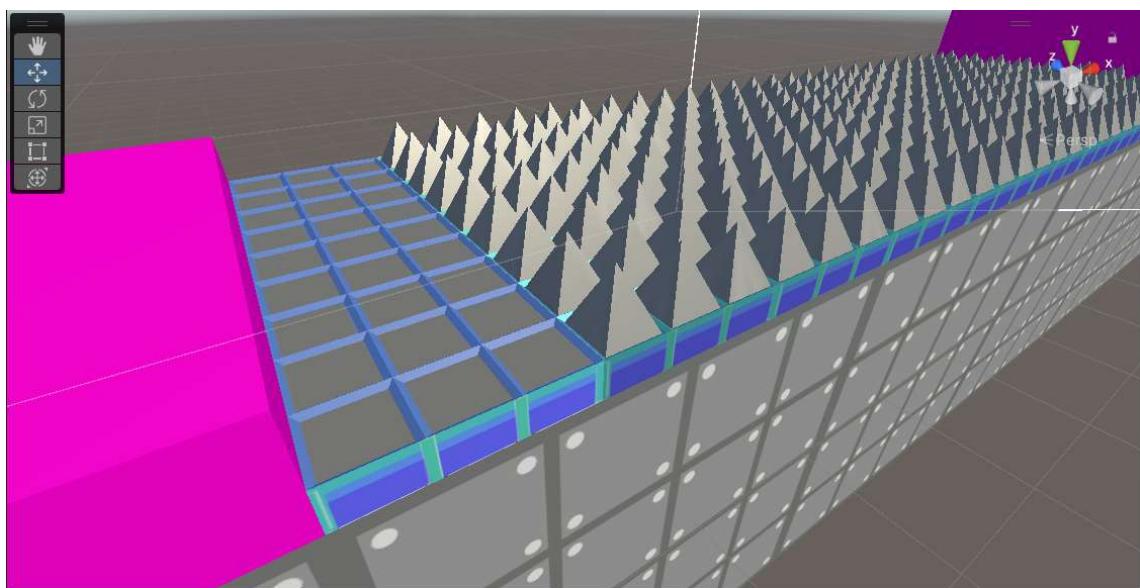


Imagen 12. Prueba textura de suelo sin aplicar shader.

Revisados los resultados de la aplicación de la textura se procedió a darle profundidad y detalle.

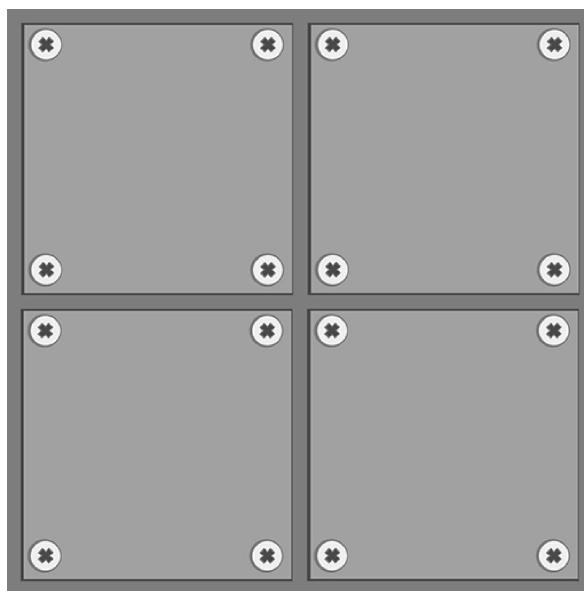


Imagen 13. Textura de suelo definitiva.

Finalmente haciendo uso del shader descrito en el apartado anterior se le dotó al suelo de un aspecto más cartoon y de reflejos. Además para que el suelo no fuera monótono y repetitivo se creó un segundo material con la misma textura en el cual se le otorga un

color azulado, se escogió el azul dada su relación con la tecnología, pudiendo así combinar ambos materiales al gusto generando patrones.

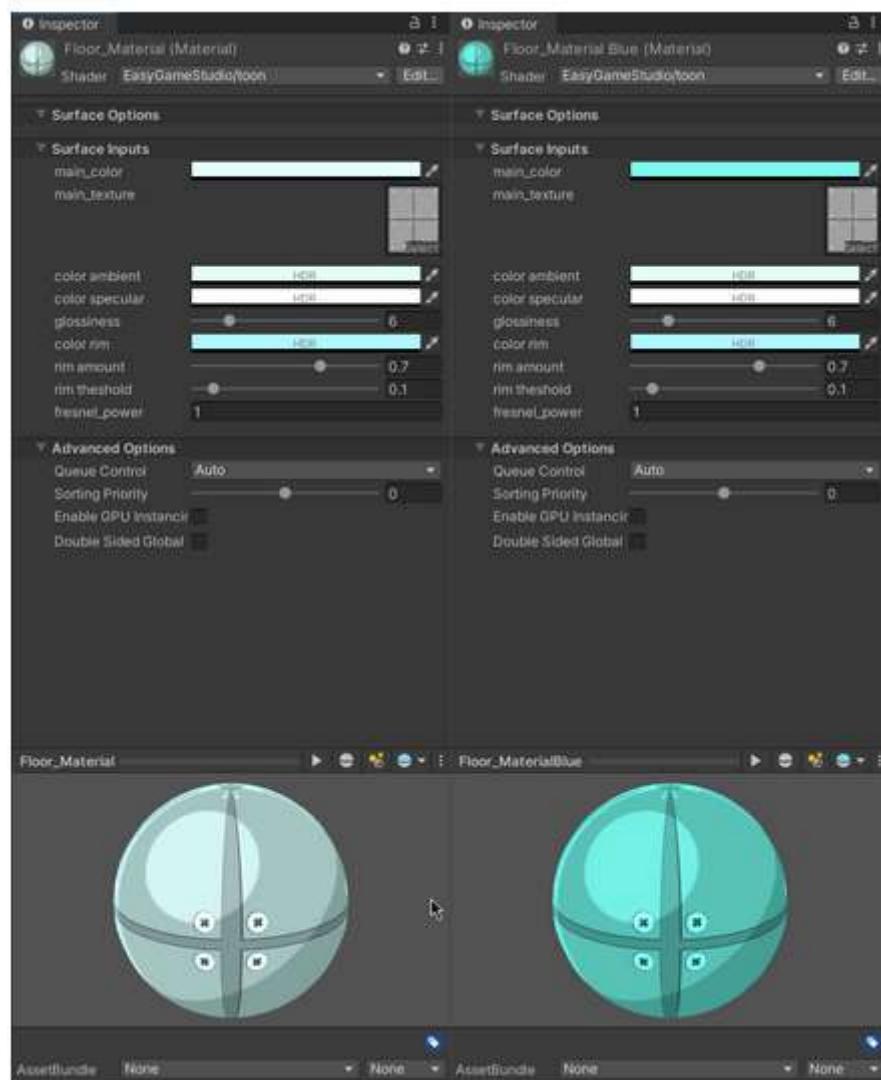


Imagen 14. Materiales suelo.

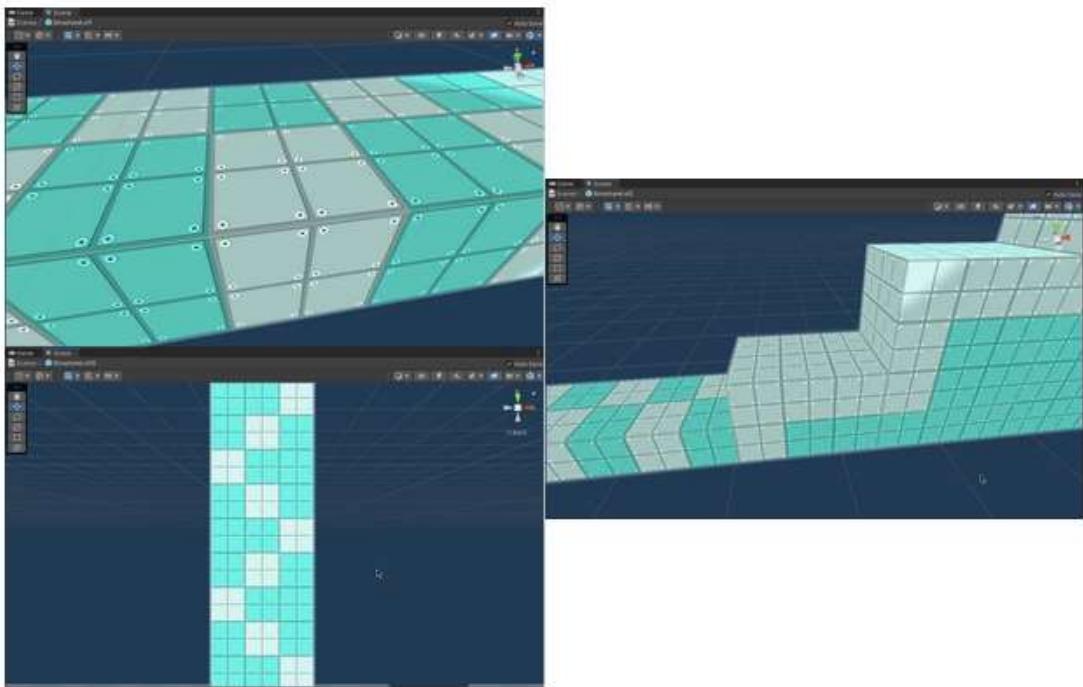


Imagen 15. Suelo con materiales aplicados.

Paredes

En el caso de la textura de paredes, al tener definido un estilo tecnológico/sci-fi hemos decidido implementar unas paredes similares a las de un laboratorio.

En concreto la intención de la textura es recrear una pared compuesta por varias chapas de metal fijadas por tornillos. Para lograr esos objetivos, en el caso de las chapas se ha optado por hacer varios rectángulos con tres tipos de colores. y en cada rectángulo se le ha dibujado unos círculos en cada esquina para simular los tornillos que fijan la chapa.

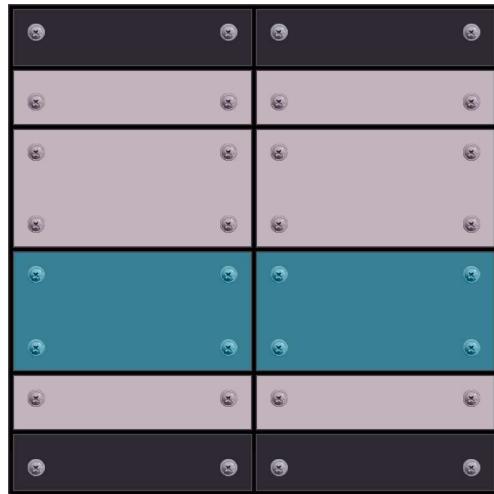


Imagen 16. Textura de pared definitiva.

Techo

En el caso del techo, para romper la monotonía al juntarlo con las texturas de las paredes y el suelo evitando así que el entorno se sintiera demasiado cuadriculado, se optó por crear una textura totalmente distinta e incluso de color cálido, que generase un contraste con los estilos anteriores.

Esta idea desembocó en una textura que proporcionaba un aspecto de cableado, de estilo futurista, de una nave espacial.

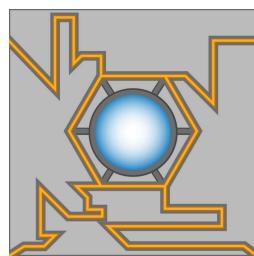


Imagen 17. Textura techo.

De la misma manera que con el suelo se crearon dos materiales distintos para añadir variedad.

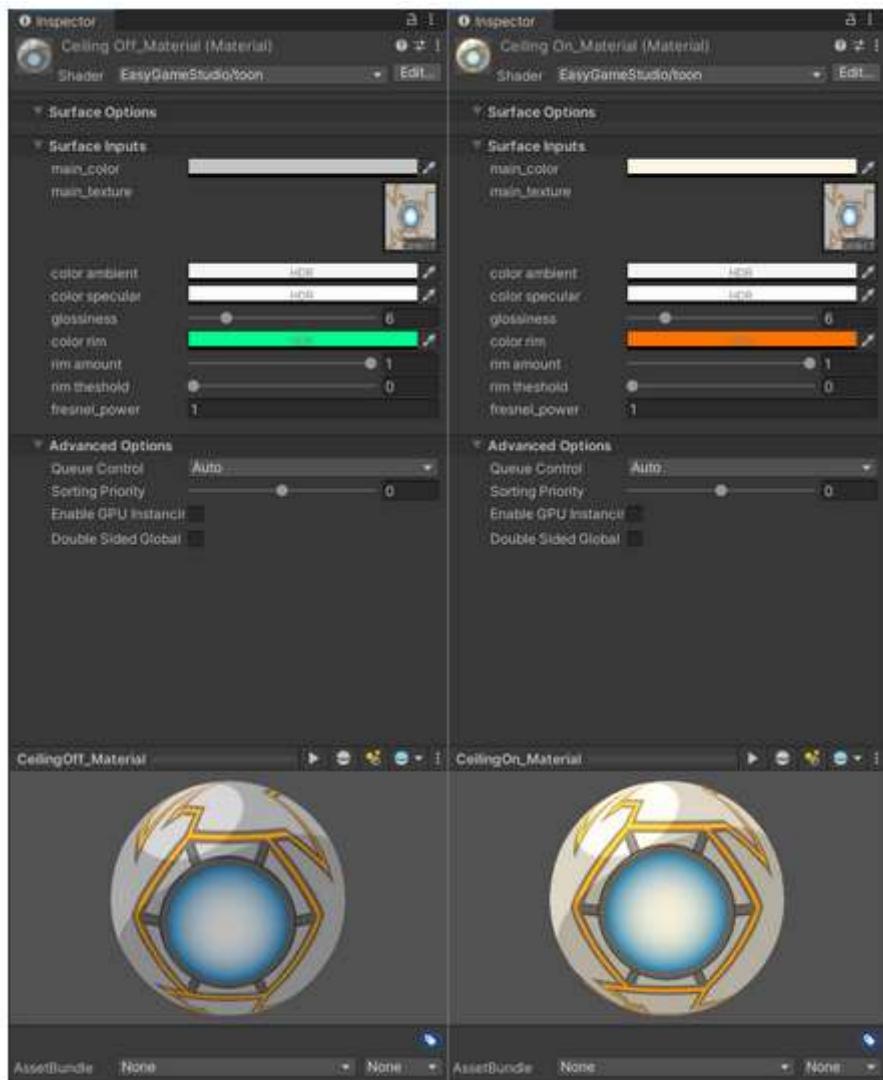


Imagen 18. Materiales techo.

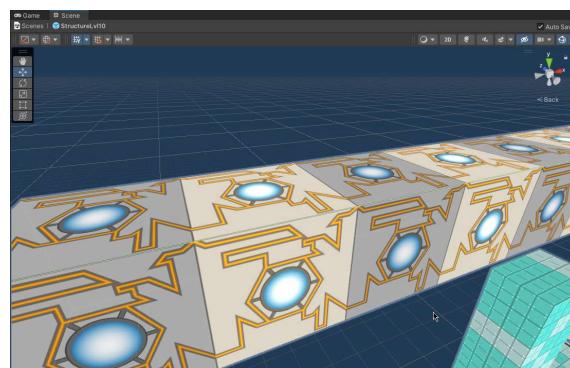


Imagen 19. Techo con materiales aplicados.

Fondos

Para los fondos hemos usado un skybox de un espacio exterior con una nebulosa.

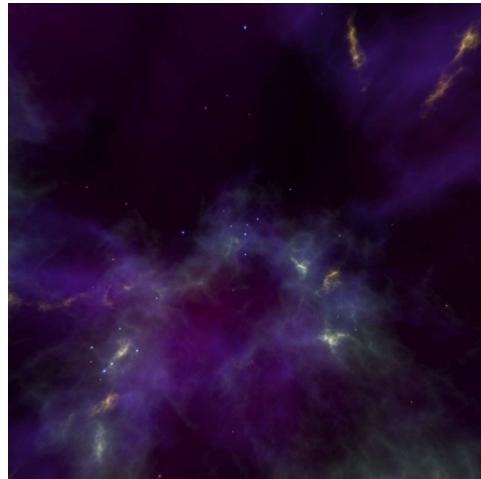


Imagen 20. Skybox de espacio exterior con nebulosa.

Entre distintas opciones se ha decidido optar por uno con colores oscuros y fríos para que no destaque en la pantalla del jugador, cuanto menos llame la atención del jugador mejor será la experiencia del jugador.

Sistema de partículas

Al finalizar un nivel, se necesitaba dar a entender al jugador que había alcanzado una victoria, y para lograr esto, se decidió crear un sistema de partículas que reforzará esa sensación.

Para las partículas de confeti se ha implementado dos sistemas de partículas, la primera son los papeles de confeti y la segunda simula el trazo que debería dc realizar el confeti.

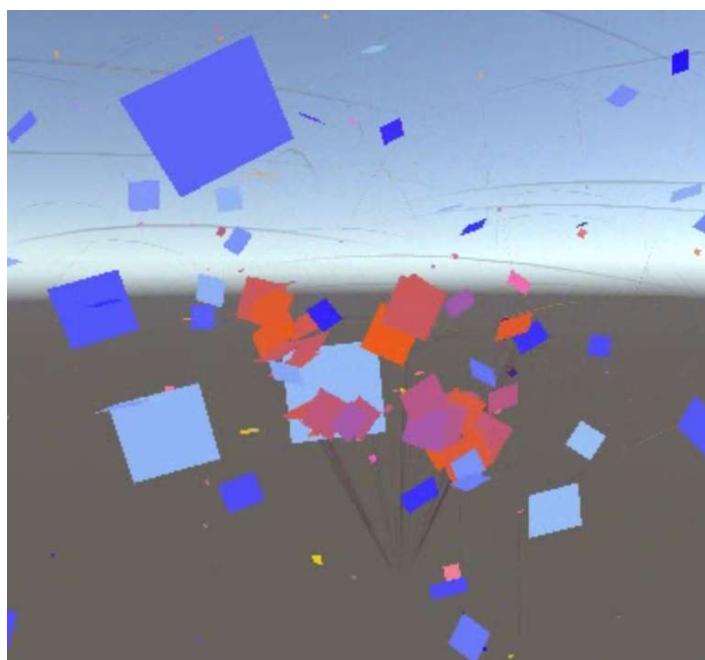


Imagen 21. sistema de partículas confeti

2.3 Coleccionables y trampas

2.3.1 Huevos

Durante la partida, el jugador podrá recoger monedas mientras explora los diferentes mundos y niveles del juego. Estas monedas se podrán utilizar posteriormente en la tienda de recursos del juego para desbloquear nuevos personajes.

Las monedas que se han creado en este juego son básicamente huevos de monstruos y podrán ser de dos tipos:

- Huevo normal: los huevos normales son la moneda más básica del videojuego. Estas monedas se encuentran en cada nivel y no desaparecen una vez se vuelva a jugar dicho nivel.

El valor de esta moneda es de 1 unidad y su objetivo sería recolectar la mayor cantidad de estas monedas para después gastarlas en la tienda a cambio de premios.

Adicionalmente, estas monedas se utilizan para marcar un camino al jugador, imitando las miguitas de pan del famoso cuento de los hermanos Grimm “Hansel y Gretel”.



Imagen 22. Huevo normal.

- Huevo dorado: Esta es la moneda especial del videojuego. Este tipo de monedas son únicas y solo se pueden obtener 1 vez, es decir, el jugador no podrá reiniciar el nivel para recolectar de nuevo este tipo de moneda.

Esta moneda se utiliza para comprar premios especiales en la tienda y en forma de logro para el jugador. Debido a la exclusividad de esta moneda, el jugador deberá de tomar riesgos o desviarse del camino del nivel para obtenerlas.

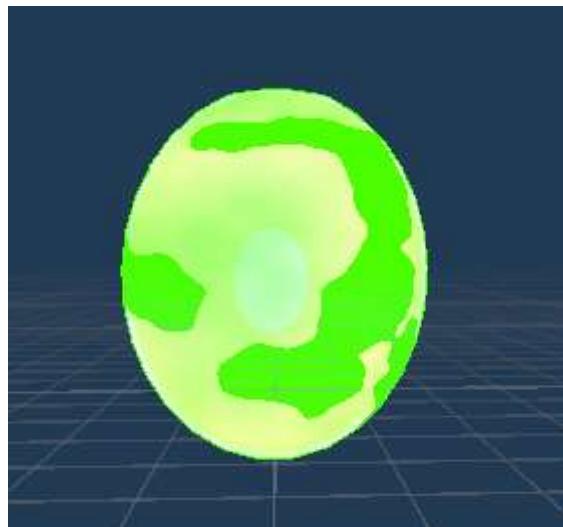


Imagen 23. Huevo dorado.

2.3.2 Trampas

Funcionalidad

Mientras intenta progresar a través de los niveles, el jugador se encontrará con diferentes obstáculos que le impedirán avanzar en su aventura. Uno de estos impedimentos son las trampas.

Las trampas son obstáculos que hará que el jugador no pueda acceder a diferentes zonas o que, al contactar con el jugador, le harán daño.

Las trampas que nos podemos encontrar en nuestra aventura son:

- Puertas: las puertas son las trampas más básicas de nuestra aventura. Estas se encuentran siempre al final de cada nivel y se activarán una vez que el jugador cambie de nivel, cerrándose y evitando así que el jugador pueda volver hacia atrás en los niveles.

Con esta trampa, instamos al jugador a tener que realizar una exploración de cada nivel, para que así no se olvide de nada que recolectar.

- Bloqueos: los bloqueos son estructuras, normalmente paredes, que el jugador se encuentra en los diferentes niveles.

La finalidad de estos bloqueos es impedir el paso del jugador o de enemigos. Los bloqueos tienen asociado un botón, el cual el jugador podrá pulsar para deshabilitar dicho bloqueo.

Así mismo, la activación de botones resulta interesante para que el jugador investigue el nivel. Un botón puede abrir un bloqueo pero, al mismo tiempo, podría habilitar otro bloqueo o trampa, impidiendo así que el jugador pueda obtener monedas o explorar una parte del nivel.

- Pinchos: los pinchos son las primeras trampas que realizan daño al jugador, restando 1 vida al total de las 3 que tiene el jugador.

Estos pinchos deberán estar pegados a alguna superficie o a alguna plataforma y tienen como finalidad impedir que el jugador se encuentre cómodo mientras está jugando los niveles. El contacto con los pinchos penalizará posibles errores del jugador restándole una vida.

Existen 2 tipos de pinchos:

- Estáticos: estos son los pinchos más comunes. Están pegados en una superficie y suelen estar en grupos extensos cubriendo zonas.
- Animados (con temporizador): variante de los pinchos estáticos. Esta variación dispone de un temporizador que indicará el momento en que la trampa deberá activarse y desactivarse.

- Bola de pinchos: la bola de pinchos es la primera trampa que es capaz de dañar al jugador y que, a su vez, es capaz de realizar movimientos, en este caso horizontales.

Estas bolas servirán para darle al jugador una sensación de persecución, y normalmente esta bola se combinará con una zona poblada de pinchos, para que el jugador sienta presión a medida que tiene que esquivarlos.

Modelado y Texturizado

Puerta

El modelo de la puerta tiene un estilo sci-fi similar a las compuertas de despresurización de las naves espaciales, además cuenta con una animación de abrir/cerrar que se activa cada vez que pasa el jugador.

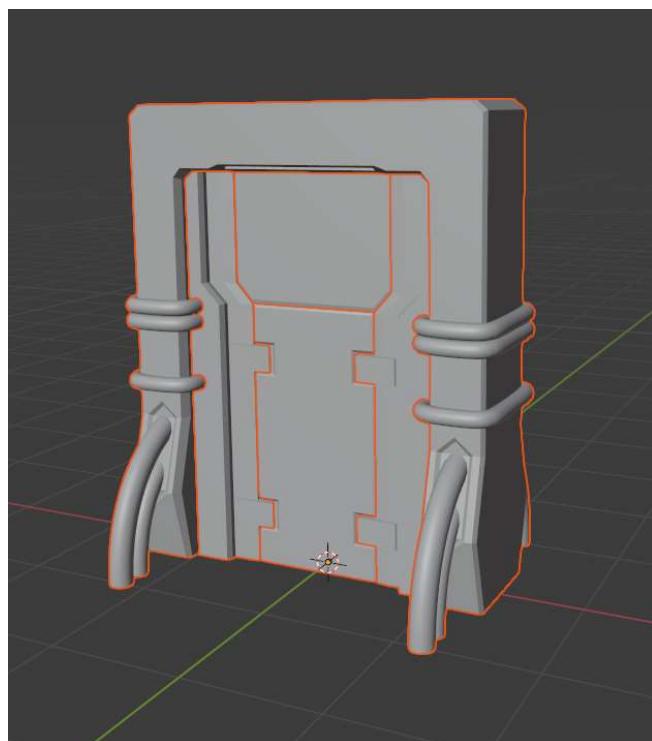


Imagen 24. Modelo de la puerta final

Bloqueos

El modelo de los bloqueos se generó dentro del propio motor de juego haciendo uso de una extensión denominada *ProBuilder*¹³. La textura y material reciclan recursos existentes en ProBuilder con el objetivo de reducir la cantidad de recursos implementados y cargados en memoria.

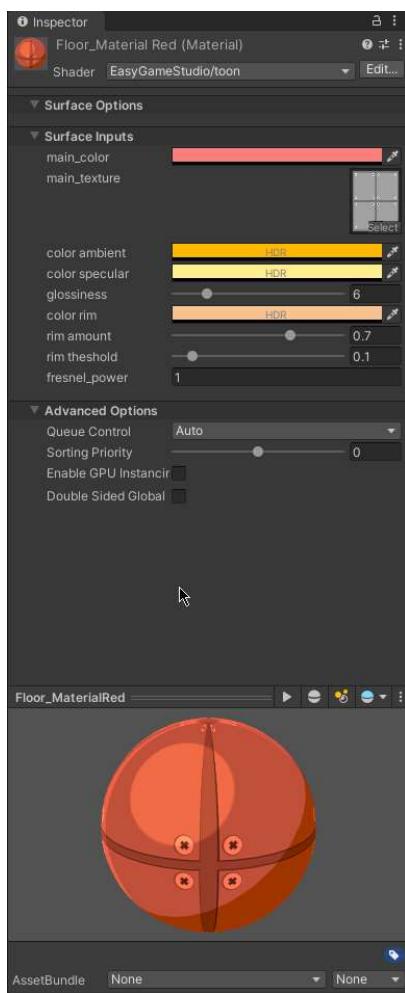


Imagen 25. Material bloqueos.

¹³ ProBuilder es una herramienta de modelado 3D que se integra dentro de Unity, permitiendo crear modelos tridimensionales dentro del propio motor gráfico. Para más información, visite la página oficial de ProBuilder: <https://unity.com/es/features/probuilder>

Pinchos

El modelaje¹⁴ de los pinchos partió de una base cuadrada a la que se le fueron añadiendo subdivisiones y vértices, siempre intentando minimizar el número de polígonos total del modelo a la vez que tener una buena topología¹⁵.

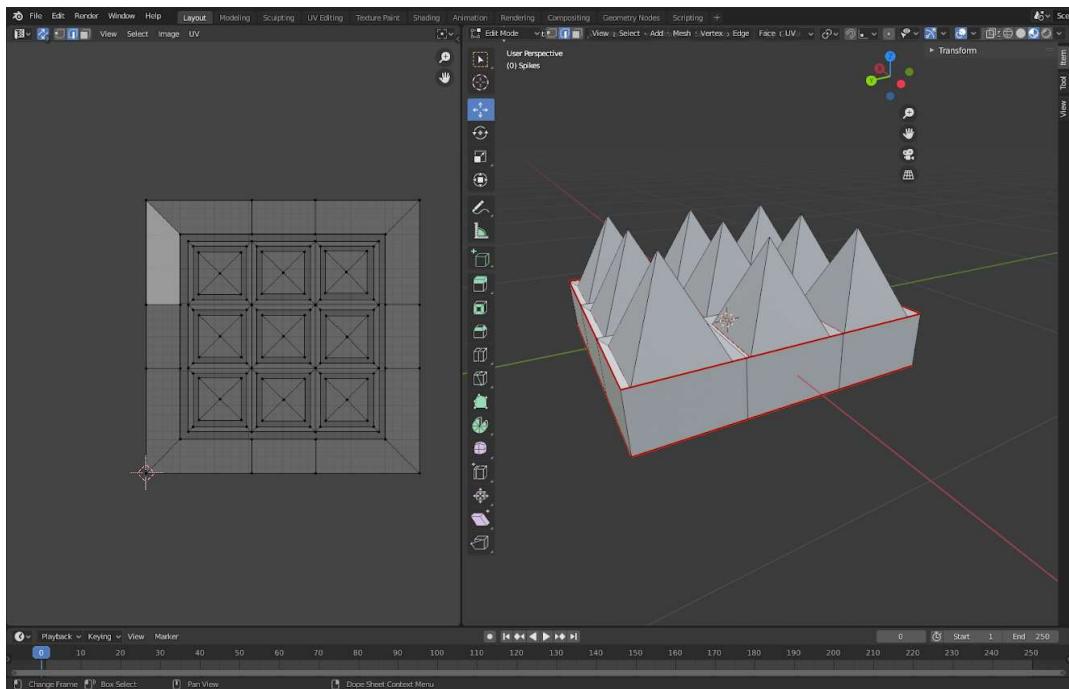


Imagen 26. Modelo 3D y UV mapping¹⁶ de los pinchos

¹⁴ Los modelos 3D se crean a partir de polígonos que se pueden añadir, quitar, dividir, modificar a través de sus vértices, aristas y caras, etc. Hasta crear la forma deseada.

¹⁵ La topología hace referencia a la disposición de los polígonos que forman el modelo 3D.

¹⁶ El UV Mapping es un despliegue de la superficie del modelo 3D en un espacio 2D, “[...] es el proceso que conecta la etapa de modelado con la de texturizado [...]” (Elsa Carrasco, 2022). Éste despliegue actúa como el envoltorio del modelo y se usa para generar las texturas del modelo.

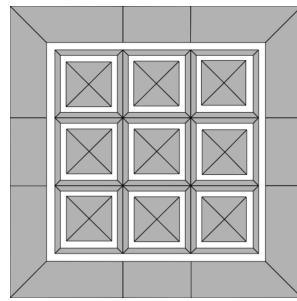


Imagen 27. Imagen exportada del UV mapping de los pinchos.

La textura que acompaña a éste modelo pasó por dos iteraciones. La primera pretendía complementar la textura de suelo y paredes sobre las que se posan los pinchos, pero se realizó una segunda iteración con colores rojizos para informar visualmente al jugador del peligro que suponen los pinchos.

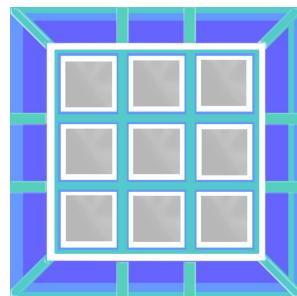


Imagen 28. Textura pinchos Iteración 1.

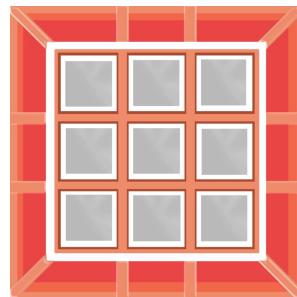


Imagen 29. Textura pinchos Iteración 2.

Para los pinchos animados se usa un intercambio de modelos que se realiza mediante una animación generada dentro del propio motor de juegos. Ésta animación se encarga

de hacer salir los pinchos de debajo de una base sin pinchos. Cuando la animación finaliza se intercambia el modelo con los pinchos escondidos por el modelo con los pinchos. Para el modelo de la base de los pinchos se eliminaron los vértices del modelo 3D que formaban los triángulos de los pinchos, se hicieron las UVs conforme al nuevo cambio y se utilizó la misma paleta de colores que para la primera Iteración de la textura de los pinchos, ya que el azul transmite calma indicando así visualmente que es seguro pasar cuando los pinchos aún no han aparecido.

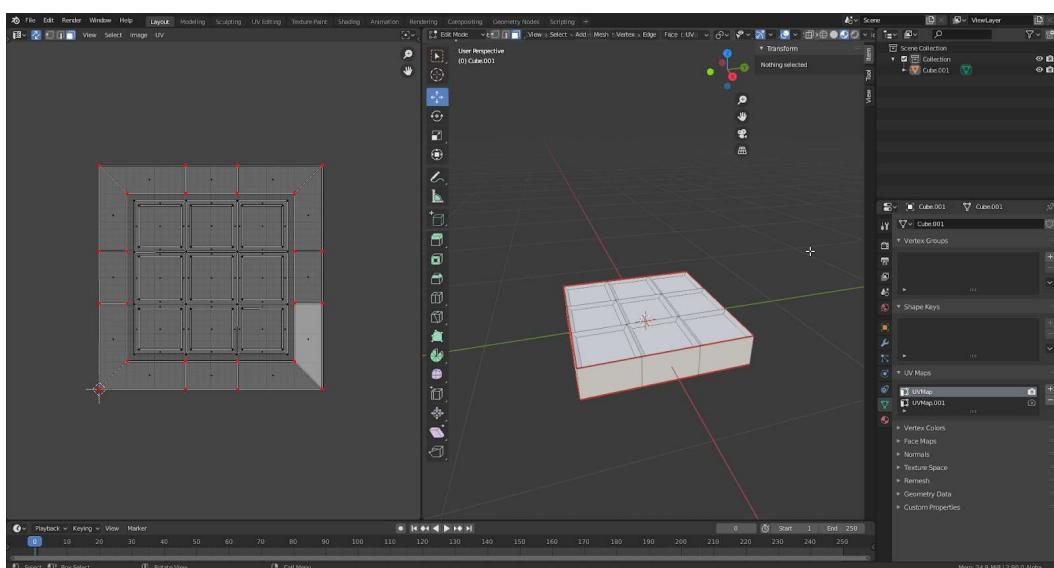


Imagen 30. Modelo 3D y UV mapping de la base de los pinchos.

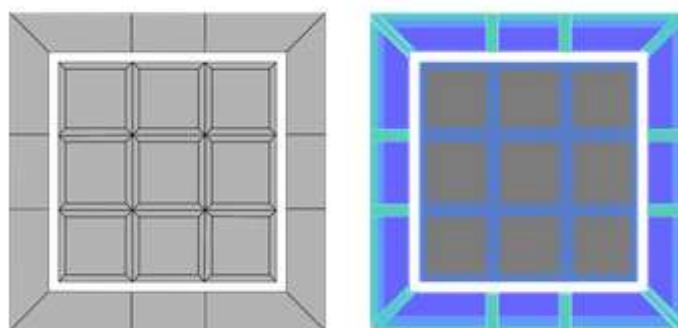


Imagen 31. Imagen exportada del UV mapping y textura de la base de pinchos.

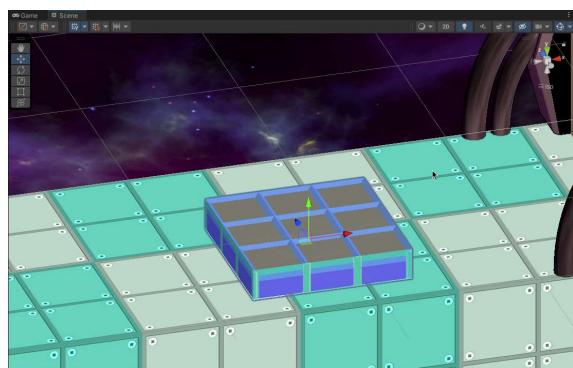


Imagen 32. Pinchos animados. Secuencia de imagen 1.

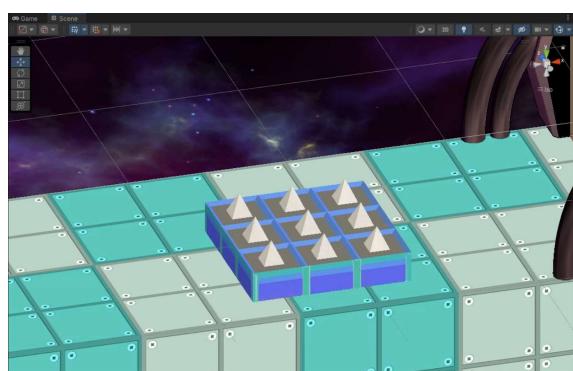


Imagen 33. Pinchos animados. Secuencia de imagen 2.

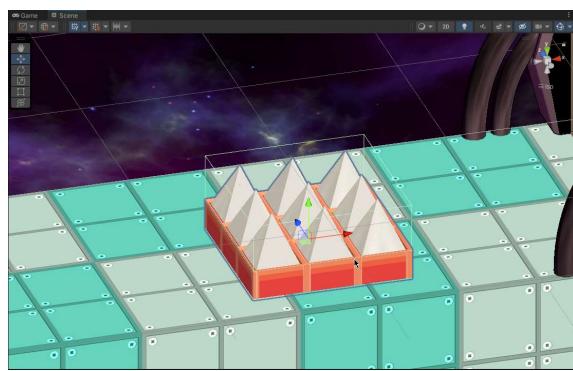


Imagen 34. Pinchos animados. Secuencia de imagen 3.

Bola de pinchos

En el juego se quiso crear un objeto de tipo trampa que pudiera moverse y que no permaneciera estática durante la partida, como hacen los pinchos estáticos o animados. Esto se logró con la creación de una bola de pinchos que puede moverse por el escenario.

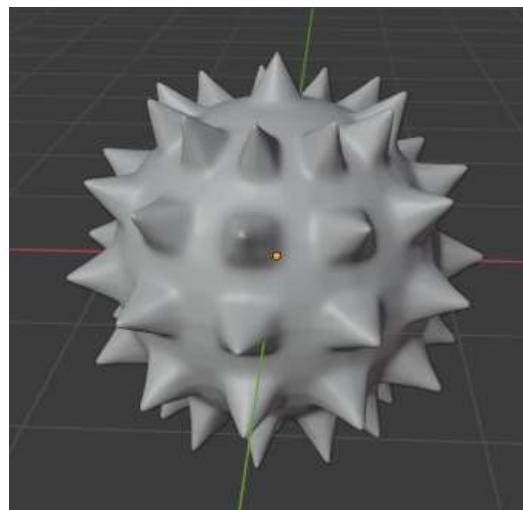


Imagen 35. Modelo de la bola pincho final

2.4 Diseño de niveles

2.4.1 Estructura

El objetivo que se tenía para Monster Heroes: Time Travelling Odyssey es que el escenario de juego se dividiera en distintos mundos, asociados a distintas épocas de la historia de la humanidad, donde los personajes controlados por el jugador acuden para salvar a la humanidad.

Así, cada mundo tiene una temática propia, y la estética de los niveles, enemigos y obstáculos están asociadas a dicha temática en función de la época en la que transcurre.

El primer mundo que nos encontramos, implementado para este trabajo de fin de máster, tiene una temática espacial. Por ello, los niveles tienen una estética futurista ambientada en el espacio. En el prototipo de videojuego entregado, se han implementado tres niveles, repartido en tres bloques, que son:

- Bloque Tutorial: El bloque tutorial es el primer bloque que se encuentra el jugador al entrar en el primer mundo del juego. Este bloque de niveles está orientado a enseñar los conceptos y mecánicas de juego básicas al jugador. Los niveles de este bloque son cortos y rápidos, pero de gran importancia, puesto que muestran al jugador las mecánicas principales de juego, asentando el conocimiento necesario para afrontar el resto de niveles.

- Bloque Medio: en este estilo de bloque transcurre la mayoría de acción del mundo.
En este bloque nos encontraremos niveles que combinan enemigos, obstáculos y plataformas, aumentando su dificultad a medida que se avanza o presentando nuevas mecánicas que surgen, adicionales a las ya presentadas en bloques anteriores (por ejemplo, con enemigos más avanzados que requieren de diferentes acciones para vencerlos).

- Bloque Especial: en el bloque especial nos encontramos aquellos niveles que rompen la mecánica de juego habitual y tienen una estructura o una mecánica diferente.

En la implementación realizada, se ha introducido un enemigo de tipo jefe para este tipo de nivel que consiste en una oleada de enemigos.

2.4.2 Niveles

En esta sección se muestran los diferentes niveles que se han implementado para la entrega del prototipo de juego de este máster. Ha de tenerse en cuenta que estos niveles tienen un enfoque académico.

En el caso del primer mundo, lo hemos ambientado en el espacio y hemos realizado once niveles en total: cinco del primer bloque, cinco del segundo bloque y un nivel de jefe final. A continuación, explicaremos cada nivel al detalle:

- **Nivel 01:** El nivel uno es el primer nivel donde aparecerá el héroe, y se tratará de una sala vacía en la que podrá interaccionar por primera vez con el movimiento básico. Esta sala también presenta la estética del mundo nuevo en el que se está empezando a jugar.

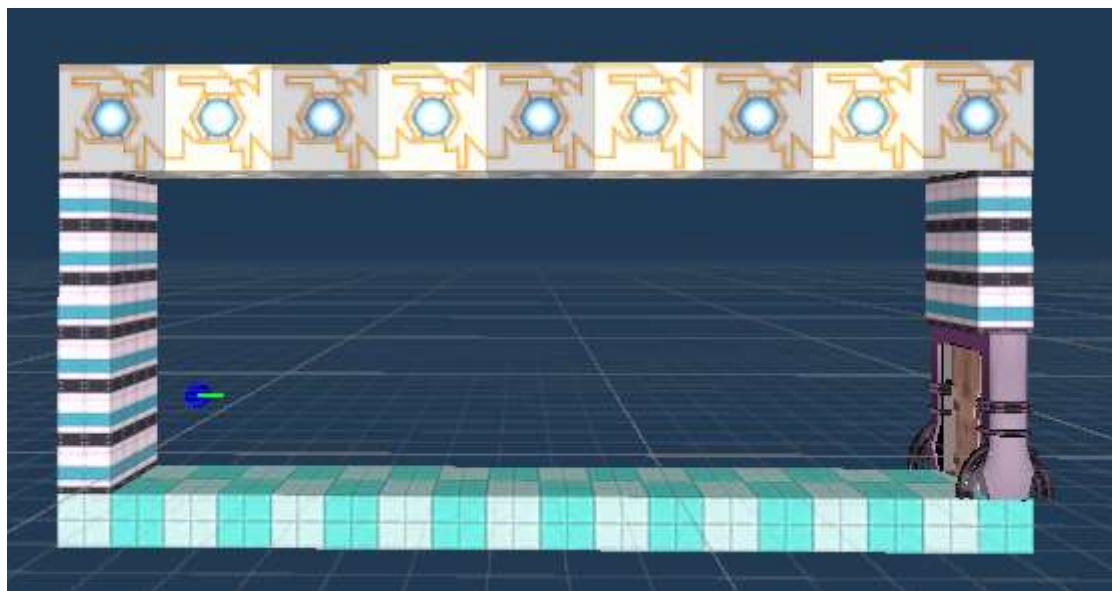


Imagen 36. Nivel 01

- **Nivel 02:** En este nivel, se enseñará al jugador la mecánica de salto, haciendo así que pueda descubrir que puede realizar un doble salto.

Además, se mostrará por primera vez las monedas, que el jugador podrá recoger.

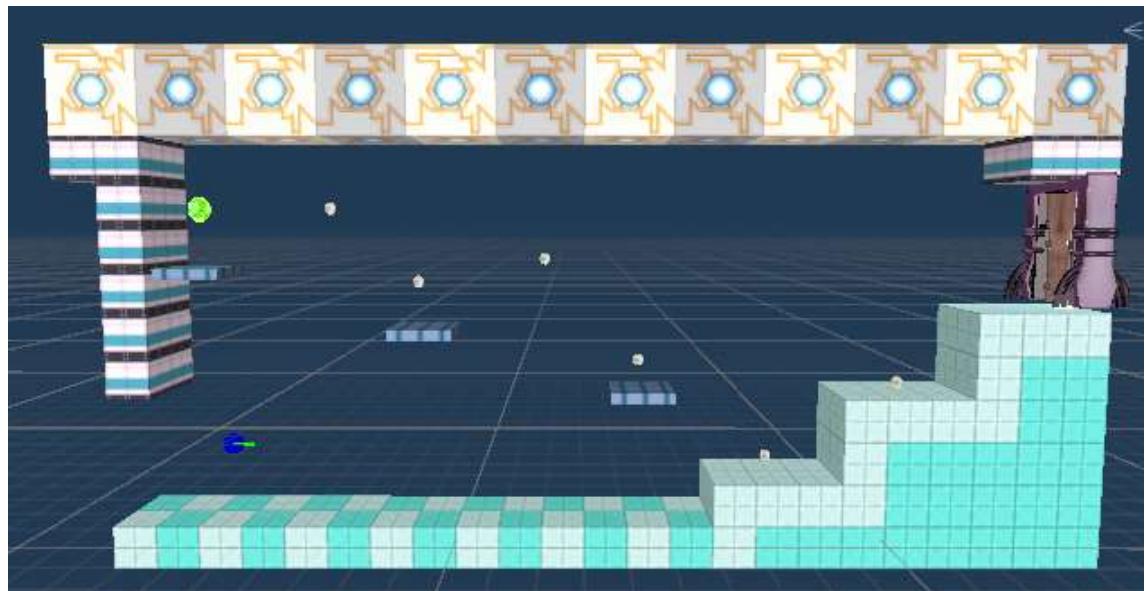


Imagen 37. Nivel 02

- **Nivel 03:** Este es el primer nivel real de plataformas. En este nivel, el jugador podrá experimentar la importancia de calcular bien el salto, puesto que en caso de fallo, perderá vidas al caer en las trampas ubicadas en la base del nivel.

Aun así, se trata de nivel de poca dificultad que también sirve para mostrar, por primera vez, la trampa de tipo pinchos estáticos y las plataformas que el jugador se encontrará en los siguientes niveles.

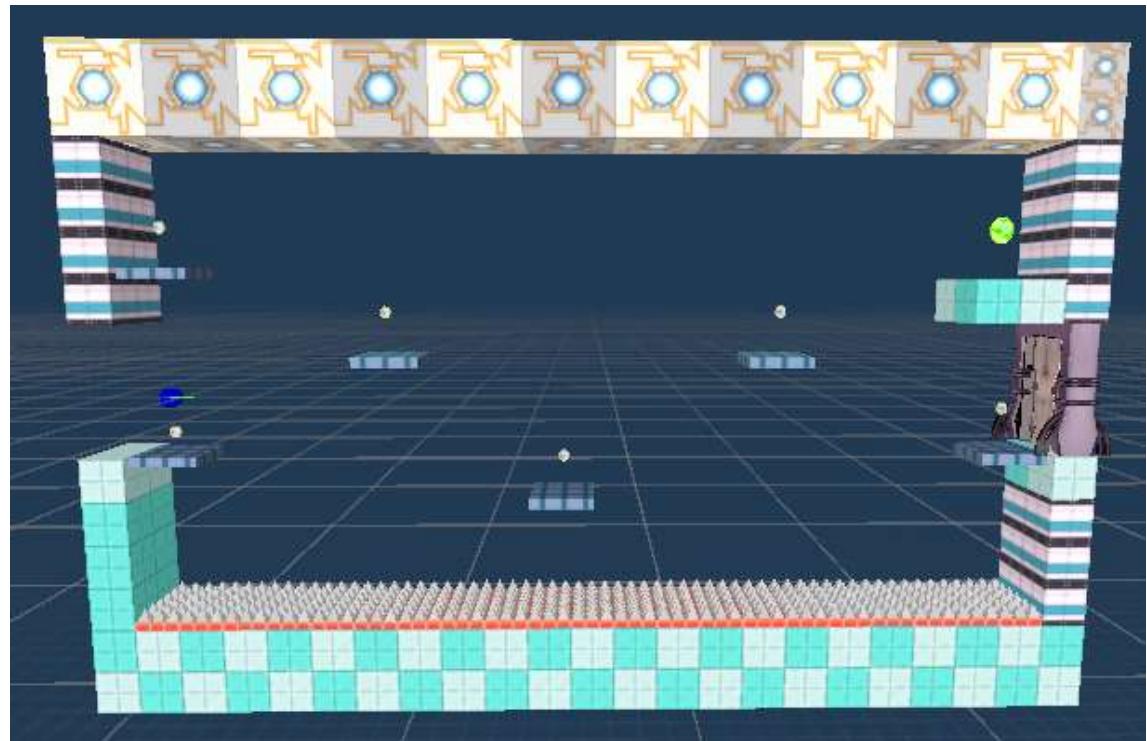


Imagen 38. Nivel 03

- **Nivel 04:** En este nivel, el jugador podrá experimentar por primera vez con bloqueos o objetos de interacción. En este caso, el jugador podrá utilizar un botón para eliminar un bloque de nivel.

También se añadirán pinchos y una pequeña zona de plataformas, donde se reforzará la sensación de peligro.

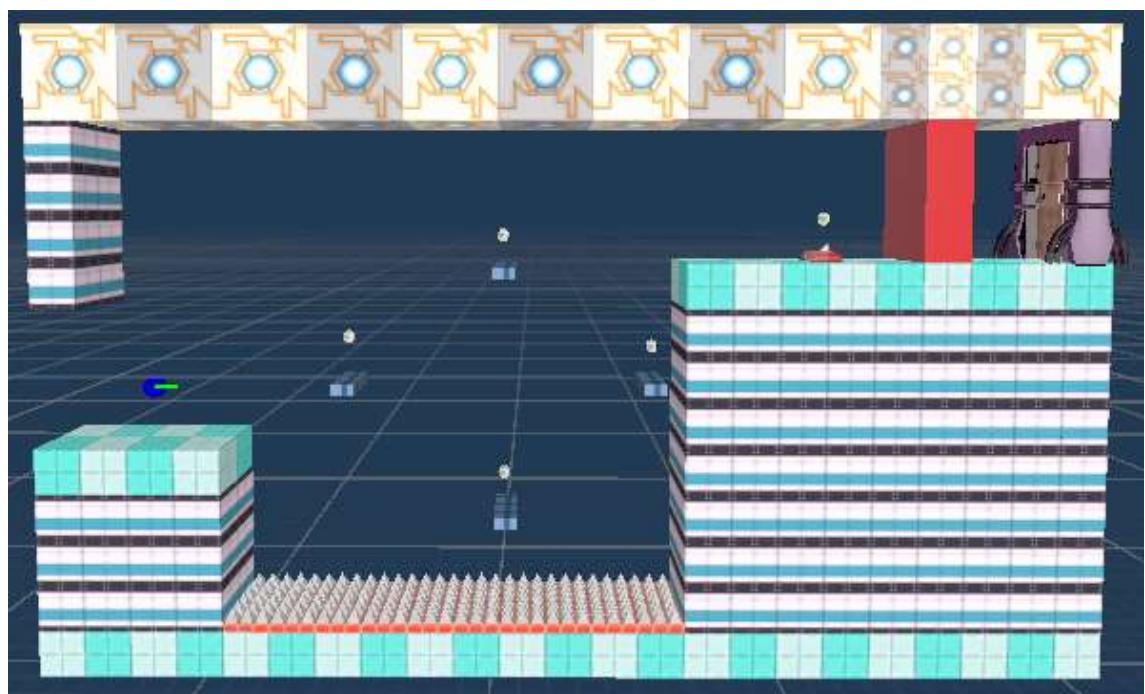


Imagen 39. Nivel 04

- **Nivel 05:** Nivel final del primer bloque, el cual aprovechamos para mostrar al jugador el enemigo más básico que se encontrara. En este nivel también hemos aprovechado para enseñar al jugador una importante lección: si quieres el premio dorado, deberás de ponerte en peligro para conseguirlo. Lección que se repetirá en los niveles del bloque dos.

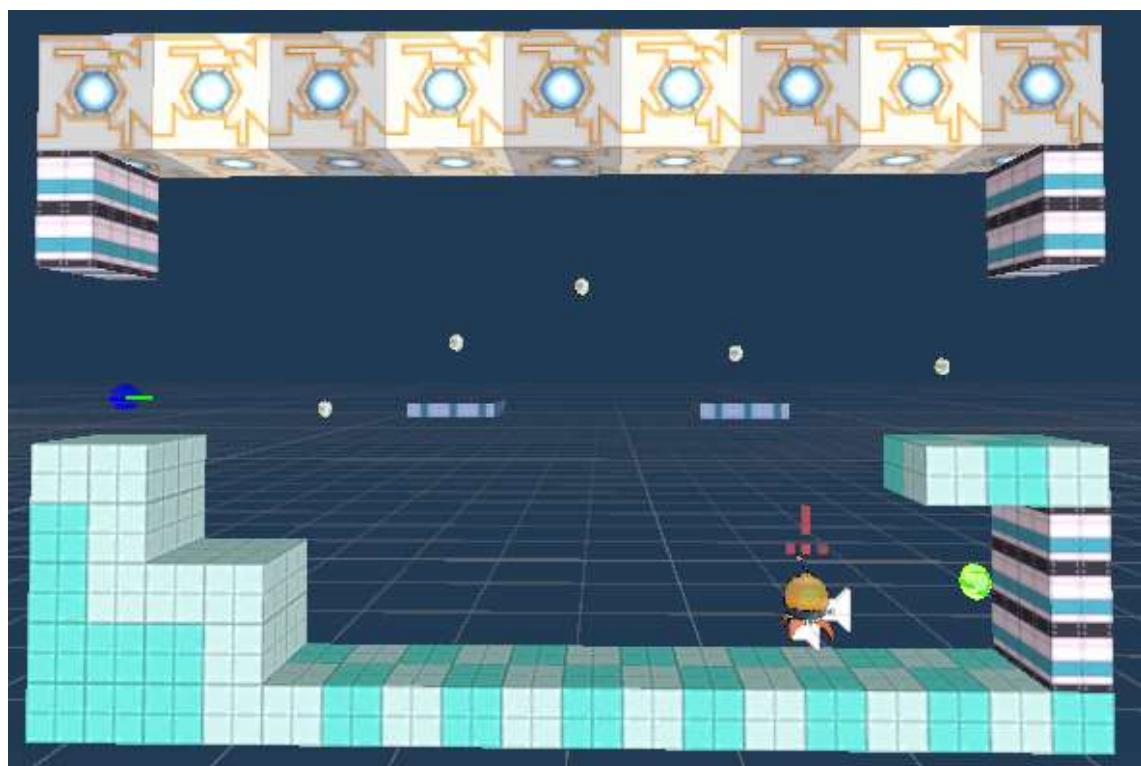


Imagen 40. Nivel 05

- **Nivel 06:** Este nivel está dividido en dos partes. La primera parte en una zona de plataformas muy pequeñas, en el cual el jugador debe de navegar con extremo cuidado si no quiere caer en los pinchos del suelo, buscando el botón que libere el bloqueo.

Una vez pulsado el botón, se le mostrará al jugador una zona de caída segura utilizando los huevos, pero este camino tiene trampa, debido a que, si el jugador decide caer recolectando los huevos normales, accionará un botón bloqueando el huevo dorado.

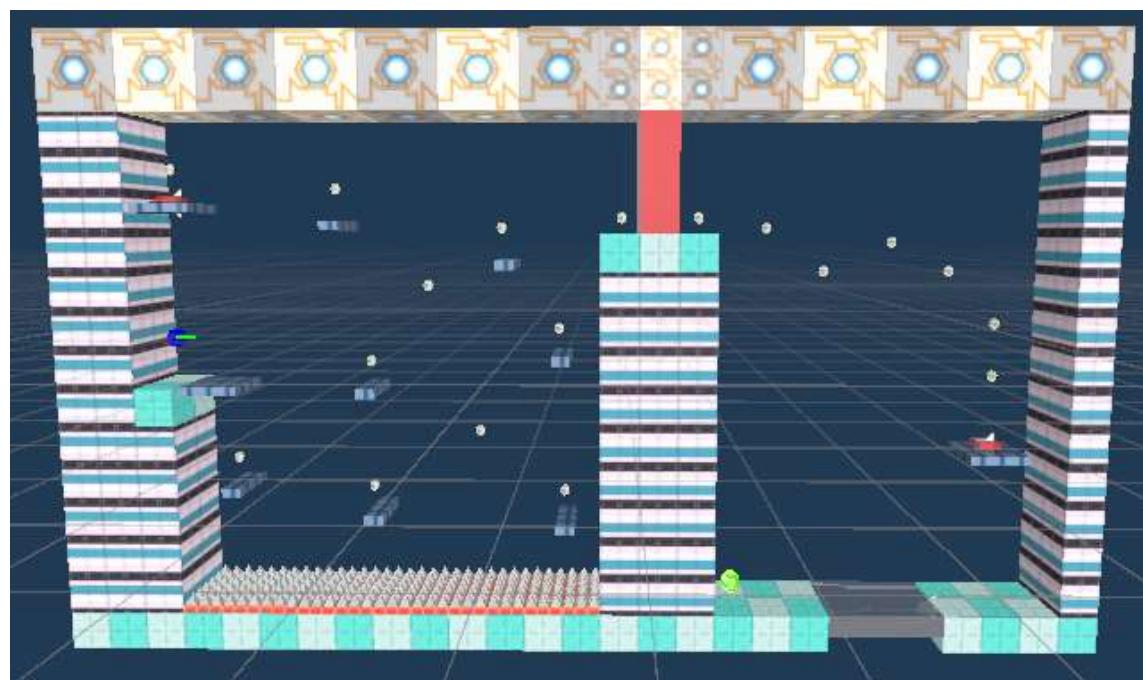


Imagen 41. Nivel 06

- **Nivel 07:** En este nivel del bloque dos presentaremos un nuevo tipo de trampa: la bola de pinchos.

En este nivel, hemos tratado de hacer que el jugador se sienta incómodo a la hora de realizar los saltos y los dobles saltos, de ahí que encontremos una mayor cantidad de pinchos, no solo en el suelo, sino también en el techo y en las paredes. También, al principio de este nivel, queremos enseñar al jugador que tiene la opción de no salir de una zona del nivel muy rápido, puesto que así perdería la oportunidad de accionar el botón que se encuentra detrás de la trampa de tipo bola de pinchos, el cual elimina el bloqueo que da acceso al huevo dorado.

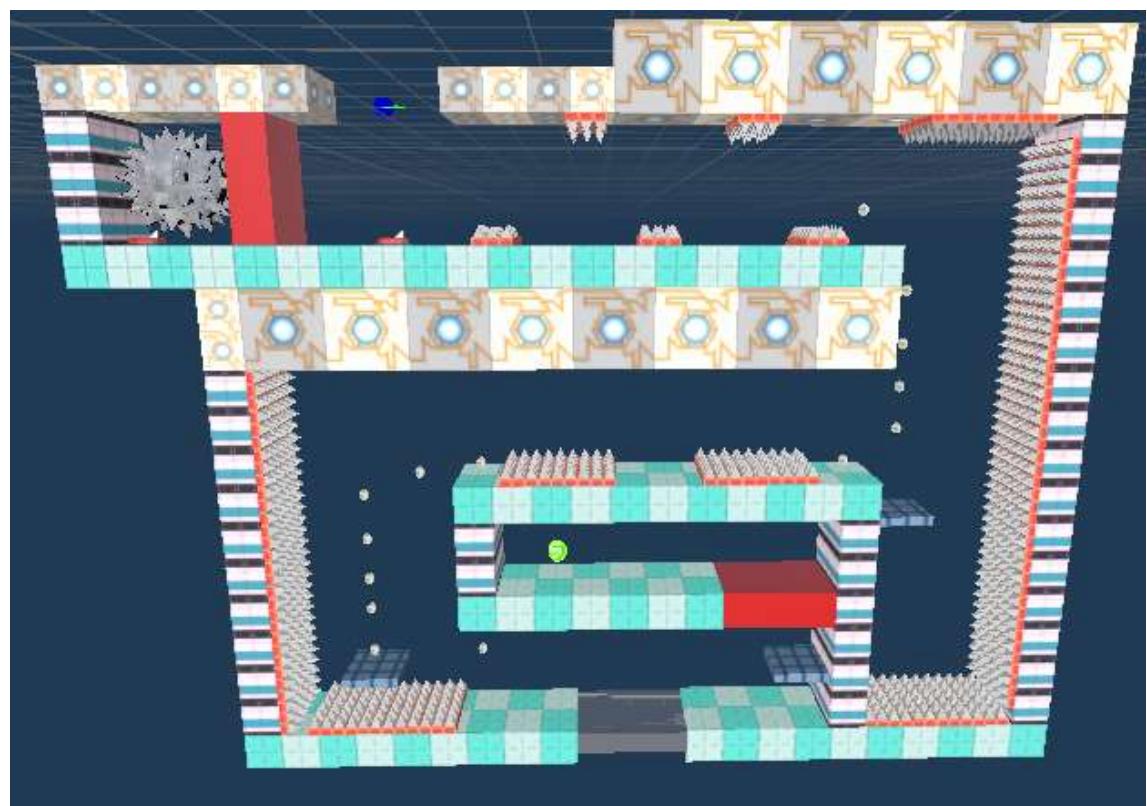


Imagen 42. Nivel 07

- **Nivel 08:** Este nivel permite jugar a un “piedra-papel-tijera” con el jugador, donde puede tomar varias decisiones. Por una parte, puede seguir el camino de monedas, como ha podido hacer de forma segura en todos los niveles anteriores. Y por otra parte, puede desviarse de dicho camino y conseguir así más premios.

Para ello, el jugador debe de dejarse caer en cualquiera de las plataformas que elija, y dependiendo de su elección, aterrizará en una moneda, unos pinchos o un botón, el cual bloquea las otras plataformas de la misma fila.

Para que este nivel surte mayor efecto en las decisiones del jugador, hemos decidido que sea uno de los últimos niveles del bloque dos.

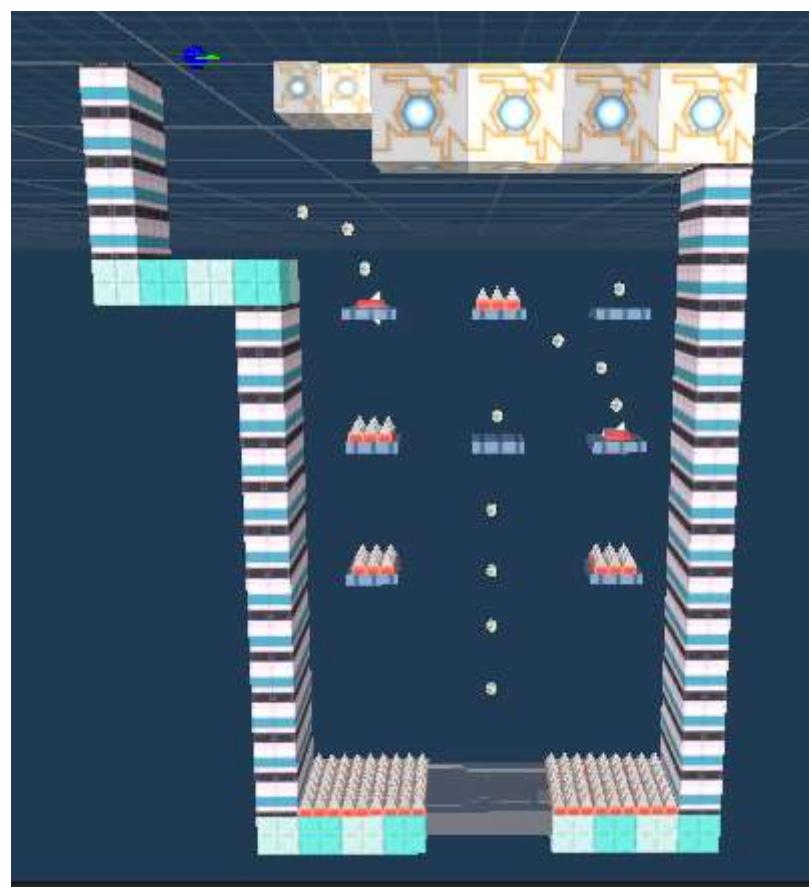


Imagen 43. Nivel 08

- **Nivel 09:** Nivel donde el jugador experimentará una subida en el nivel de dificultad. Este nivel consta de tres partes.

La primera parte introduce un nuevo enemigo, de mayor tamaño que el enemigo básico que se había presentado hasta el momento. El jugador debe liberar de forma voluntaria a este enemigo para poder proseguir en el nivel.

La segunda parte del nivel contará con una sección en la que se ubican multiples enemigos básicos que no nos perseguirán, pero nos dispararán de forma automática al vernos. El jugador debe atravesar esta sección para poder accionar un botón que da acceso a la última sección del nivel.

La tercera parte es más sencilla, y servirá para que, en caso de no haber eliminado al robot de mayor que aparece en esta sección, este nos persiga hasta cambiar de nivel y así, disponer de una zona segura antes del final.



Imagen 44. Nivel 09

- **Nivel 10:** Este nivel presenta los pinchos con temporizador. Está dividido en dos partes.

La primera parte consta de una zona de plataformas como en los niveles anteriores, pero en este caso, las plataformas tienen pinchos animados, por lo que el jugador deberá de esperar a que estos desaparezcan y pasar muy rápido por ellos mientras no están activos.

En la segunda parte se encontrarán dos pasillos llenos de pinchos con temporizador, donde el jugador deberá de esperar a que se desactiven y pasar rápido a través de ellos.

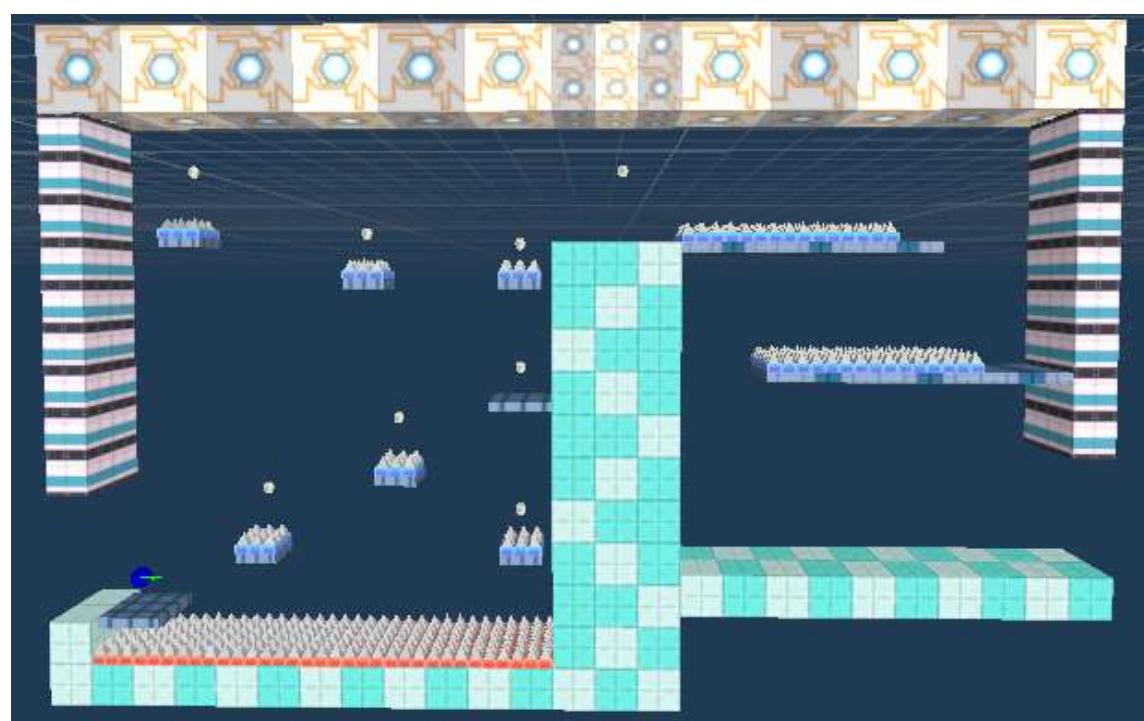


Imagen 45. Nivel 10

- **Nivel BOSS:** El nivel jefe final es el primer y único nivel del bloque tres. En él, el jugador deberá de dejarse caer a una sala grande, tras lo cual empezará una horda con enemigos del tipo pesado que se introdujeron en el nivel 9.

El objetivo de este nivel es sobrevivir esta horda, eliminando a los enemigos, hasta que se termine la oleada.

Una vez terminado, se desbloqueará la puerta de salida y el jugador podrá alcanzar el final del nivel.

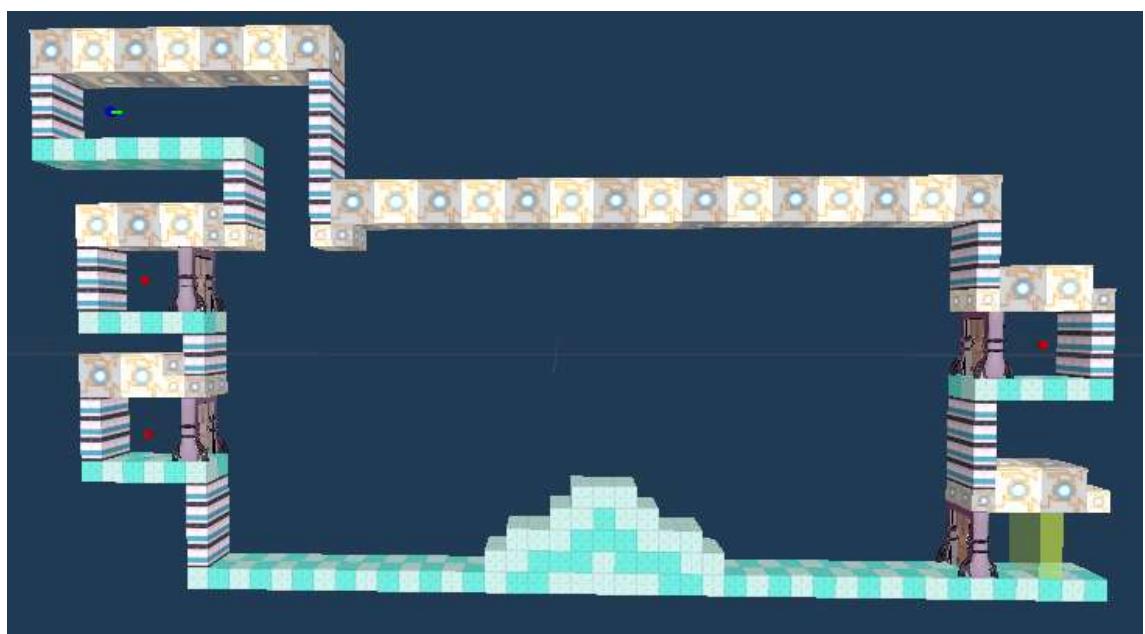


Imagen 46. Nivel Boss

2.5 Definición del Gameplay

En esta sección se describen los principales elementos relacionados con la interacción con el jugador. En esta sección se explicará cómo se interacciona con los dispositivos físicos, la interacción con el personaje controlado por el jugador y cómo el personaje de juego (el monstruo controlado por el jugador) interacciona con su entorno.

2.5.1 Sistema de interacción

Uno de los principales problemas que se abordó a la hora de implementar este proyecto se basaba en la naturaleza de los distintos dispositivos en los que debe ejecutarse.

Monster Heroes ha sido diseñado para poder ejecutar en un gran número de dispositivos de diferente naturaleza, desde dispositivos móviles de gama baja o pocas prestaciones, hasta ordenadores con una gran capacidad de cómputo. Por ello, el juego debe exhibir el mismo comportamiento en dispositivos con muy diversas capacidades técnicas y mecanismos de interacción, donde cada dispositivo podría disponer de uno o más sistemas de interacción con el usuario.

Por ejemplo, en un PC se soporta la interacción con un teclado físico, pero también es posible disponer de un periférico como un mando o controlador de consola. Por su parte, un móvil debe permitir la interacción con la pantalla, que no estaba disponible en un PC, y al mismo tiempo, podría disponer de un mando de consola conectado vía Bluetooth.

Dar soporte a los diferentes sistemas de interacción ha sido uno de los principales requisitos de diseño del juego, el cuál, se ha basado en la creación de una capa de abstracción, que actúe como mediador entre el sistema de input nativo de Unity, las acciones que deben realizarse con esta interacción, y la utilización del sistema de interacción físico dentro del juego.

El sistema de interacción, así mismo, debía garantizar que es extensible, adaptable e independiente de la lógica de juego. La inserción de un nuevo tipo de controlador, por

ejemplo, un control de realidad virtual, no debía afectar a la lógica de juego y debería suponer un impacto mínimo sobre el mismo.

Para implementar este sistema, se ha utilizado un modelo basado en capas, donde cada capa tiene una responsabilidad diferente. La siguiente imagen muestra las diferentes capas del modelo de interacción:

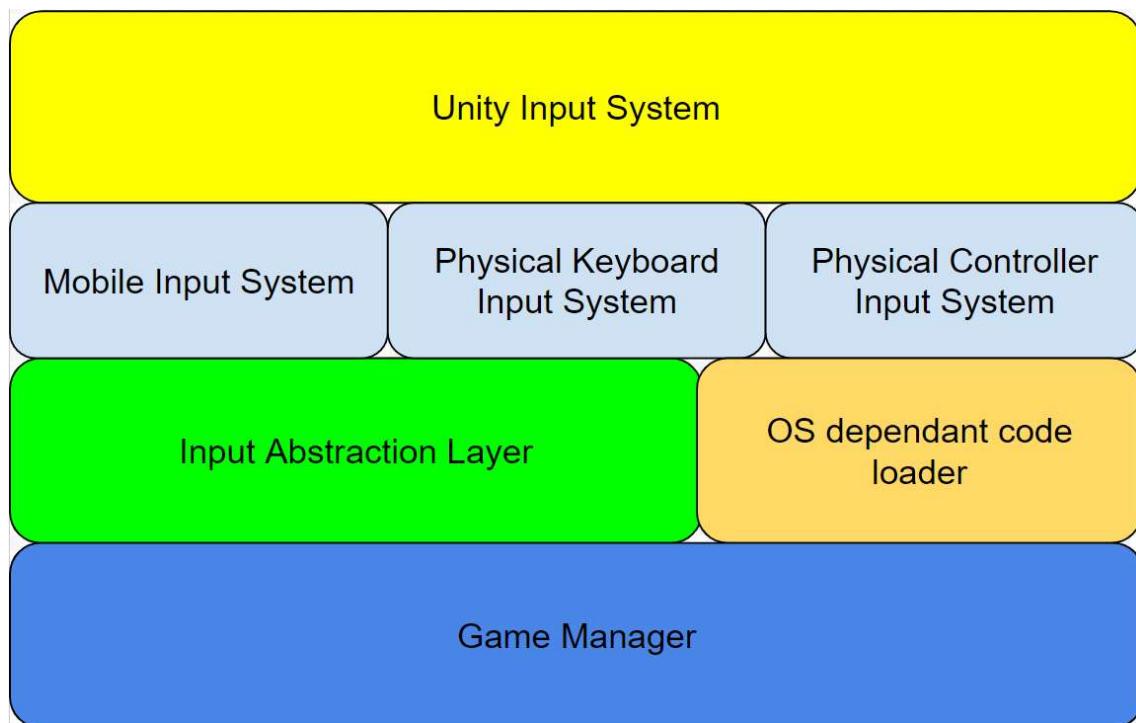


Imagen X. Diagrama de bloques del sistema de interacción del juego

A continuación, se explica la responsabilidad de cada uno de los componentes mostrados en el diagrama anterior:

- **Unity Input System:** Es la base del sistema de interacción.

El sistema de eventos de Unity se utiliza como medio para detectar la interacción del jugador con los distintos dispositivos con los que se conecta el juego.

Será, por tanto, el encargado de notificar la interacción con teclado, pantalla, periféricos de control, etc.

- **Custom Input Systems:** Abarca cualquier tipo de sistema de interacción que se pueda implementar en el juego. En el diagrama, esta capa se corresponde con las tres implementación que se han efectuado (Mobile Input System, Physical Keyboard Input System y Physical Controller Input System).

Esta capa consiste en la implementación final de los sistemas de interacción creados para este proyecto y están basados en el Unity Input System.

Estos sistemas de interacción se cargarán o no en el juego dependiendo del sistema operativo en el que se esté ejecutando.

Se han implementado tres sistemas de interacción propios y se encargan de la escucha de interacción con dispositivos móviles de pantalla táctil, dispositivos de teclado y dispositivos periféricos como controladores de videoconsolas.

Si por ejemplo, se creará un nuevo tipo de controlador (como un dispositivo VR), se implementaría dentro de esta capa de interacción y su funcionalidad no afectaría a ninguna otra capa.

- **Input Abstraction Layer:** Se trata de una capa de abstracción ubicada entre los Custom Input Systems y la lógica de juego.

Su función es traducir la interacción del jugador a eventos y acciones que sucedan durante la partida.

Un ejemplo de esto puede ser la acción de salto. En dispositivos con controladores de tipo teclado, esta acción se disparará al pulsar la barra espaciadora. Sin embargo, en dispositivos móviles, esta acción sucederá al pulsar el botón de salto en la pantalla. La Input Abstraction Layer se encargará de traducir estas dos acciones en la misma acción de salto independientemente de la procedencia de la acción.

- **OS Dependant Code:** Este código dependiente del Sistema Operativo está encargado de cargar los diferentes Custom Input System en función del dispositivo en el que se ejecuta el juego.

Este código se ejecutará o no en función del Sistema Operativo en el que se ejecuta el mismo.

Para implementar esta funcionalidad, se han utilizado macros que se incluyen o

no en el proyecto en tiempo de compilación. Por ejemplo, un dispositivo de tipo PC no incluirá el código necesario para la interacción mediante pulsación en una pantalla táctil.

- **GameManager¹⁷:** Su principal responsabilidad consiste en la recepción de las acciones notificadas por el Input Abstraction Layer.

Durante la ejecución de la partida, es un componente totalmente ajeno a la existencia de los distintos sistemas de interacción y su función será la de notificar al personaje controlado por el jugador las diferentes acciones que debe realizar.

El GameManager, además, es el encargado de realizar la carga de código dependiente del Sistema Operativo. Por tanto, esta clase es la responsable de la inicialización del sistema de interacción y, al mismo tiempo, es la receptora final de todo el sistema de interacción, aunque no es consciente del tipo de interacción con el que se está jugando.

2.5.2 Acciones del jugador

El control del personaje es uno de los componentes más importantes del juego ya que será uno de los elementos principales del Gameplay del mismo. Al tratarse de un juego casual orientado a dispositivos móviles, la interacción debe ser simple, rápida e intuitiva, garantizando que resulte fácil de entender por el jugador. Un control del personaje complejo o poco intuitivo con diferentes combinaciones de teclas o combos resultaría contraproducente dada la naturaleza casual del juego y los dispositivos para los que se lanzará..

Al tratarse de un juego 2.5D, donde el jugador sólo se puede mover en un plano 2D, pero el entorno es totalmente tridimensional, se decidió que el personaje controlado por el jugador debía ser capaz, principalmente, de moverse lateralmente y saltar.

¹⁷ Clase principal encargada del control de la carga, ejecución y gestión de la partida del jugador. Esta clase se encarga del control centralizado de todos los componentes que afectan a la ejecución de la partida.

Interacciones más complejas, como la activación de botones, palancas, realizar o recibir daño, etc. debía realizarse de forma indirecta debido a la interacción del personaje con su entorno. Esto es, al colisionar el personaje con distintos objetos del juego, provocando así una reacción en el juego sin necesidad de una intervención directa del jugador.

Este tipo de interacción permite jugar una partida completa utilizando únicamente dos dedos, lo cual contribuye de manera significativa a la experiencia de usuario de un jugador móvil que, en general, no será capaz de usar más dedos mientras sostiene el dispositivo.

Por otra parte, los jugadores que controlen el juego con un mando de consola o con un teclado de PC, estarán familiarizados a este tipo de interacción puesto que se trata de un tipo de interacción estándar en este tipo de videojuegos.

La última ventaja de disponer de un tipo de interacción sencilla como se propone es que, en todos los tipos de dispositivos que se han indicado, la interacción se realiza siempre con la misma mano: El movimiento del personaje se realizará con la mano izquierda, mientras el salto se controlará con la mano derecha, independientemente del método de interacción utilizado.

A continuación se muestra el esquema de interacción del juego en los distintos tipos de controladores:

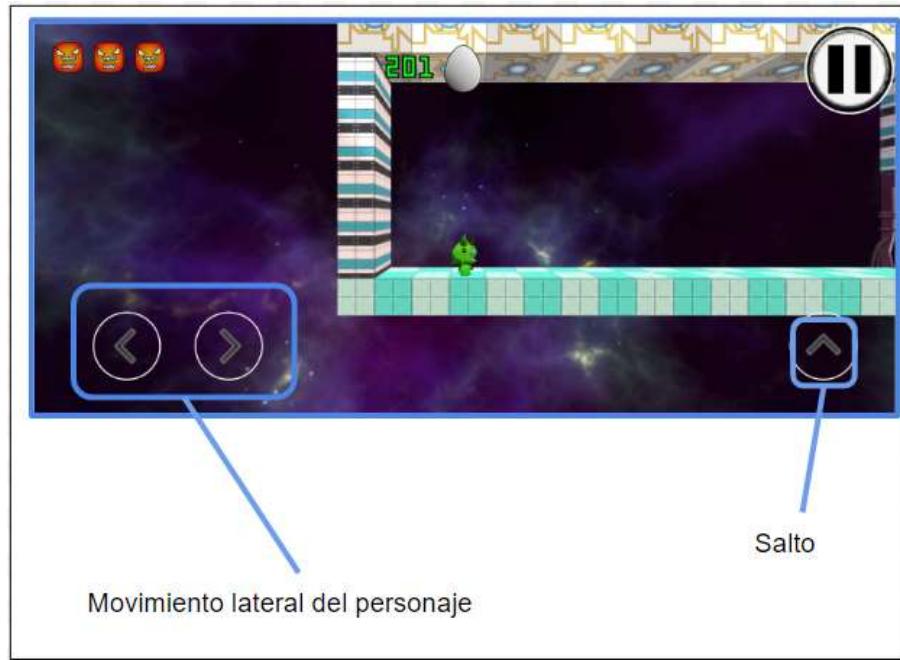


Imagen 47. Control del personaje en pantalla táctil

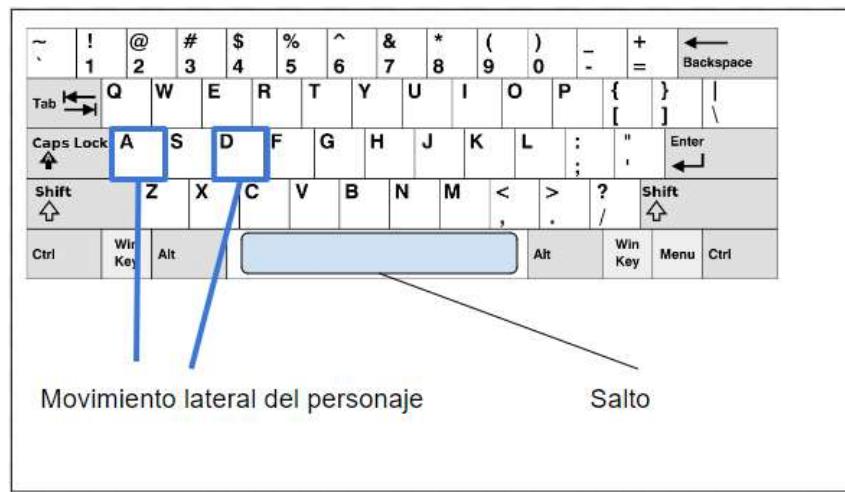


Imagen 48. Control del personaje mediante teclado



Imagen 49. Control del personaje con un controlador de consola

2.5.3 Control del personaje

El control del personajes se ha realizado utilizando el CharacterController¹⁸ provisto por Unity. Este CharacterController se caracteriza por permitir el movimiento del personaje en un entorno tridimensional sin tener que interaccionar con objetos de tipo RigidBody¹⁹.

Sin embargo, este tipo de controladores está pensado para juegos en los que el personaje no está limitado a una posición fija, sino que puede moverse por todo su entorno tridimensional. Esto no resulta suficiente teniendo en cuenta la limitación existente en

¹⁸ Sistema de control de personaje provisto por Unity. Este sistema de control garantiza que se podrá mover el personaje y detectar colisiones con el entorno sin necesidad de ciertos componentes de cálculo de físicas.

Para más información véase <https://docs.unity3d.com/ScriptReference/CharacterController.html>

¹⁹ Sistema de gestión de físicas provisto por Unity. Este objeto permite controlar la posición de un objeto de forma automática basado en su simulación física.

Para más información véase <https://docs.unity3d.com/ScriptReference/Rigidbody.html>

este proyecto. El personaje controlado por el jugador no ha de desplazarse en el eje Z de movimiento del juego.

Así mismo, el CharacterController no proporciona un sistema de control de gravedad, ni tampoco un sistema que permita distinguir tipos de objetos a la hora de determinar si el jugador está apoyado en el suelo, o si durante un salto, el jugador ha impactado con el techo y por tanto, debe rebotar y comenzar a caer.

Para implementar este sistema, se implementó un sistema de movimiento del personaje adicional al CharacterController de Unity. Este sistema debía responder tanto a la simulación física del personaje como a la interacción con el jugador e interacción del personaje con el entorno.

Para implementar este sistema se introdujeron 3 componentes en la jerarquía del GameObject asociado al personaje controlado por el jugador y, posteriormente, este mismo sistema sería exportado para el control de personajes no jugables dentro del juego:

- Detector de colisión con el suelo: realiza el cálculo matemático asociado al impacto del personaje con un elemento de tipo suelo o enemigo. Al detectar la colisión, la gravedad dejaría de afectar al personaje, deteniendo así la caída del jugador.
- Detector de colisión con el techo: encargado de detectar la colisión, durante un salto, con un objeto de tipo suelo o entorno de juego. Este cálculo se realiza únicamente si el personaje tiene una velocidad positiva en el eje Y. Es decir, si está subiendo al realizar un salto.
- Centro de masa: Se trata del centro de masa del personaje. Este objeto es utilizado para determinar la posición del personaje y será utilizado por otros elementos para interactuar con él. Por ejemplo, será el punto que tomarán como referencia los enemigos para disparar al personaje.

Utilizando todos estos elementos, el movimiento del personaje se basa en los inputs recibidos por el jugador y en el efecto de la gravedad.

El movimiento lateral del personaje se realiza realizando un suavizado del movimiento del personaje. El input recibido en este juego es binario (0 ó 1), no se reciben inputs parciales. Por ello, cuando el jugador indica que quiere desplazarse lateralmente, esta acción no se traduce en un movimiento instantáneo en esa dirección, sino que se realiza una interpolación entre los valores 0 y 1 para que se tenga una sensación de movimiento suave entre la situación estática y en movimiento del personaje mediante la utilización de la función matemática Lerp²⁰.

El movimiento en el eje vertical, en cambio, tiene tres posibles estado y estas:

- **Reposo:** El detector de colisión con el suelo está activo. Esto significa que el personaje está apoyado en el suelo y por tanto, no debe seguir cayendo. La velocidad de caída pasa a ser cero inmediatamente.
- **Caída:** Si el detector de colisión con el suelo no está activo, esto indica que el jugador está en caída y, por tanto, a la velocidad de caída actual se le suma -9.8 multiplicado por el tiempo desde la última medición de velocidad de caída.
- **Salto:** En esta caso, la velocidad de movimiento en el eje Y pasa a ser positiva, es decir, sube dentro del eje Y.

El salto se puede aplicar en dos situaciones diferentes:

- Si el jugador está apoyado en el suelo, en cuyo caso, la velocidad de caída pasa a ser la fuerza de impulso de salto.
- Si el jugador está en el aire y no ha realizado un doble salto: El personaje puede realizar un doble salto, que consiste en saltar una vez se está en el aire. Esto está permitido únicamente una vez desde la última vez que el personaje estuvo en estado de reposo.

²⁰ Lerp permite realizar una interpolación entre dos valores flotantes (números reales) en el tiempo, permitiendo así realizar un transicionado suave entre dos valores.

Para más información véase <https://docs.unity3d.com/ScriptReference/Mathf.Lerp.html>

Otro aspecto importante sobre el movimiento del jugador es que su orientación no es correcta con el movimiento que realiza. Esto se ha realizado para garantizar que el jugador puede apreciar en todo momento qué acciones está realizando el personaje (movimiento de brazos, movimiento de la boca y la cabeza, efecto de respiración, etc.) de forma correcta. Es por ello que el jugador siempre está ligeramente rotado hacia la cámara de jugador. Esto garantiza la correcta visualización del personaje.

En caso de no aplicar esta rotación, no se apreciaría su respiración, gestos faciales como el movimiento de la boca, sólo se vería uno de sus ojos y el brazo que queda más alejado de la cámara apenas quedaría visible. Orientarlo con una ligera rotación permite “humanizar” al personaje y mejora la experiencia de usuario.

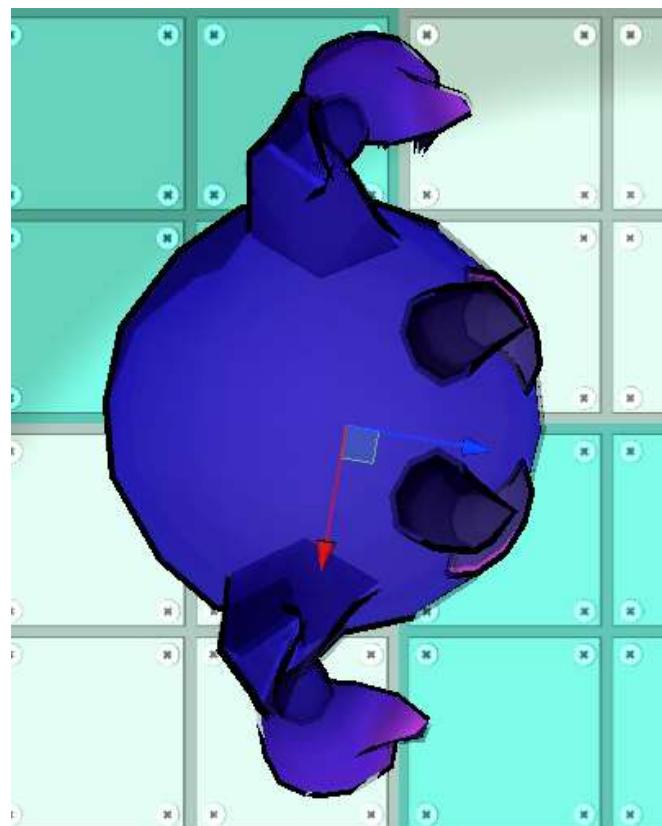


Imagen 50. Eje de orientación del personaje. Como se puede observar está ligeramente rotado frente al suelo que si tiene una orientación perfectamente alineada con el eje de movimiento del jugador.

2.5.4 Animación del personaje

El movimiento del personaje en el escenario está en todo momento enlazado con la forma en que este se anima. Por ello, el controlador del personaje está, en todo momento, asociado con el objeto Animator encargado de su movimiento.

Esta conexión se realiza a través de dos grupos de variables que gestiona la máquina de estados finitos del Animator. El primer grupo de variables controla el movimiento del personaje cuando está siendo controlado por el jugador, el segundo grupo está asociado al movimiento del jugador en la tienda.

La siguiente tabla muestra el control del personaje en movimiento controlado por el jugador:

Variables para el control del movimiento del personaje	
Variable	Descripción
movementSpeed	Valor flotante ²¹ encargado del control de la velocidad de movimiento del jugador.
jump	Trigger ²² que indica que el personaje ha saltado y por tanto debe realizar la animación de salto
onAir	Indica si el jugador se encuentra en el aire o no y, por tanto, el Animator debe pasar a un estado de caída.
deadTrigger	Trigger que indica que el jugador ha muerto y, por tanto, debe realizarse la animación de muerte del jugador.

La siguiente tabla muestra el control del personaje cuando se encuentra exhibido en la tienda:

²¹ Un valor flotante es un número decimal.

²² Una variable de tipo trigger es una variable con dos posibles valores true o false (0 ó 1) que “hace un reset de valor a false cuando se usa en una transición” (Unity Documentation, AnimatorControllerParameterType.Trigger) de una animación a otra.

Variables para el control del personaje en la tienda

Variable	Descripción
bought	Variable tipo trigger que indica si se compró al personaje. Su activación hará que el personaje efectúe una animación de celebración al ser comprado.
Selected	Variable tipo trigger que indica que el personaje ha sido seleccionado para su utilización durante el juego. Al activarse, el personaje realizará una animación de celebración con salto.
cannotBuy	Variable tipo trigger que indica que el jugador intentó comprar al personaje pero no tenía dinero. Al activarse el Animator efectuará una animación de frustración.

La siguiente imagen muestra cómo se ha realizado la conexión de todos los elementos que definen la máquina de estados finitos de animación del personaje:

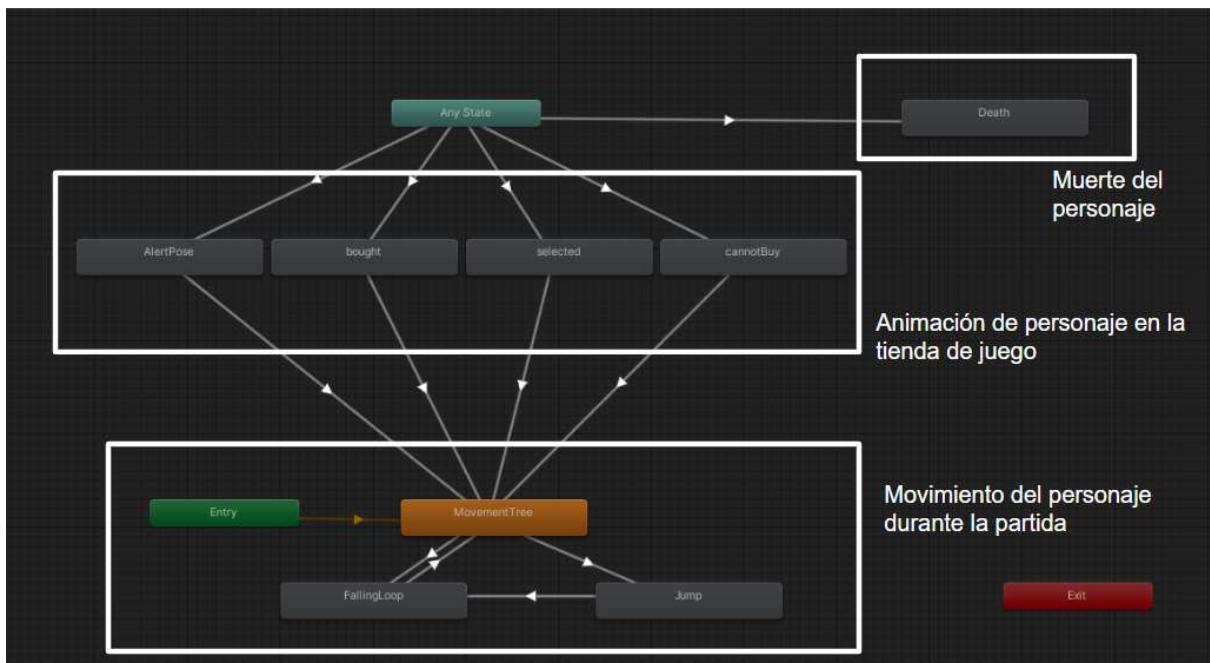


Imagen 51. Máquina de Estados Finitos del Animator del personaje principal

Por su parte, el Blend Tree²³ asociado al movimiento del personaje para estar en situación de reposo, andar o correr se conforma como sigue:

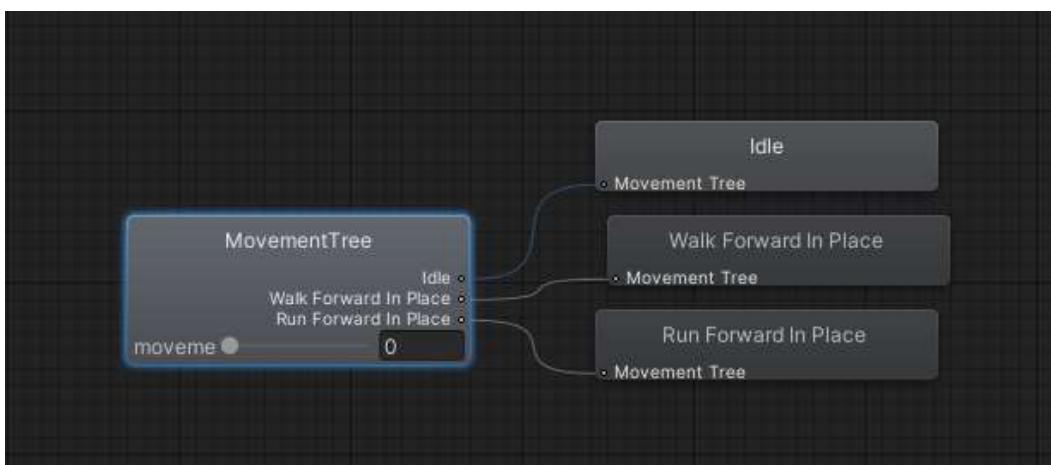


Imagen 52. Árbol de animación para el movimiento del personaje

Con esta máquina de estados finitos se logra que el personaje realice las transiciones entre los distintos estados con suavidad, ajustando el tiempo de transición y, cuando se está moviendo apoyado sobre el suelo, se mueva de forma suave y lineal, sin saltos bruscos entre animaciones.

2.5.5 Control de cámara

Durante la partida, la cámara debe enfocar en todo momento al jugador o, al menos, perseguirlo por todo el escenario de juego de forma suave pero efectiva. Para lograr esto, se ha creado un sistema de seguimiento suavizado del personaje.

Este sistema se basa en que la cámara siempre está posicionada en función de un objeto invisible que persigue al jugador mediante una interpolación temporal, de forma que el movimiento de la cámara está suavizado en el tiempo y no es tan brusco como el

²³ “Los Blend Trees son utilizados para permitir múltiples animaciones en ser mezcladas de manera suave al incorporar partes de ellas todas a diferentes grados. La cantidad en la cual cada movimiento aporta al efecto final es controlado utilizando un *blending parameter* (un parámetro de mezcla) [...]” (Unity Manual, Blend Trees)

movimiento del jugador. Esto proporciona al juego un efecto de transición suave, que evita que el jugador tenga sensación de fatiga debido a los movimientos bruscos que puede realizar el personaje frente a los movimientos que realiza la cámara.

Este sistema de seguimiento se ha logrado mediante la utilización de la función Lerp, permitiendo una interpolación lineal entre la posición en la que se encuentra el objeto de seguimiento del personaje, que determinará la posición de la cámara, y la posición del propio personaje. Así, cuando el personaje se mueve, se define una nueva posición objetivo y el objeto de seguimiento interpola entre la posición actual y la posición a la que se ha movido el personaje.

Este efecto resultó en un movimiento muy interesante que también se ha utilizado para el inicio del nivel. De este modo, cuando empieza la partida, el objeto de seguimiento del personaje está ligeramente descolocado, haciendo que se transición y se centre en el personaje a los pocos instantes de comenzar la partida.

2.6 Interfaz de Usuario (UI)²⁴

La primera iteración de la UI del juego tenía objetivo transmitir monstruosidad, ya que los protagonistas del juego son los diferentes monstruos que puede controlar el jugador.

A partir de esta premisa se generaron las primeras versiones de esta. Posteriormente se decidió optar por un estilo cartoon más amigable y no tan monstruoso. Así mismo, dado que el juego transcurría en una nave espacial se optó por estilizar la UI como si de un panel de controles de una nave espacial se tratara. A continuación se muestra la creación y evolución de los distintos elementos para la UI.

2.6.1 *Splash Screen*²⁵

Basándose en un tutorial de internet y ajustando parámetros al gusto, se creó un *Shader* pensando en un halo incómodo y una especie de disolución o desintegración. Para lograr este efecto se hace uso de la siguiente configuración de nodos:

²⁴ La interfaz de usuario es el medio de comunicación a través del cual el jugador navega a través del juego y el juego retorna información al jugador.

²⁵ Pantalla de presentación.

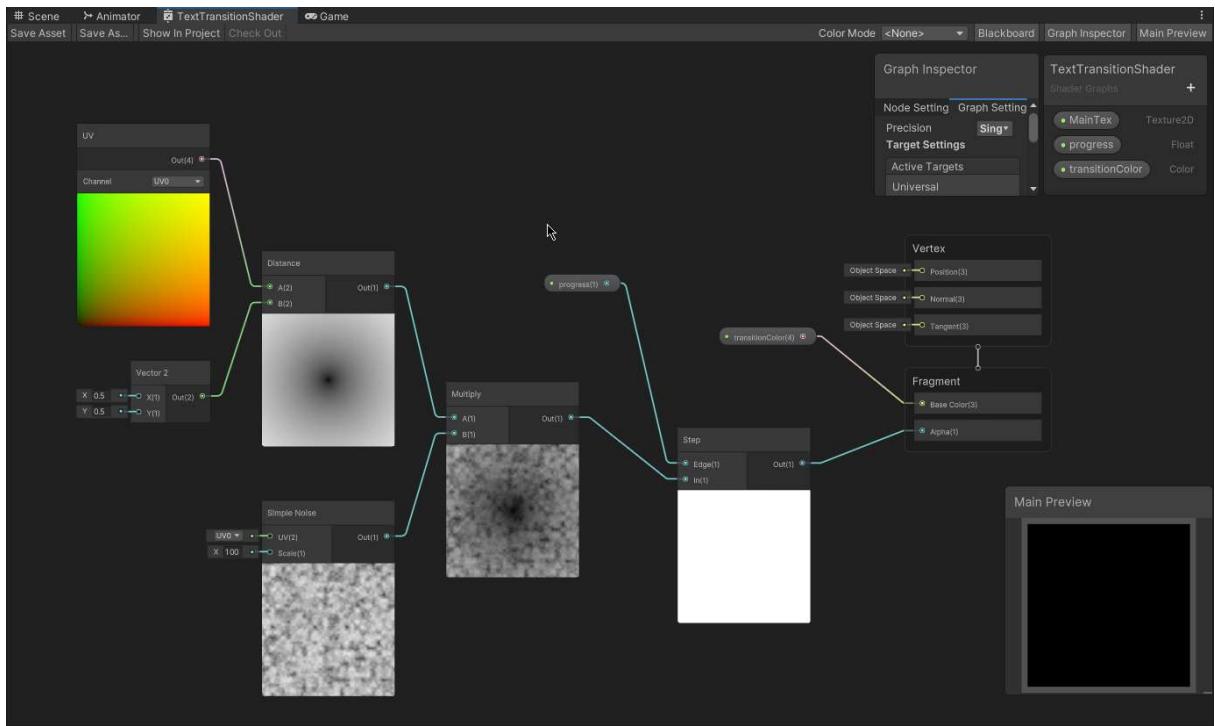


Imagen 53. Configuración de nodos.

Para conseguir el efecto deseado se crea una imagen con información de blancos, negros y grises en forma de círculo. A continuación ésta imagen se multiplica para añadir intensidad a sus valores y, además, añadirle el valor de un nodo *Simple Noise* que le da un efecto granulado. Finalmente se asigna un parámetro que controla el progreso de la visibilidad del efecto para poder manipularlo a través de una animación.

Gracias a la animación de los elementos de texto y el parámetro del efecto el resultado final es el siguiente:



Imagen 54. Secuencia de imagen 1.



Imagen 55. Secuencia de imagen 2.

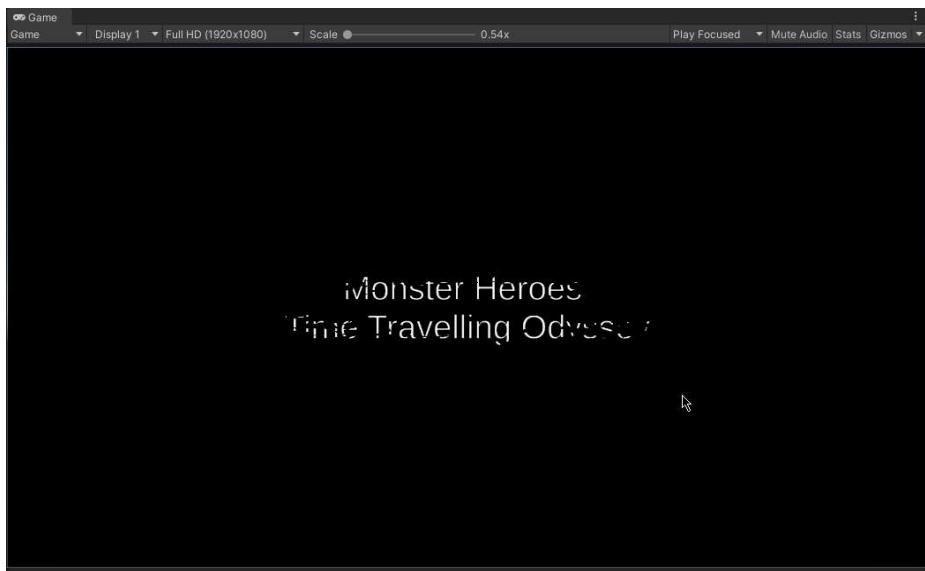


Imagen 56. Secuencia de imagen 3.



Imagen 57. Secuencia de imagen 4.

La segunda iteración de UI pretendía acentuar el aspecto cartoon así que se optó por un *shader* más sencillo y que acompaña la temática de los viajes en el tiempo, creando así un círculo que aparece como si de un barrido de una aguja de reloj se tratara.

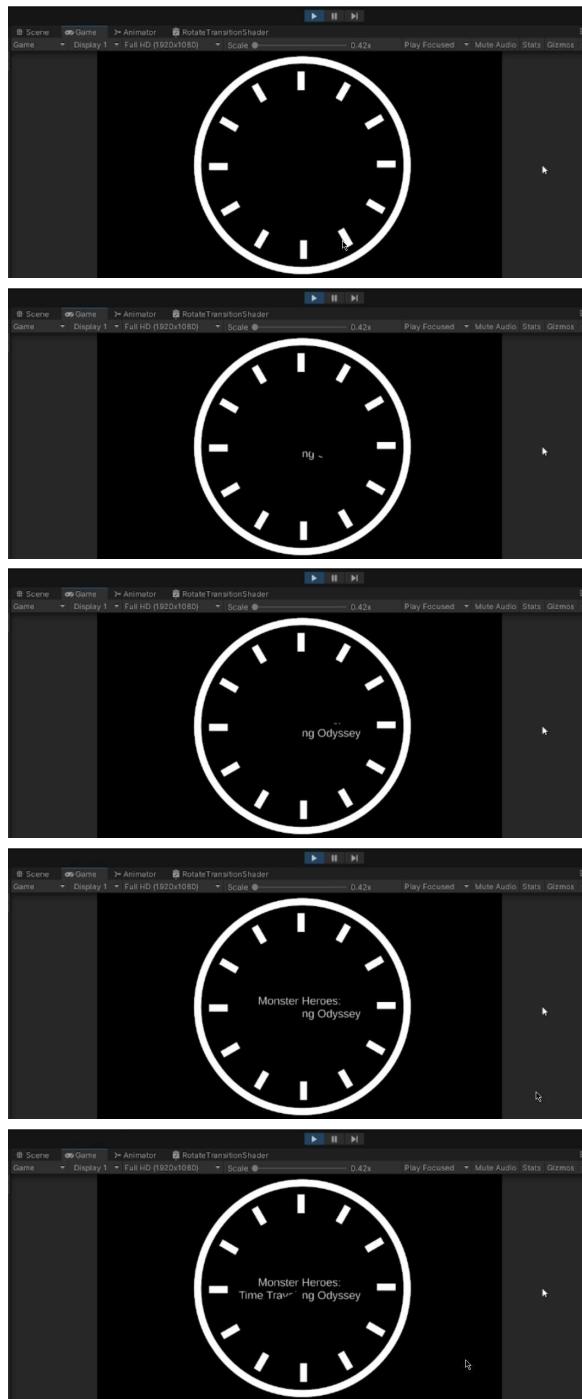


Imagen 58. Secuencia de imágenes.

Como se puede apreciar en la imagen anterior, a parte del efecto del *Shader*, se añadió una imagen creada manualmente para simular un reloj conjuntamente con el *Shader*.

Los nodos que lo conforman son los siguientes:

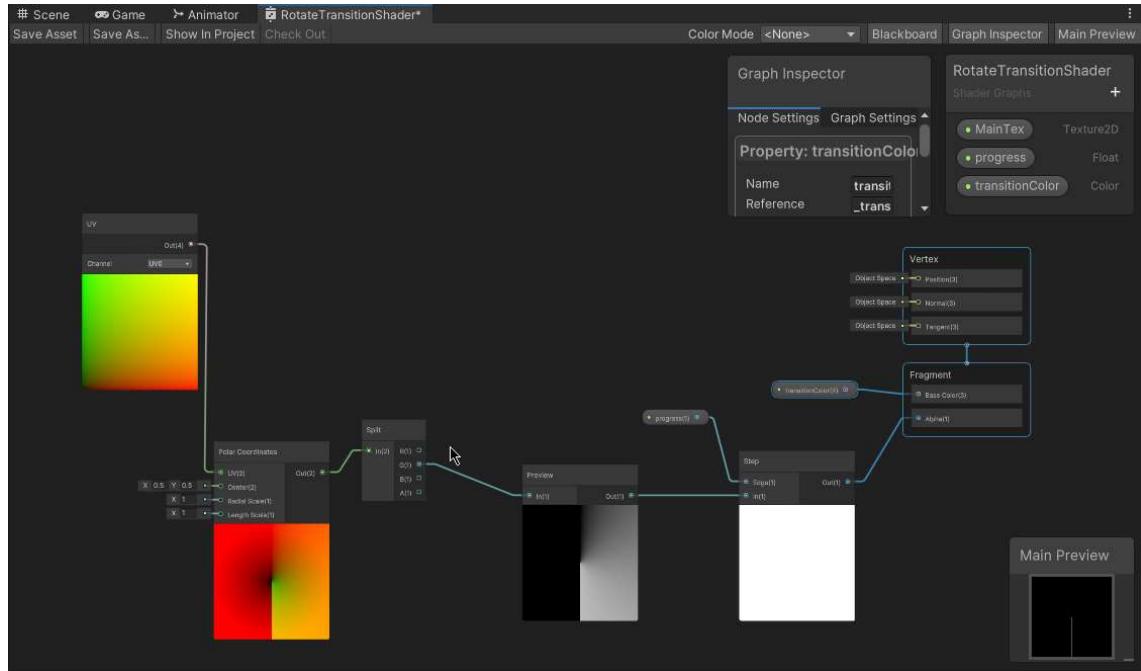


Imagen 59. Configuración de nodos

Se hace uso del nodo *Polar Coordinates*, el cual hace uso de un sistema de coordenadas bidimensional matemático, denominado con el mismo nombre²⁶. El resultado de aplicar este sistema es la conversión de los canales X e Y de la UV²⁷ de entrada. “El canal x de la entrada a UV se convierte en un valor de distancia desde el punto especificado por el valor de entrada Centro y el canal y de la misma entrada se convierte en el valor de un ángulo de rotación alrededor de ese punto.”(Unity Manual, Polar Coordinates Node). Con el siguiente nodo se pasa de color a escala de grises la información de la imagen para que esta pueda actuar como una transparencia. Finalmente se añade una variable de control llamada “progreso” que representa el progreso actual del ángulo en las coordenadas polares, consiguiendo así el efecto de transición deseado.

²⁶ En el sistema de coordenadas polares “cada punto de un plano viene determinado por una distancia respecto a un punto de referencia y un ángulo respecto a una dirección de referencia.” (Unity Manual, Polar Coordinates Node)

²⁷ UV es el nodo que representa las coordenadas X e Y de la imagen. Esta se representa en un espacio del cero al uno donde la coordenada (0,0) se sitúa abajo en la esquina izquierda y la coordenada (1,1) en la esquina de arriba a la derecha.

Para completar el efecto se animó dentro del propio motor de juegos una aguja de reloj que acompañará al barrido creado por el *Shader*.



Imagen 60. Resultado final del efecto de reloj

Finalmente la *SplashScreen* se descartó a favor de reducir el tiempo que el jugador tarda en acceder al menú principal, en pro de que este tuviera acceso directo y rápido al juego.

2.4.2 Botones

Usando el Boximon²⁸ (uno de los monstruos que formaba parte del juego en sus primeros prototipos) como referencia, se creó un botón con la forma de una boca de monstruo que pretende simular su apertura y cierre a través del estado en que se encuentra el botón (*Normal*, *Hover*, *Pressed*)²⁹.

²⁸ Modelo disponible en la Unity Asset Store utilizado en las versiones iniciales del proyecto. Para más información, visite:

<https://assetstore.unity.com/packages/3d/characters/meshtint-free-boximon-fiery-toon-series-153958>

²⁹ Los botones se encuentran en diferentes estados dependiendo de la interacción con ellos. Los tres estados básicos son *Normal* (cuando el botón está en reposo, no hay interacción), *Hover* (cuando el cursor se encuentra sobre el botón, indicando que está listo para ser pulsado) y *Pressed* (cuando el botón es pulsado).



Imagen 61. Imagen de referencia del Boximon inicial.

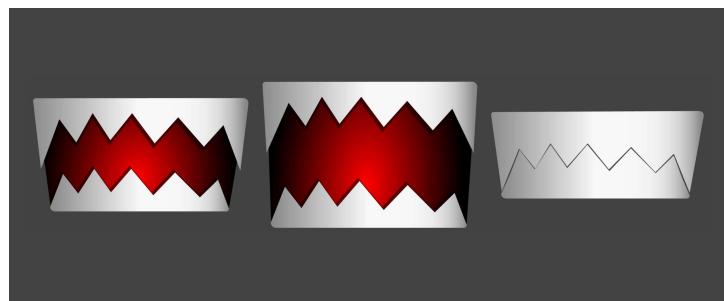


Imagen 62. Aspecto botones iteración 1. Estado *normal*, *hover* y *pressed* respectivamente.

Para crear el texto de los botones se decidió utilizar la herramienta de Unity denominada *TextMeshPro*, disponible como un añadido integrado en el motor de juego, dada su mayor versatilidad y cantidad de opciones para editar el texto. En un inicio como fuente para los textos se usó “LiberationSANS”, la fuente que pone a disposición el motor de juegos por defecto, con un efecto “Outline” que proporciona al texto un borde del color y grosor en píxeles deseado a través de la herramienta anteriormente mencionada “TextMeshPro”.

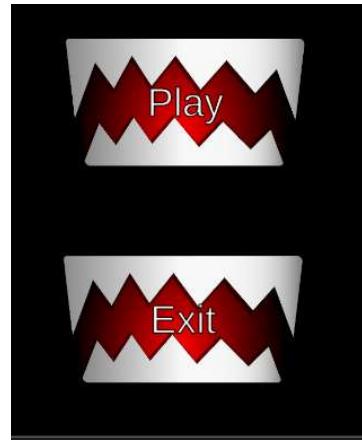


Imagen 63. Aspecto botones iteración 1 con texto.

La siguiente iteración de los botones, dada la decisión de acentuar más el estilo cartoon que la monstruosidad, cogió como referencia la interfaz de usuario One Minute GUI³⁰ de la Unity Asset Store. De este modo, se decidió conservar la idea de la boca de un monstruo. resultando en los siguientes recursos gráficos:



Imagen 64. Aspecto botones iteración 2. Estado *normal*, *hover* y *pressed* respectivamente.

Los textos que acompañaron a los botones indicando su funcionalidad conservaron la misma fuente “LiberationSANS”.

³⁰ One Minute GUI: Se utilizó como posible referencia gráfica para la interfaz de usuario. Para más información, ver: <https://assetstore.unity.com/packages/2d/gui/one-minute-gui-32346>



Imagen 65. Aspecto botones iteración 2 con texto.

La tercera y última iteración optó por enfatizar el hecho que el juego transcurre en una nave. Por este motivo los botones principales se crearon con diferentes aspectos y formas para simular el panel de control de una nave. En este caso se decidió integrar el texto en la propia imagen de los botones ya que éstos estaban en perspectiva. Para apoyar la idea del ordenador de la nave la fuente del texto integrado utilizada fue “Forced Square”³¹.

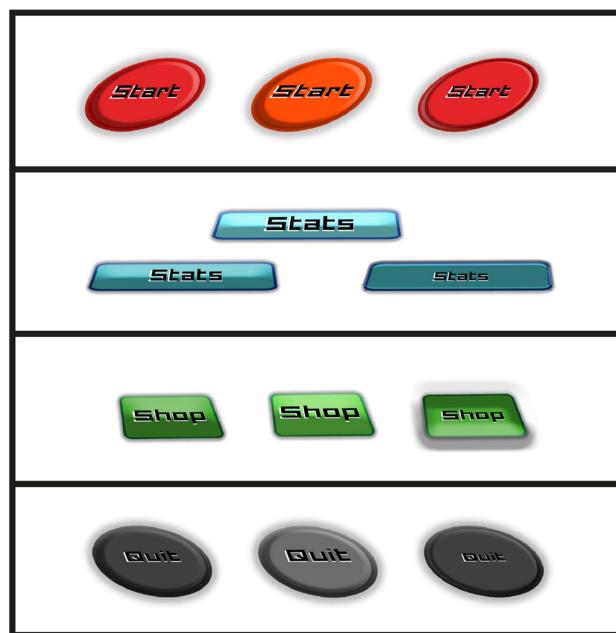


Imagen 66. Aspecto botones principales iteración 3. Estado *normal*, *hover* y *pressed* respectivamente.

³¹ Véase Bibliografía

Para los botones secundarios que se pueden encontrar en los diferentes menús de la pantalla principal del juego se decidió hacer un guiño a sus anteriores iteraciones e implementar la boca del monstruo, ya que éstos son los protagonistas, y se cogió como referencia la siguiente imagen:

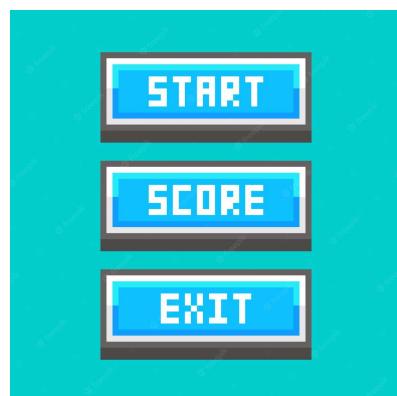


Imagen 67. Referencia³² botones secundarios iteración 3.

El resultado de la combinación de ambas referencias fue el aspecto final de los botones.

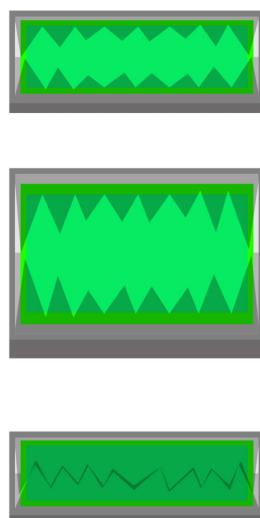


Imagen 68. Aspecto botones secundarios iteración 3. Estado *normal*, *hover* y *pressed* respectivamente.

Además dado que algunos menús requerían de botones secundarios más sencillos y a pequeña escala se realizó una variación de estos.

³² Véase Bibliografía

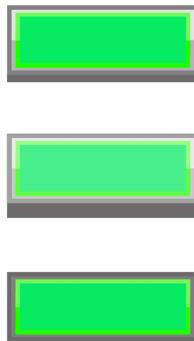


Imagen 69. Aspecto botones secundarios iteración 3 variación 1. Estado *normal*, *hover* y *pressed* respectivamente.

Como se puede observar los botones secundarios no tenían el texto integrado dado que éstos no se encontraban en perspectiva. El texto fue añadido como en la iteración 1 y 2 a través de la herramienta “TextMeshPro” con la diferencia de la fuente de texto “Forced Square”.



Imagen 70. Aspecto botones secundarios iteración 3 con texto.

Por último se realizó una segunda variación de los botones secundarios cambiando su color habitual por el rojo, para seguir en consonancia con el diseño de la pantalla de fin de juego descrita en el apartado “2.4.3 InGame UI, sección D. GameOver”.

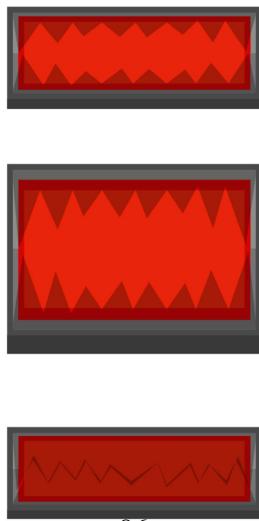


Imagen 71. Aspecto botones secundarios iteración 3 variación 2.

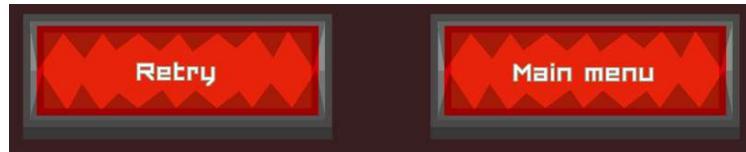


Imagen 72. Aspecto botones secundarios iteración 3 variación 2 con texto.

En esta tercera iteración se implementó un botón para silenciar el audio en el menú de pausa (el menú se encuentra descrito en el apartado “2.4.3.3 Pausa”). Con el diseño de este botón se pretendía crear un poco de contraste con el aspecto cuadriculado de la resta de botones e interfaz de usuario sin romper el estilo artístico. Éste contraste se consiguió usando un círculo estilizado sin pixelar como contenedor del icono de la nota musical.



Imagen 73. Aspecto del botón de audio. Estado *normal*, *hover* y *pressed* respectivamente.

Para indicar visualmente cuando el audio se encuentra silenciado se creó un ícono de prohibición siguiendo la misma línea para mantener el contraste.



Imagen 74. Aspecto icono prohibición y botón audio silenciado.

La funcionalidad de los botones de la interfaz de usuario ha sido programada a través del sistema de eventos nativo de Unity. Este sistema de eventos permite crear un vínculo entre la pulsación de un botón y la acción que se va a llevar a cabo al realizar esta pulsación, que podrá estar contenida en el mismo objeto o en otro disponible en la escena, según sea necesario.

Las funciones ejecutadas tras pulsar el botón han sido programadas manualmente y están contenidas, por lo general, en clases que extienden la clase `GameObject` de Unity.

Al mismo tiempo, también se ha implementado un patrón Singleton³³ para algunas de las acciones comunes en todas las escenas de juego. Un ejemplo de ello es el sistema de carga de niveles, cuya funcionalidad se encuentra en una clase que implementa este patrón y que no se destruye al cambiar de escena, permitiendo el acceso unificado, dentro de todo el juego, a dicha funcionalidad.

2.4.3 *InGame UI*³⁴

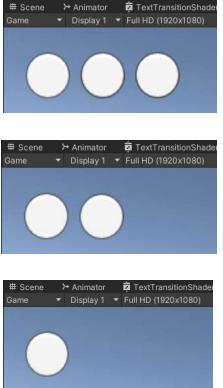
Vida

Antes del diseño del ícono se tomaron varias decisiones, ya que la vida de un personaje implica la gestión de su muerte.

³³ Un patrón de diseño Singleton es aquel que garantiza que únicamente existirá una instancia de la clase.

³⁴ *InGame UI* hace referencia a la interfaz de usuario activa durante la propia partida, por ejemplo, información necesaria para el jugador durante el *gameplay* como el número de vidas y monedas o menús únicamente accesibles durante la partida como el menú de pausa.

¿Qué o quién es capaz de quitar vida al jugador?	Los objetos dentro del juego capaces de quitar vida son las trampas (plataformas con pinchos), los enemigos y en consecuencia sus proyectiles en caso que tengan la capacidad de disparar.
¿Cuál es el número máximo de vidas que puede tener el jugador?	El número máximo de vidas es 3.
¿El jugador puede recibir más de un golpe o no?	No, el jugador perderá siempre una vida al recibir un golpe.
¿Qué pasa cuando se pierde una vida pero no es la última?	El jugador vuelve al punto inicial del nivel en el que se encuentra.
¿Qué pasa cuando se pierde la última vida?	Se carga la pantalla de Game Over.
¿Dónde aparece el personaje tras su muerte?	<p>Si todavía tiene vidas restantes el jugador aparece en el punto inicial de la sección de nivel en la que se encuentra.</p> <p>Si ha perdido su última vida se carga la pantalla de Game Over. La próxima partida empezará al principio del primer nivel.</p>
¿Qué pasa con los colecciónables tras perder una vida y al perderlas todas?	Los colecciónables se guardan tras perder una vida. Los colecciónables no se pierden al morir, por lo que no es necesario volver a mostrar el conteo al morir.

<p>¿Cómo decidimos mostrar en pantalla el número de vidas restantes?</p>	<p>El número de vidas restantes se mostrará a través del número de iconos en pantalla.</p>  <p>Imagen 75. Prototipo icono vidas</p>
---	---

Fuente: Elaboración propia

Siguiendo la misma línea que los botones, se diseñaron los iconos que representan la vida del jugador.



Imagen 76. Iconos de vida.

Monedas

Desde los inicios del desarrollo del juego se decidió tener una tienda donde comprar nuevos personajes. Así nació la idea de tener moneda propia en el juego la cual estaría representada por dos tipos de huevos: huevos normales y huevos dorados. Ambos se encuentran repartidos por los niveles donde el jugador al entrar en contacto con dicha moneda la recoge. Por este motivo surgió la necesidad de tener un contador de huevos normales y otro de huevos dorados visibles durante la partida. El proceso de diseño de los iconos fue simple. Se deformó un círculo para acomodarlo a la forma de un huevo y posteriormente se aplicó un degradado de colores en su interior. Se decidió el uso de un

degradado en lugar de un color plano para darles un efecto 3D a los iconos mediante el sombreado que se logra a través del degradado.

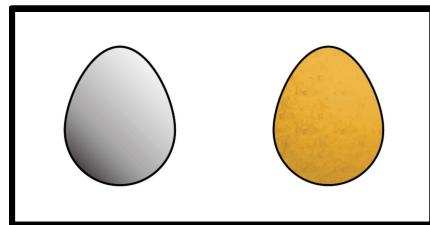


Imagen 77. Iconos de monedas. Huevos normales y huevos dorados respectivamente.

El ícono de ambos huevos se muestra conjuntamente a un texto con la misma fuente utilizada para los botones donde se indica la cantidad de monedas actuales.

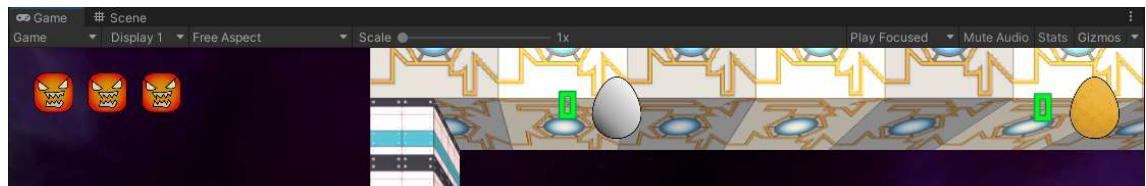


Imagen 78. *InGame* UI donde se muestran las monedas conjuntamente a la vida.

Dado que los huevos dorados son especiales y raros de encontrar en los niveles el número total de estos se muestra solo al empezar la partida y al obtener uno nuevo mediante una animación realizada en Unity. Debido a la poca cantidad de huevos dorados es fácil recordar la cantidad actual sin necesidad de mostrarla constantemente en pantalla, además en el menú principal se puede consultar el número de huevos normales y dorados actuales. Además su número e ícono se encuentran envueltos en el mismo marco que se usó en el menú principal. Se exponen detalladamente las decisiones de diseño respecto a este marco en el apartado “2.4.4 Menú Principal”.

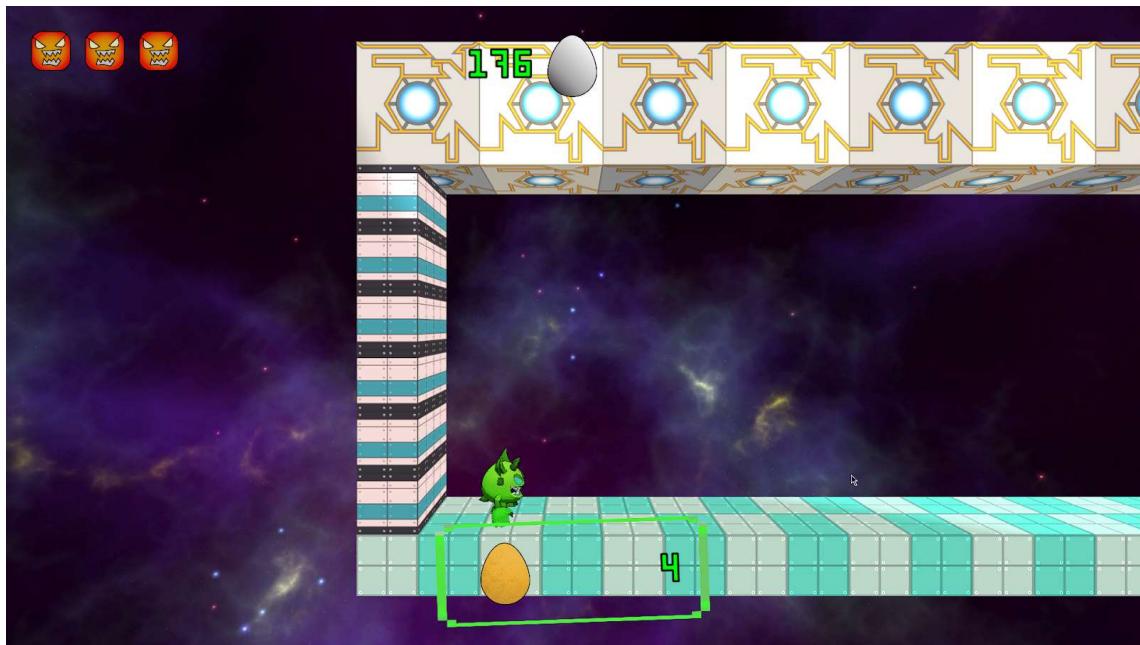


Imagen 79. *InGame* UI fotograma de la animación en la que se muestra el número de huevos dorados durante 4 segundos.

El número de monedas ya sean huevos normales o dorados se muestran recogiendo de la base de datos del juego el total de monedas actuales. Este tipo de dato se obtiene como un número entero que se transforma en un tipo de dato *string*, es decir, una cadena de caracteres, para simplificar se podría decir que es un tipo de dato que guarda escritos. En el caso de los huevos dorados cada vez que se recoge uno se reproduce la animación que muestra la cantidad de monedas.

Pausa

En cuanto a estética el menú evolucionó conjuntamente con las diferentes iteraciones de diseño, en este caso tres, tal y como se ha podido observar en el apartado “2.4.2 Botones”.

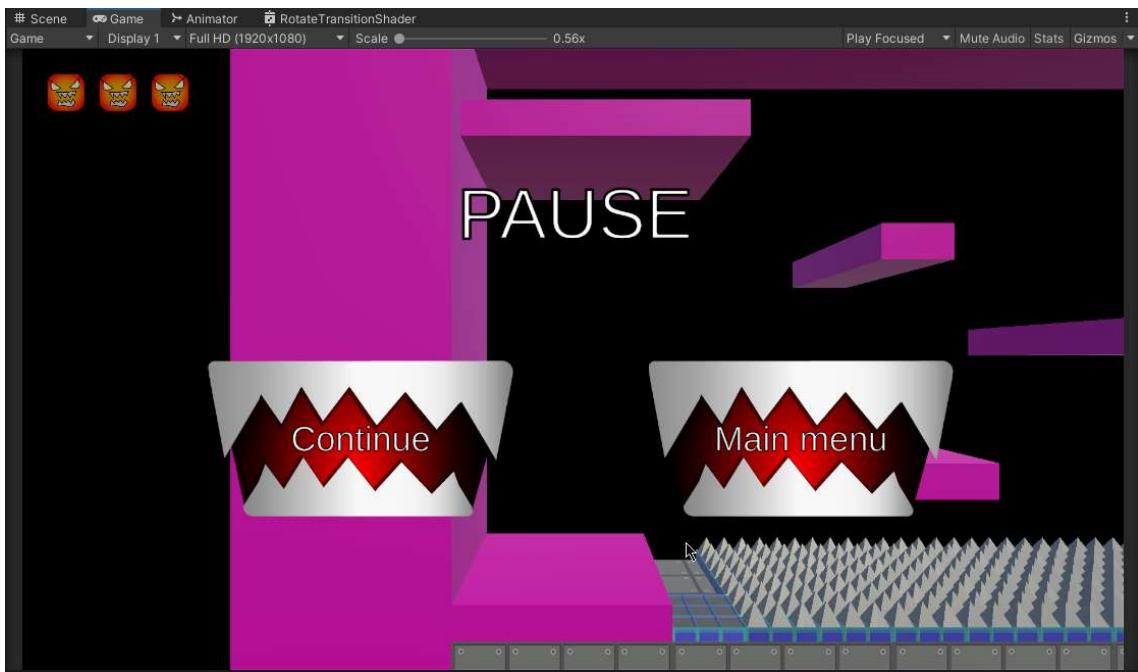


Imagen 80. Aspecto menú de pausa iteración 1.

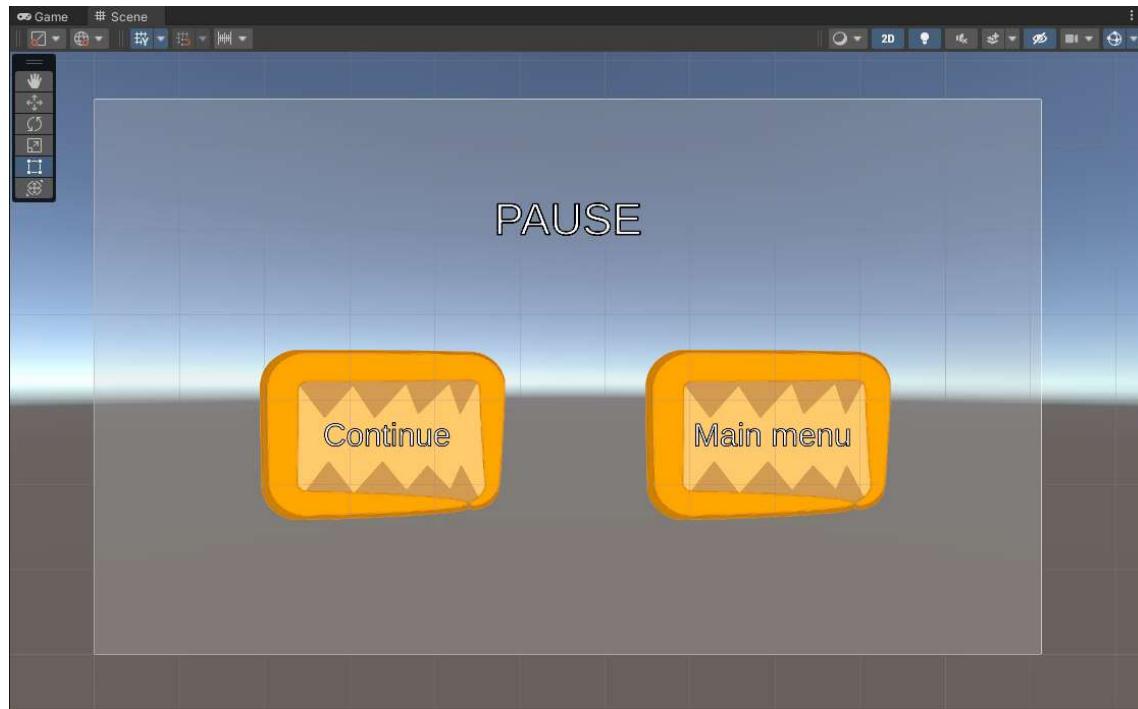


Imagen 81. Aspecto menú de pausa iteración 2.



Imagen 82. Aspecto menú de pausa iteración 3.

Cómo apreciar la tercera iteración, a diferencia de las anteriores, cuenta con un tercer botón para silenciar o activar el audio del juego, los detalles de diseño de este botón se encuentran en el apartado “2.4.2 Botones”. El interior del marco envolvente se ha cubrido con una imagen con baja opacidad de color verde para dar la sensación de que el menú es una pantalla superpuesta procedente del ordenador de la nave.

A nivel de código el juego fue diseñado para poder utilizar con facilidad una variable propia del motor de Unity denominada “timeScale”. Ésta variable representa “la escala a la que el tiempo pasa”(Unity Manual, Time.timeScale)³⁵. Esto implica que mientras su valor sea 1 el tiempo transcurre normalmente, si su valor es mayor que uno el tiempo pasará más rápido, si es menor transcurrirá más lento y si es 0 se detendrá. El hecho de detener el tiempo en el juego implica que toda acción que se ejecute cada fotograma dejará de hacerlo, con lo cual la mayoría de eventos en el juego se detienen. Existen pocas excepciones como es la música del juego ya que ésta ignora timeScale y sigue su curso aunque ésta esté a cero. Sin embargo gracias al diseño del código las excepciones no han sido problemáticas. Aún así se creó una función accesible desde cualquier fichero de código que indica si el juego está o no pausado, pudiendo así a través de una condición gestionar las excepciones. Así pues, tras pulsar el botón de pausa el valor de

³⁵ Véase Bibliografía

“timeScale” se cambia a 0 y se muestra el menú de pausa. Para reanudar el juego o volver al menú principal, tras pulsar el botón deseado, se restaura el tiempo y se ejecuta la función asignada al botón pulsado.

Game Over

Ésta pantalla se muestra cuando el jugador ha perdido todas sus vidas y brinda la opción de reintentar el nivel o volver al menú principal. Para ésta pieza de interficie de usuario se hace uso de la combinación de colores rojo y negro.

El color negro se escogió ya que “Significa [...] la muerte y la ausencia de color. [...] Al ponerlo junto a otros colores genera el efecto [...] restando sensación de amplitud.” (Daniel González Jiménez, 2014)³⁶. Además de ser un recurso recurrente en los videojuegos para indicar la muerte, siendo pues un significado ya establecido dentro del lenguaje de comunicación que usan los videojuegos.

El color rojo se eligió debido a que “Simboliza la sangre, [...]” (Daniel González Jiménez, 2014) y una vez más es un recurso recurrente en el lenguaje comunicativo habitual de los videojuegos. A pesar de ser un juego cartoon el simbolismo de éste color conjuntamente al negro comunican de manera efectiva el hecho de haber agotado las vidas del personaje, entrando en un estado de muerte permanente, finalizando así la partida.

El detalle del diseño de los botones se encuentra descrito en el apartado “2.4.2 Botones” y el marco en el que se envuelve el panel se encuentra en el apartado “2.4.4 Menú principal”. A continuación se muestra la secuencia evolutiva del *Game Over*.

³⁶ Véase Bibliografía

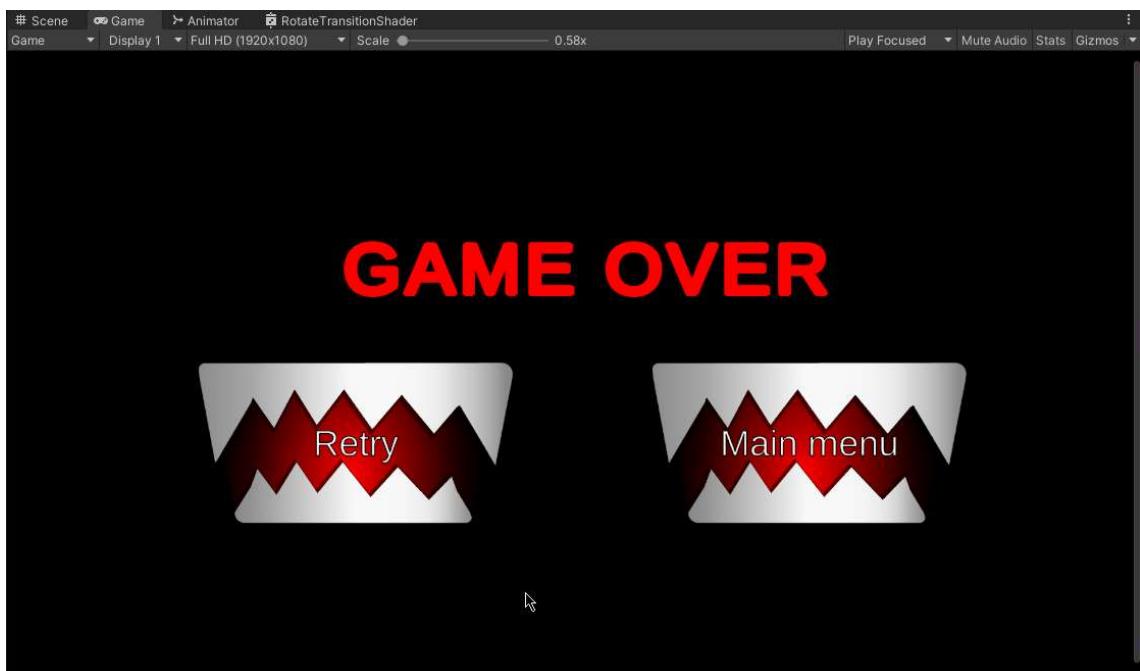


Imagen 83. Aspecto *Game Over* iteración 1 y 2.

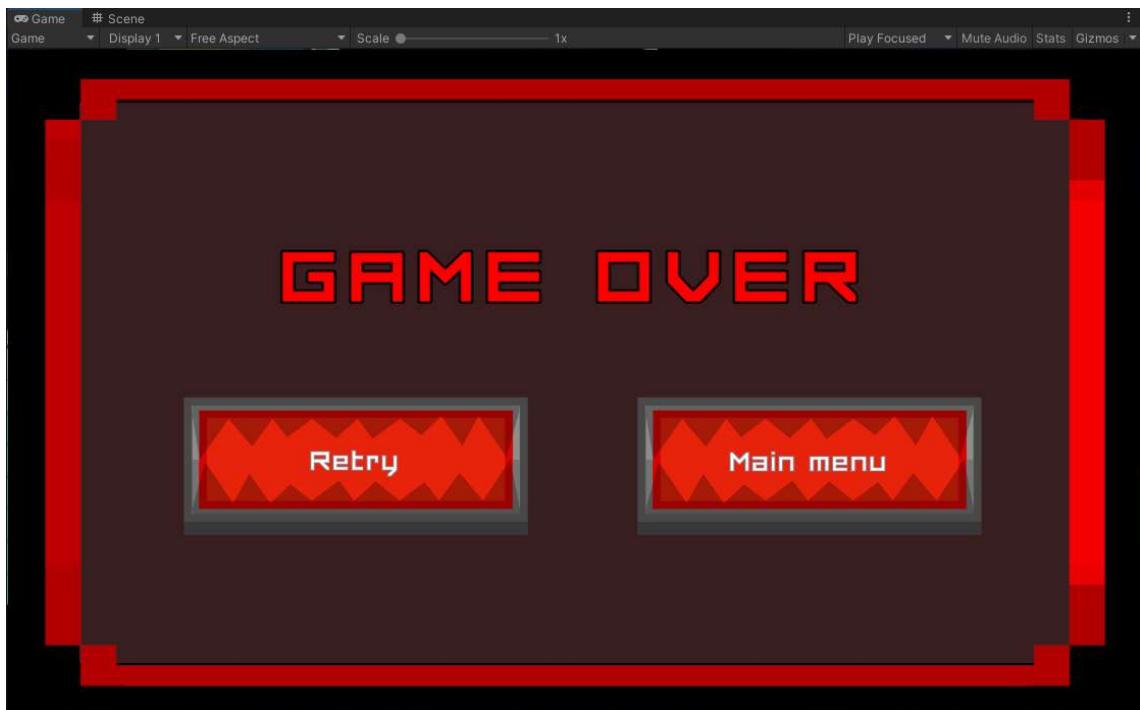


Imagen 84. Aspecto *Game Over* iteración 3.

Como se puede observar la pantalla de *Game Over* no existe un nuevo aspecto para la segunda iteración ya que el nuevo aspecto de los botones no se llegó a adaptar para ésta pantalla.

En cuanto a implementación de código cuando se cumple la condición de pérdida de todas las vidas se invoca la función responsable de mostrar el menú tras esperar el suficiente tiempo en segundos para que la animación de muerte se visualice al completo. Gracias a esa espera (*delay*) el jugador recibe el *feedback* necesario del monstruo que controla antes de que aparezca la pantalla *Game Over*.

2.4.4 Menú principal

El menú principal es la primera pantalla que se muestra al jugador. En su primera iteración se buscaba la funcionalidad mínima de este sin darle énfasis al diseño.

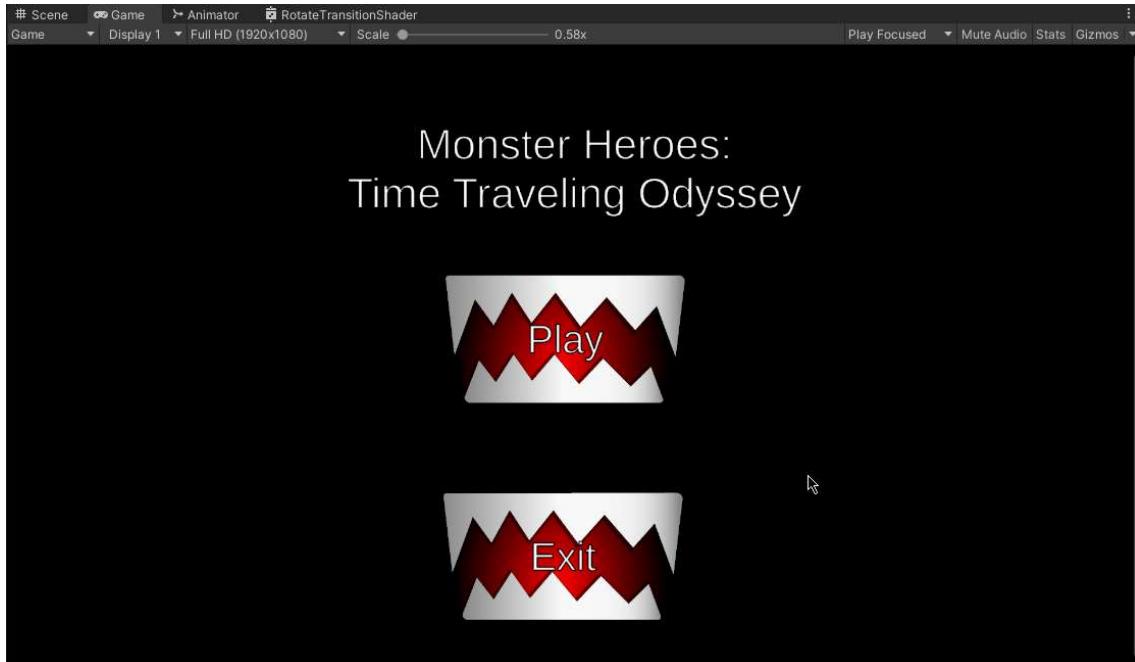


Imagen 85. Aspecto Menú Principal iteración 1.

La segunda iteración incorporó un fondo de pantalla que muestra una captura del espacio exterior estilo cartoon generado a través de la inteligencia artificial de Canva³⁷, a parte del nuevo aspecto de los botones en su segunda iteración. El nuevo fondo quiso comunicar al jugador que se encuentra en el espacio.

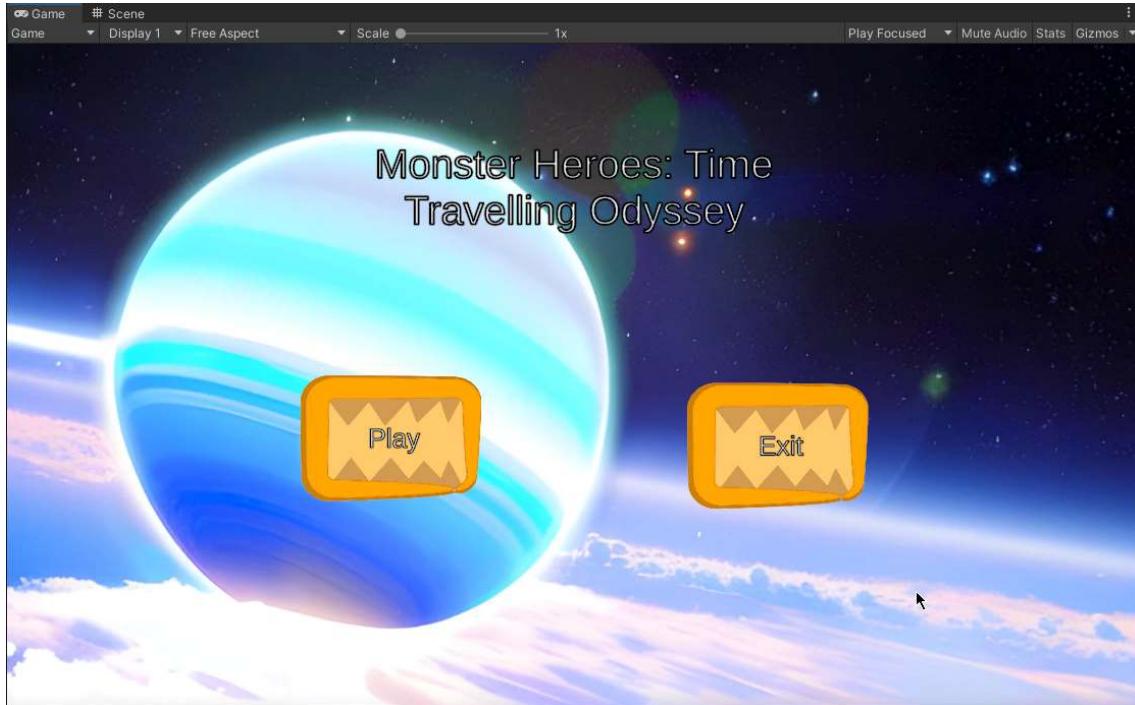


Imagen 86. Aspecto Menú Principal iteración 2.

La última iteración implementó un diseño totalmente nuevo. Éste pretendió comunicar que el juego transcurre en una nave, por este motivo tiene el aspecto de un panel de control de una nave, en este caso espacial, además de conservar el fondo incluido en la segunda iteración. Para hacer una idea de la perspectiva se utilizó como referencia la siguiente imagen.

³⁷ Canva es una herramienta online de diseño gráfico con versión gratuita. Ésta herramienta ofrece la posibilidad de generar imágenes mediante inteligencia artificial. Las palabras clave usadas como parámetros fueron “fondo espacio exterior cartoon”.



Imagen 87. Referencia³⁸ para el menú principal..

En este panel de control se encuentran los botones que dan acceso a las distintas opciones disponibles. El diseño de estos se detalló en el apartado “2.4.2 Botones”.

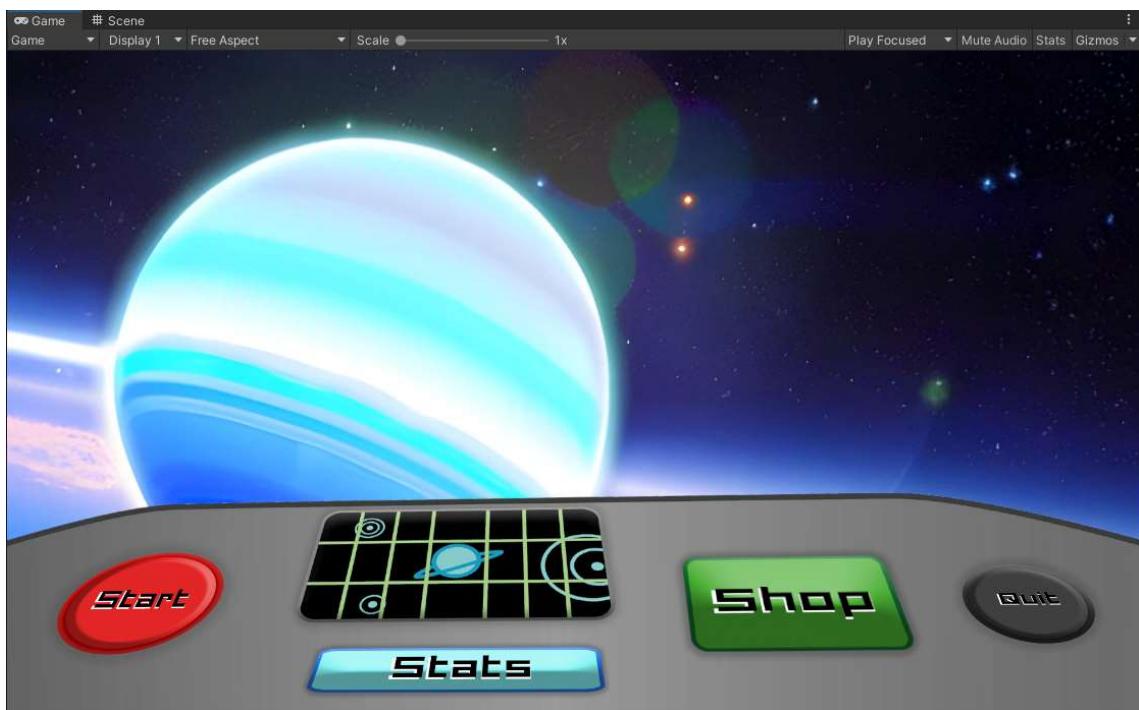


Imagen 88. Aspecto Menú Principal iteración 3.

³⁸ Referencia gráfica obtenida del siguiente enlace

https://www.freepik.es/vector-gratis/fondo-panel-control-nave-espacial_2814233.htm

Como se puede apreciar para complementar a los botones se creó un recurso gráfico con aspecto de radar para reforzar la sensación de estar ante el panel de control de una nave espacial.

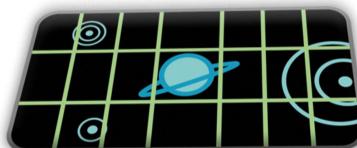


Imagen 89. Recurso gráfico. Radar.

A continuación se describe el marco que se creó para encapsular todos los paneles de opciones disponibles del menú principal. Éste marco se creó con aspecto pixelado como si una terminal de comandos del ordenador de la nave se tratara. Ésta decisión artística es un pequeño guiño a la programación ya que por lo general se relacionan éste tipo de gráficos con las personas dedicadas a ésta profesión.



Imagen 90. Recurso gráfico. Marco.

Dada la utilización general de éste recurso gráfico para enmarcar los diferentes elementos de la UI se creó una variación en escala de grises para poder posteriormente a través de la opción “Color Tint” que contienen las Imágenes en Unity darle el valor de color en RGB deseado. El resultado de esto es su utilización en la pantalla de *GameOver* descrita en el apartado “2.4.3 *InGame UI*” en la sección D.



Imagen 91. Recurso gráfico. Marco en escala de grises.

Selección de niveles (Start)

Selección de niveles es el panel en el cual el jugador puede escoger qué nivel jugar. Dado el tamaño de los botones el panel dispone de tres botones encapsulados en un espacio en el cual se puede hacer *scroll* para facilitar el acceso a estos.



Imagen 92. Panel de selección de niveles. Niveles desbloqueados.



Imagen 93. Panel de selección de niveles. Nivel bloqueado.

La posibilidad de tener un espacio donde hacer *scroll* es uno de los objetos el motor de juegos permite crear denominado “Scroll View”.

Los botones de los niveles permiten su selección a medida que se desbloquean. Para conocer el último nivel desbloqueado se consulta una base de datos interna que indica cuál es el valor del último nivel que quedó desbloqueado.

Datos (Stats)

El panel de datos nos indica el número actual de colecionables, es decir, el número de huevos normales y dorados recogidos. El número de monedas se muestra extrayendo de la base de datos la información y convirtiéndola en cadena de caracteres para poder mostrarla con texto.



Imagen 94. Panel de datos.

Tienda (Shop)

El panel tienda permite comprar a través de las monedas existentes en el juego nuevos monstruos y seleccionar el monstruo con el cual se desea jugar. El panel consta de tres secciones. En cada una de ellas se muestra el modelo 3D del monstruo debajo del cual se encuentra el botón de compra o selección, dependiendo de si el monstruo ha sido comprado o no. Si el monstruo no ha sido comprado se muestra en la parte superior de la sección su precio. Además también se puede visualizar el número de monedas disponibles y cómo se actualizan tras realizar una compra.



Imagen 95. Panel tienda.

En los modelos 3D visibles en la tienda se puede observar la reproducción de distintas animaciones. Esto es posible gracias a que la imagen mostrada es la de una cámara que enfoca a los personajes que se encuentran situados en un cubículo generado con planos. La imagen es capaz de mostrar en directo todo lo que ocurre en cámara gracias a un tipo de textura e imagen que ofrece el motor de juegos que permite esta visualización. Para su funcionamiento se indica a la cámara que convierte en textura las imágenes que capta en una “Render Texture”. Sin embargo éste tipo de textura una Imagen normal de Unity no es capaz de visualizarla. Para lograr visualizarla se usa otro tipo de imagen

denominada “Raw Image” a la cual se le puede asignar la “Render Texture” para que la muestre.

Salir (Quit)

El panel de salida se compone de un mensaje de confirmación y dos botones, dando a elegir la opción de salir definitivamente del juego o no. El botón “Yes” activa una función que proporciona Unity denominada “Application.Quit” que cierra la aplicación del juego. El botón “No” activa una función que oculta el mensaje de confirmación.



Imagen 96. Panel de salida.

2.6. Sistema de eventos

El sistema de eventos dentro del juego provee de un mecanismo sencillo, extensible y modular para detectar la ocurrencia de determinados eventos en el juego y actuar en consecuencia.

Este sistema ha sido ampliamente utilizado dentro de los niveles del juego, permitiendo escuchar la ejecución de la partida y reaccionar a ello realizando acciones como cerrar las puertas al cambiar de zona del nivel para evitar el retroceso, cambiar el punto de reinicio de la partida al perder una vida o realizar la carga y descarga dinámica de niveles.

2.6.1 Planteamiento general del Sistema de Eventos

El sistema de eventos se planteó desde un principio como un modelo abstracto capaz de extenderse con gran facilidad, con una implementación centralizada distribuida en tres clases diferentes:

- Abstract Precondition: Son los detonantes de los eventos dentro del juego. Estas clases se encargan de evaluar la ocurrencia de determinadas circunstancias en el juego.
Algunos de los ejemplos más importantes que se han implementado serían el paso de un determinado tiempo desde la activación de otro evento o la entrada del jugador en un objeto de colisión.
- Abstract Effector: Se trata del efecto que se desencadena al activarse un evento. Esta clase disparará una serie de acciones que ocurrirán dentro del juego.
Algunos ejemplos de la utilización de este tipo de effectors son la carga y descarga (o eliminación) de determinados sectores del nivel, apertura o cierre de puertas, activación de trampas, etc.
- Event Manager: Se trata del centro neurálgico de un evento. Este elemento se encarga de conectar 1..* precondiciones con 1..* efectos.

Este componente se encarga de evaluar si el evento está activo. Para ello, implementa un patrón listener que conecta las precondiciones con el Event Manager. Al detectarse un cambio en las elementos de escucha (precondiciones), determina si ha de activarse los efectos y, en su caso, notifica a todos los Effectors que han de activarse.

La siguiente imagen muestra el funcionamiento genérico del sistema de eventos:

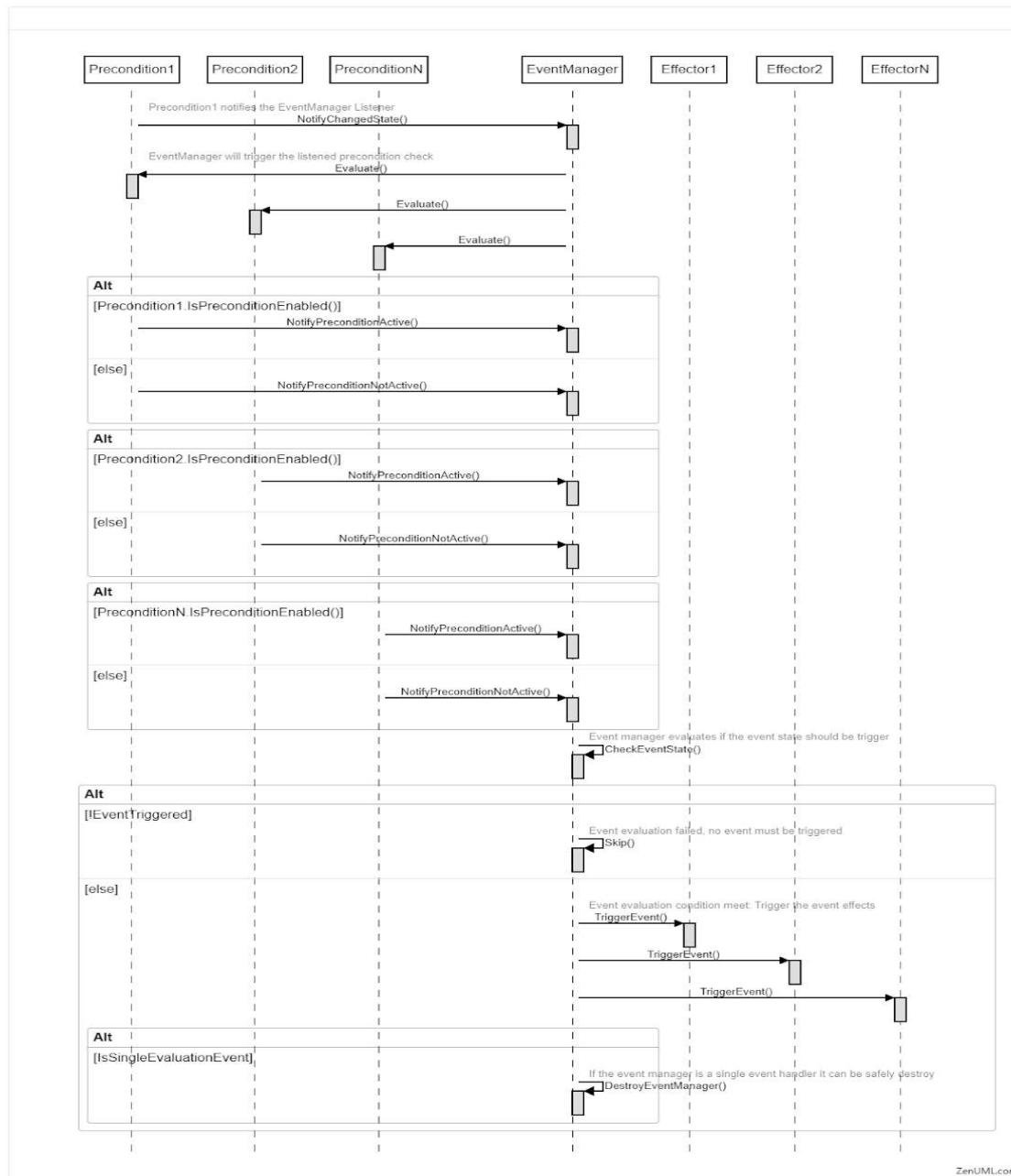


Imagen 97. Diagrama de secuencias genérico del sistema de eventos

Como puede verse en el diagrama anterior, el EventManager dispone de una lista de N precondiciones a las que escucha. El EventManager implementa un patrón listener de escucha, de forma que se evita la realización de una escucha activa (es decir, el EventManager no consulta el estado de las precondiciones, son las precondiciones las encargadas de notificar al EventManager la ocurrencia de una situación que podría cambiar el estado de ejecución de la partida).

La activación de un evento, por tanto, no es responsabilidad del EventManager, sino que son los objetos de tipo Precondition los que pueden iniciar una activación de evento. Esto sucede cuando un objeto Precondition detecta que ha cambiado su estado, por ejemplo, al llegar el jugador a una posición determinada o pasar un determinado tiempo. Al activarse esta Precondition, este notifica a todos los EventManagers que se han suscrito como listeners de la ocurrencia del evento. Cada uno de estos EventManager reacciona ante esto evaluando la situación de los elementos a los que está suscrito como listener y evaluando si ha de realizarse una modificación del estado.

Si el EventManager, tras consultar a sus Preconditions, determina que el evento ha de activarse, este reacciona notificando a todos los objetos de tipo Effector, indicando que debe de activarse el efecto asociado al mismo, por ejemplo, abriendo una puerta en el escenario o cargando en memoria una nueva sección del escenario y eliminando la sección más antigua no accesible.

2.6.2 Ejemplo 1: Desactivación de bloqueos

Uno de los ejemplos más utilizados en los niveles disponibles ha sido la desactivación de bloqueos de nivel. Este es uno de los ejemplos más sencillos para ilustrar cómo funciona el sistema de eventos.

A lo largo de los distintos niveles del juego se pueden encontrar unos botones rojos que sirven para activar o desactivar objetos. Estos botones fuerzan a que el jugador tenga que desplazarse a una zona concreta del escenario para activar una acción en el juego y, en el caso de la desactivación de bloqueos, les permita progresar en el nivel.

Los botones disponen de un collider que, al entrar el jugador dentro del mismo, activa una precondición que notifica a un EventManager, el cual, como resultado de su activación, hará que un objeto de la jerarquía de Unity se desactive y permita el paso del jugador a secciones diferentes del nivel.

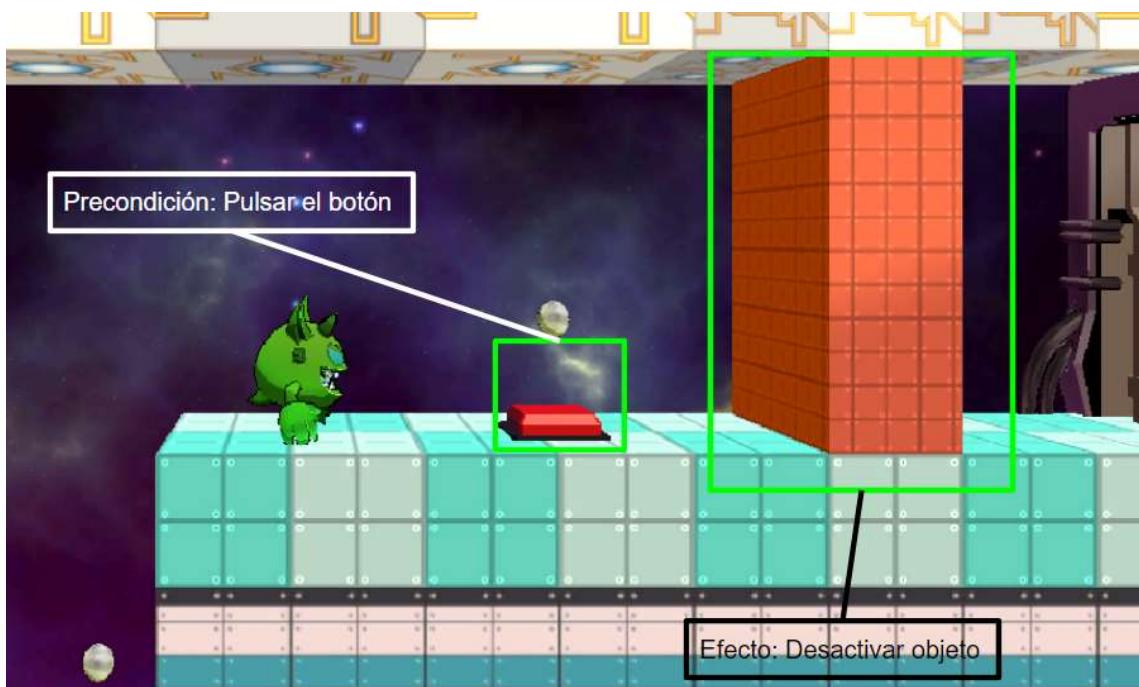


Imagen 98. Desactivación de bloqueos durante la partida

El siguiente diagrama de secuencia muestra el funcionamiento del sistema de desactivación de bloqueos. Como puede observarse, es un ejemplo muy simple en el que se puede entender con facilidad el funcionamiento del sistema de eventos para acabar desactivando un objeto en la jerarquía de objetos del motor gráfico.

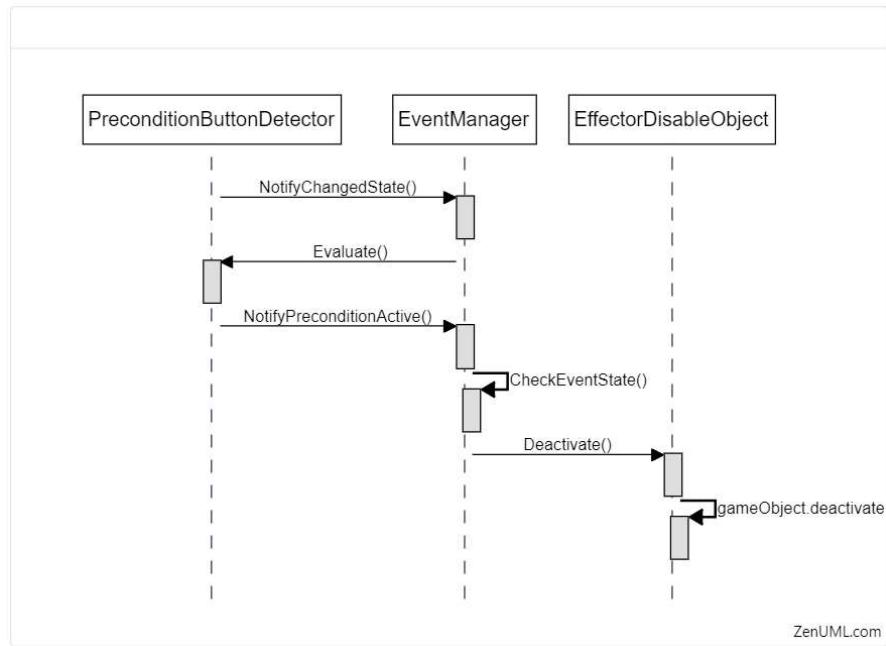


Imagen 99. Diagrama de secuencia del sistema de eventos para la desactivación de bloqueos

2.6.3 Carga y descarga dinámica de secciones de nivel

Este es uno de los ejemplos más importantes de utilización del sistema de eventos, puesto que es el principal caso de uso para el que se realizó el diseño original de este sistema.

La carga dinámica de secciones de nivel ha permitido realizar una optimización muy importante en la utilización de memoria durante la ejecución de una partida en el juego, garantizando la correcta ejecución del mismo en dispositivos de bajas prestaciones o con menor capacidad de cómputo, reduciendo la carga de procesamiento y memoria RAM utilizada durante la partida y aprovechando el propio diseño de los niveles como un mecanismo de optimización.

Los niveles en los que se interacciona durante una partida están formados por un número determinado de secciones, cada una de ellas con sus propios elementos interactivos, sistemas de eventos propios, agentes inteligentes en caso de disponer de NPCs, etc.

Mantener en memoria todos estos elementos durante la ejecución de una partida es perfectamente viable en un dispositivo de altas prestaciones como un PC o una consola. Sin embargo, esto no es aceptable en dispositivos con menores prestaciones como un dispositivo móvil de gama media donde el uso de memoria debe estar optimizado en todo momento, manteniendo la menor cantidad posible de objetos cargados en memoria en todo momento.

En caso de no realizar esta optimización la tasa de refresco del juego o FPS³⁹ podría verse afectada, reduciendo de forma considerable la velocidad de ejecución, y la experiencia de usuario.

Este sistema de eventos está asociado a cada una de las secciones de nivel. Cada nivel dispone de un collider que actúa como precondición para la carga y descarga de niveles. Esta precondición se activa cuando el jugador ha alcanzado el punto de finalización del nivel.

Al activarse la precondición, el EventManager asociado a este evento se activará y realizará las acciones de carga de sección de nivel si hay una sección de nivel tras la última sección cargada y, al mismo tiempo, en caso de que la sección más antigua deba ser eliminada, se eliminará, liberando la memoria utilizada por dicha sección.

Por ejemplo, la siguiente imagen muestra cómo el jugador se encuentra en una sección (Sección 2) y en memoria se mantienen tres secciones cargadas (secciones 1, 2 y 3) que son las secciones que podrían ser visibles dentro de la zona de juego en la que se encuentra el jugador.

³⁹ Frames Per Second, “Fotogramas Por Segundo”: Hace referencia a la tasa de refresco de imagen en el dispositivo como el conteo del número de fotogramas mostrado en la pantalla del dispositivo por segundo.



Imagen 100. Ejemplo de estado de partida antes de la detección de cambio de sección

En esta imagen se han mostrado con elementos de color verde transparente los colliders de contacto con los que se realizaría la carga y descarga de las diferentes zonas de juego.

En esta situación, cuando el jugador alcance la zona de cambio de nivel de la sección 2, deberá eliminar la sección más antigua cargada que ya no sería visible (sección 1) y cargar la siguiente sección que podría ser visible. Esta sería una nueva sección (sección 4) que puede apreciarse en la siguiente imagen.

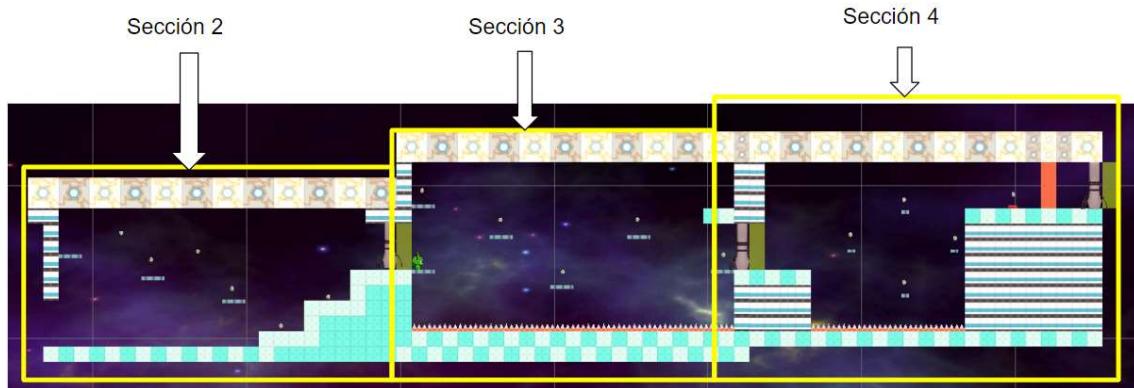


Imagen 101. Ejemplo de estado de partida después de detectar un cambio de sección

Como puede apreciarse en la anterior imagen, la sección 1 fue eliminada y la sección 4 cargada en memoria, garantizando que únicamente están cargadas aquellas secciones

que el jugador podría visualizar durante la partida, minimizando el consumo de memoria y procesador. Todas las secciones que no son necesarias son eliminadas de la escena de juego.

La siguiente imagen muestra un diagrama de secuencia que explica cómo se comunican los diferentes componentes asociados al sistema de eventos de carga y descarga de niveles.

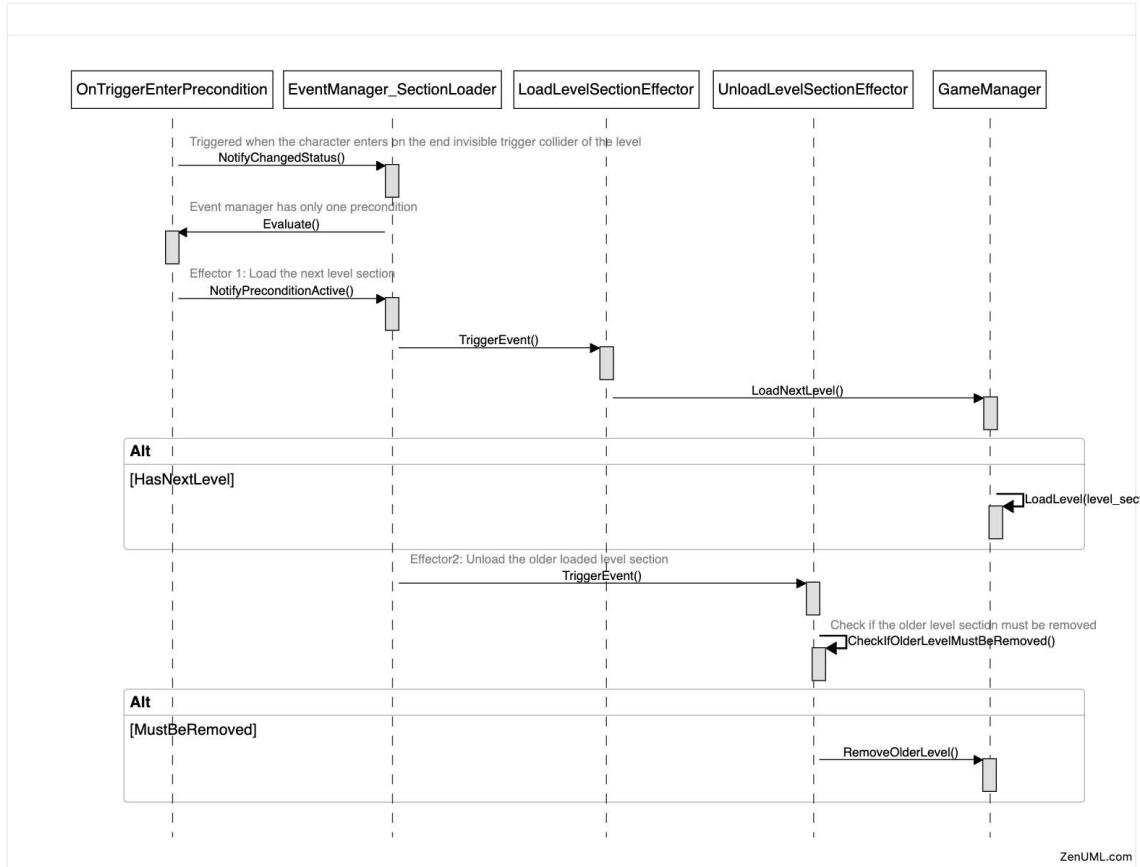


Imagen 102. Sistema de eventos para la carga y descarga de secciones de nivel

2.7 NPCs⁴⁰ e Inteligencia artificial

En esta sección se va a mostrar cómo se han implementado los diferentes elementos a tener en cuenta de la NPC, incluyendo la definición de los mismos, sus capacidades y el sistema de inteligencia artificial que define la conducta inteligente que han de exhibir estos personajes.

2.7.1 Definición de personajes

Desde las primeras fases del desarrollo del juego se definieron dos tipos de jugadores no jugables. Un robot básico con capacidad de disparar únicamente en su eje de movimiento, y un robot con capacidades avanzadas que podría disparar en 360 grados en torno a su posición dentro del entorno.

A continuación se muestra la definición de cada uno de estos personajes:

NPC	NPC_SmallWalkerBot
Imagen	
Definición de comportamiento	<p>Este tipo de enemigo es un enemigo con un desplazamiento lento dentro del mismo eje. Por defecto, este NPC podrá moverse en un eje sin saltar entre plataformas.</p> <p>Al detectar al jugador, este personaje debe ser capaz de orientarse hacia él, perseguirlo, y disparar sus dos cañones a la vez sin modificar el ángulo de sus disparos.</p> <p>No es capaz de saltar o dejarse caer.</p>
Daño	Este personaje recibirá daño al caer el jugador sobre cualquier parte del mismo. Sólo tendrá una vida asignada.

⁴⁰ NPC: Non Playable Character. Personajes no jugables. Son aquellos personajes del juego que no están controlados por el jugador.

Como puede verse, este primer NPC es el personaje básico del juego con unas capacidades mínimas pero capaz de mostrar un comportamiento inteligente.

NPC	NPC_BigRobotBlast
Imagen	
Definición de comportamiento	<p>Se trata de un personaje con una gran capacidad de movimiento. A diferencia del NPC_SmallWalkerBot, este robot será capaz de moverse rápido, saltar o dejarse caer de las distintas plataformas del juego, convirtiéndolo en un agente inteligente mucho más sofisticado.</p> <p>Este personaje, además, podrá utilizar sus brazos para apuntar en cualquier dirección en el entorno, 360 grados en torno a su orientación, de modo que no sólo disparará cuando el jugador se encuentre en el mismo plano que el NPC, sino que podrá atacarse independientemente de su posición.</p>
Daño	Este robot recibirá daño cuando el jugador caiga sobre su cabeza. El resto de partes del cuerpo no recibirá daño.

Este robot, como puede observarse, es un tipo de enemigo más avanzado, capaz de saltar entre plataformas, dejarse caer, perseguir al jugador e incluso apuntarle y disparar mientras lo persigue, convirtiéndolo en un ejemplo de enemigo más avanzado dentro del juego.

2.7.2 Feedback de personajes

Los personajes en nuestro juego debían ser capaces de exhibir un comportamiento inteligente, pero al mismo tiempo, el jugador debe ser capaz de entender qué están pensando.

Antes de entrar en detalles sobre la implementación del sistema de agentes inteligentes se debe destacar una de las decisiones de diseño que se tomaron para realizar la implementación de estos personajes. Esta decisión es, precisamente, la necesidad de transmitir al jugador el estado en que se encuentran los personajes, ya que es importante mantener al jugador informado durante toda la partida sobre este estado para evitar la sensación de trampas por parte de los NPCs.

Para lograr esto se ha implementado un sistema que notifica de la transición de estados entre calma y combate de los personajes. Esto se mostrará mediante una exclamación roja encima de la cabeza del personaje y una animación específica que indica que el personaje ha entrado en estado de combate.

Tanto para el NPC_SmallWalkerBot como para el personaje NPC_BigRobotBlast, esta animación consistirá en un salto, acompañado de la aparición y animación de un signo de exclamación encima de la cabeza del personaje.



Imagen 103. Fotograma de la animación de transición de NPC_SmallWalkerBot al estado combate

Como se puede apreciar en la imagen anterior, al detectar al personaje, el robot efectuará un salto y una exclamación de color rojo aparecerá sobre su cabeza. La siguiente imagen ilustra cómo el personaje NPC_BigRobotBlast efectúa una animación similar y también dispone del indicador de exclamación para dar a entender al jugador que lo ha detectado y comienza la transición al estado de combate.



Imagen 104. Fotograma de la animación de transición de NPC_BigRobotBlast al estado combate

Como puede verse, este robot, además de realizar el salto y la animación de la exclamación sobre su cabeza, también empieza a apuntar hacia el jugador.

2.7.3 Capabilities - Sistema de capacidades

Para la implementación del agente inteligente asociado a los NPCs, era necesario dotar a los personajes de las capacidades necesarias para poder efectuar las acciones para las que han sido diseñados, pudiendo estos tanto disponer de sensores como conocer su propia capacidad para realizar determinadas acciones.

Sensor de visión:

Para la implementación de sensores, se creó un sensor de visión basado en una modificación del modelo de visión propuesto en el libro Unity AI Game Programming (ver bibliografía).

En este libro, se plantea un modelo de visión en el que el personaje es capaz de observar a su alrededor con un cono de visión programable. El personaje únicamente necesita saber la dirección en la que puede ver, así como la distancia a la que se encuentran los objetos que puede ver.

Este modelo, aunque perfectamente válido, resulta ligeramente pesado teniendo en cuenta que únicamente se jugará en un plano de juego, ignorando uno de los ejes del espacio tridimensional. Es por esto, que para la implementación que nos ocupa, se realizó una modificación con el fin de mejorar tanto el performance como detener el procesamiento de la visión tan pronto como sea posible.

A continuación se muestra el algoritmo en pseudocódigo asociado al sensor de visión de cualquier NPC dentro del juego. Nótese que en caso de devolver FALSE, se considera que el jugador no ha sido detectado. En caso de devolver TRUE se considera que el jugador ha sido detectado.

```

Si (el jugador está muerto)
    return FALSE
Si (Distance entre NPC y jugador > distancia de vision)
    return FALSE
Si (Jugador no está por delante del NPC)
    return FALSE

ray = RAY41 (NPC_SensorPosition y Centro de masa del jugador)
Si (no ray impacta en jugador)
    return FALSE
Si (ray impacta en jugador)
    return TRUE

```

Pseudocódigo simplificado asociado al sensor de visión de los NPC del juego

Como puede verse en este pseudocódigo, se ha pretendido minimizar el cómputo realizado para calcular si el NPC es capaz de detectar o no al personaje. Para ello, se ha cancelado la operación de cálculo, indicando que el jugador no es visible, tan pronto como se ha podido, de modo que si no es posible detectar al jugador, no se prosiga con el cálculo. Esto es de vital importancia puesto que garantiza un mejor rendimiento del videojuego.

Las condiciones para detener el cálculo, ordenadas tal y como ejecuta el algoritmo de visión son:

- Vida del jugador: El NPC no debe detectar al jugador si este ha muerto.
- Distancia con el jugador: Si la distancia es mayor a la capacidad de visión del NPC, el jugador no será visible.
- Posición del jugador: Si el jugador se coloca a la espalda del NPC, el NPC no debe ser capaz de detectarlo.

⁴¹ RAY hace referencia a la operación cálculo de ray casting realizada por el motor de físicas de Unity. Esta operación se realiza entre dos posiciones en el mundo de juego y permite determinar con qué objeto ha impactado el rayo, sabiendo así si un objeto es visible o no para el cálculo de visión del NPC.

- RayCasting: Si las condiciones anteriores fallan, entonces se realiza un cálculo por RayCasting entre el NPC y el jugador. Esto es el proceso más costoso y no deberá realizarse a no ser que realmente sea posible ver al jugador.

Sensores de movimiento:

Además de los sensores de visión expuestos anteriormente, los NPCs disponen de sensores especiales que les permiten conocer su capacidad de movimiento dentro del entorno.

Estos sensores se basan en elementos asociados al NPC que le permiten determinar cómo se pueden mover. Se dispone de tres sensores:

- Sensor de impacto frontal: Permite al NPC determinar si, en caso de seguir moviéndose al frente, acabará chocando con un objeto y por lo tanto, deberá parar su avance.
- Sensor de caída: Le permite conocer si, en caso de seguir un desplazamiento frontal, el NPC caerá y por tanto, debe detener su avance.
- Sensor de salto: Permite al NPC determinar si, en caso de seguir avanzando y realizando un salto, será capaz de superar un determinado obstáculo.

La forma en que funcionan estos sensores se basa en el cálculo físico de impacto entre la posición del sensor y la capa física asociada al entorno de juego. De este modo, si un sensor impacta con un objeto determinado, se considera que el sensor está activo, si no, se considera que está inactivo.

Por ejemplo, un NPC que esté patrullando en el entorno no avanzará hasta chocar con una pared. Antes de llegar a la pared, el sensor de impacto frontal indicará que no hay un objeto con el que impactar y, por tanto, puede seguir avanzando de forma segura. Sin embargo, cuando está lo suficientemente cerca de la pared, el sensor de impacto frontal se activará, indicando que debe detenerse para no chocar con el objeto.

En esta situación, en caso de que el NPC pueda saltar para sortear un objeto, se consultará el sensor de salto. Si la pared es demasiado alta y el NPC no pudiera saltar para sortearla, el sensor de salto estaría activado, puesto que estaría detectando la pared. Si por el contrario, si se pudiera sortear la pared con un salto, el sensor de salto no estaría activo, puesto que no se detectaría pared en la altura que puede alcanzar con un salto, y el NPC sabría que, saltando, puede superar dicho obstáculo.

Capabilities:

El concepto capabilities se refiere a las posibles acciones que un agente inteligente es capaz o no de realizar.

La idea de implementar un sistema de capacidades surge a modo de experimento tras leer el sistema implementado en el juego “The Last Of US” y explicado en el libro Game AI Pro: Collected Wisdom of Game AI Professionals (ver bibliografía).

En este libro se expone como en el juego “The Last Of Us”, todos los personajes disponían de capacidades específicas y, utilizando el mismo sistema de Inteligencia Artificial, era el propio agente el que seleccionaba las acciones que podría realizar en un momento dado. Así, por ejemplo, un personaje incapaz de correr, siempre se desplazaría andando o reptando, mientras que uno capaz de correr, determinaría que en un estado de alerta, la mejor forma de moverse sería correr.

Este mismo sistema fue implementado en nuestro juego, de forma que una misma implementación de la Inteligencia Artificial del juego permite a cada NPC determinar qué acciones puede o no realizar.

Para implementar este juego se crearon las NPCConfigurations. Estas NPCConfigurations son las diferentes configuraciones que se puede asignar a un determinado personaje y que determinarán, a grandes rasgos, su propio comportamiento.

Los NPCs están así asociados a una determinada NPCConfiguration. Esta configuration está, a su vez, formada por NPCCapabilities, que son las capacidades específicas que puede realizar un determinado NPC, como saltar, perseguir al jugador, huir del jugador, etc, y una NPC Configuration, que determina los parámetros con los que se ejecutarán las acciones implementadas por el NPC. Aquí se pueden encontrar valores como velocidad, capacidad de salto, etc.

A continuación se muestra una tabla con las diferentes NPCCapabilities realizables por un determinado personaje:

Nombre	Descripción
CanChasePlayer	Indica si el NPC puede perseguir al jugador una vez que lo ha detectado. Cuando tiene esta capacidad, al detectar al jugador, el NPC puede correr para perseguirlo
MustRunAwayFromPlayer	Indica si el personaje debe huir del jugador. Aunque en la versión final del juego entregada no se ha utilizado, se implementó un sistema que hacía que, al detectar al jugador, el NPC huyera de él intentando alejarse tanto como le fuera posible.
CanJumpFromPlatformsOn CalmedState	Indica si, en un estado de calma, el personaje puede saltar de las plataformas, por ejemplo, durante su patrulla mientras no ha detectado al jugador.
IsPatrollerCharacter	Indica si el NPC puede y debe patrullar dentro de la zona de escenario en la que se encuentra mientras está en un estado de calma.
IsLookAroundWhileIdle	Indica si el NPC puede cambiar la dirección en la que mira mientras está en un estado de calma. Esto permite al NPC, por ejemplo, permanecer en un punto sin patrullar, pero mirando a su alrededor cada cierto tiempo.
CanJumpFromPlatformOnC ombatState	Indica si el NPC puede saltar desde o entre plataformas en estado de combate. Esto permite, por ejemplo, perseguir al jugador sorteando obstáculos según indique el sensor de salto del NPC
CanShoot	Indica si el NPC puede y tiene la capacidad para disparar.
MustAim	Indica si el NPC debe apuntar antes de disparar.

Como puede verse, esta configuración permite definir, de forma completa, diferentes perfiles de enemigos dentro del juego. Por ejemplo, un personaje que deba perseguir al jugador, apuntándole, saltando y, mientras no lo vea, pueda patrullar por la zona sin saltar ni dejarse caer de la plataforma en la que se encuentra, tendría una configuración en la que dispondría del siguiente conjunto de NPCCapabilities:

```
NPCCapabilities = {CanChasePlayer, IsPatrollerCharacter,  
CanJumpFromPlatformOnCombatState, CanShoot, MustAim}
```

Adicionalmente, la configuración del personaje deberá incluir valores que permitan determinar cómo realizará las acciones. Esto es la configuración específica del personaje e incluirá valores como su velocidad de desplazamiento, capacidad de salto, etc. La siguiente tabla muestra los parámetros de configuración que se han utilizado para crear cualquiera de los NPCConfigurations dentro del juego:

Nombre	Descripción
WalkSpeedMultiplier	Indica la velocidad de movimiento mientras el personaje anda por el escenario.
RunningSpeedMultiplier	Indica la velocidad de movimiento mientras el personaje corre por el escenario.
JumpForce	Indica la fuerza de salto del personaje. Esto determinará la altura que puede alcanzar en un salto y afectará directamente al sensor de salto del personaje.
VisionDistance	Indica la máxima distancia a la que el personaje puede detectar al jugador
StopDistance	Indica la distancia a la que, en caso de perseguir al jugador, debe detenerse el NPC.

Estos parámetros de configuración, por tanto, permiten determinar el modo en que realizará las acciones el NPC durante la partida.

Todo esto se combinó para crear un ScriptableObject, de forma que se pudieron crear, dentro de Unity, objetos de configuración específica de personajes y asignarlos directamente según fuera necesario.

Para terminar esta sección, a continuación se muestra un ejemplo de NPC utilizado durante el desarrollo del videojuego:

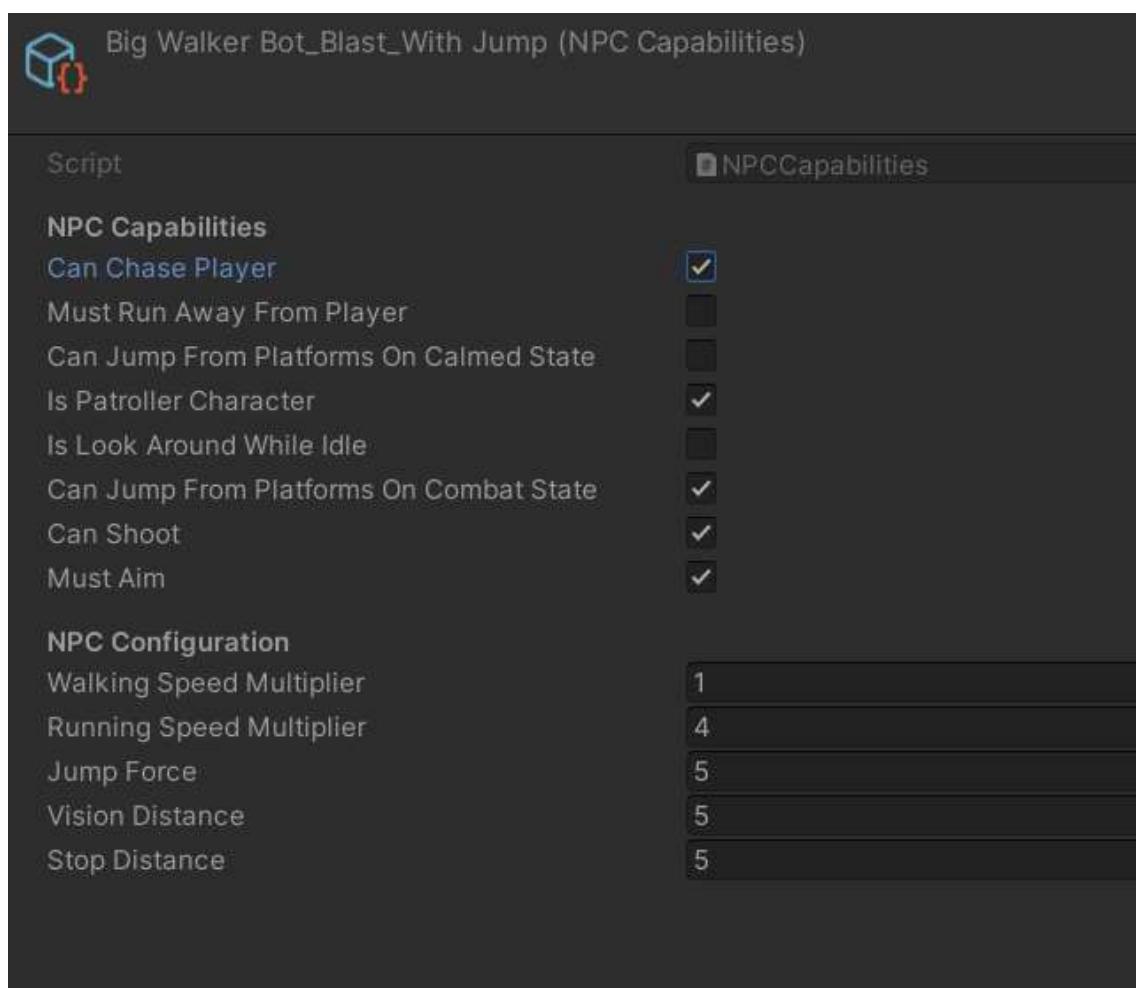


Imagen 105. Configuración básica de NPC_BigRobotBlast en el juego

2.7.4 Inverse Kinematics

El personaje no jugable NPC_BigRobotBlast se caracteriza por su capacidad para apuntar en cualquier dirección dentro del plano en el que se encuentra. Sin embargo, no

se dispone de animaciones específicas para lograr que este personaje realice ese movimiento con sus brazos.

Para lograr el efecto de apuntado, se utilizó la técnica conocida como Inverse Kinematics. Esta técnica permite determinar el movimiento que ha de realizar un determinado agente robótico para alcanzar una posición determinada y se ha utilizado en diversos campos de Inteligencia Artificial, como la robótica, sistemas de conducción automática, etc.

Para más información sobre Inverse Kinematics, por favor consulte la sección Unity Inverse Kinematics de la bibliografía.

Para implementar este sistema de Inverse Kinematics se empleó el nuevo sistema de Inverse Kinematics propuesto por Unity, se utilizó la topología de huesos del esqueleto y se crearon dos componentes “RigBuilder” asociados a la jerarquía de huesos de los brazos del robot.

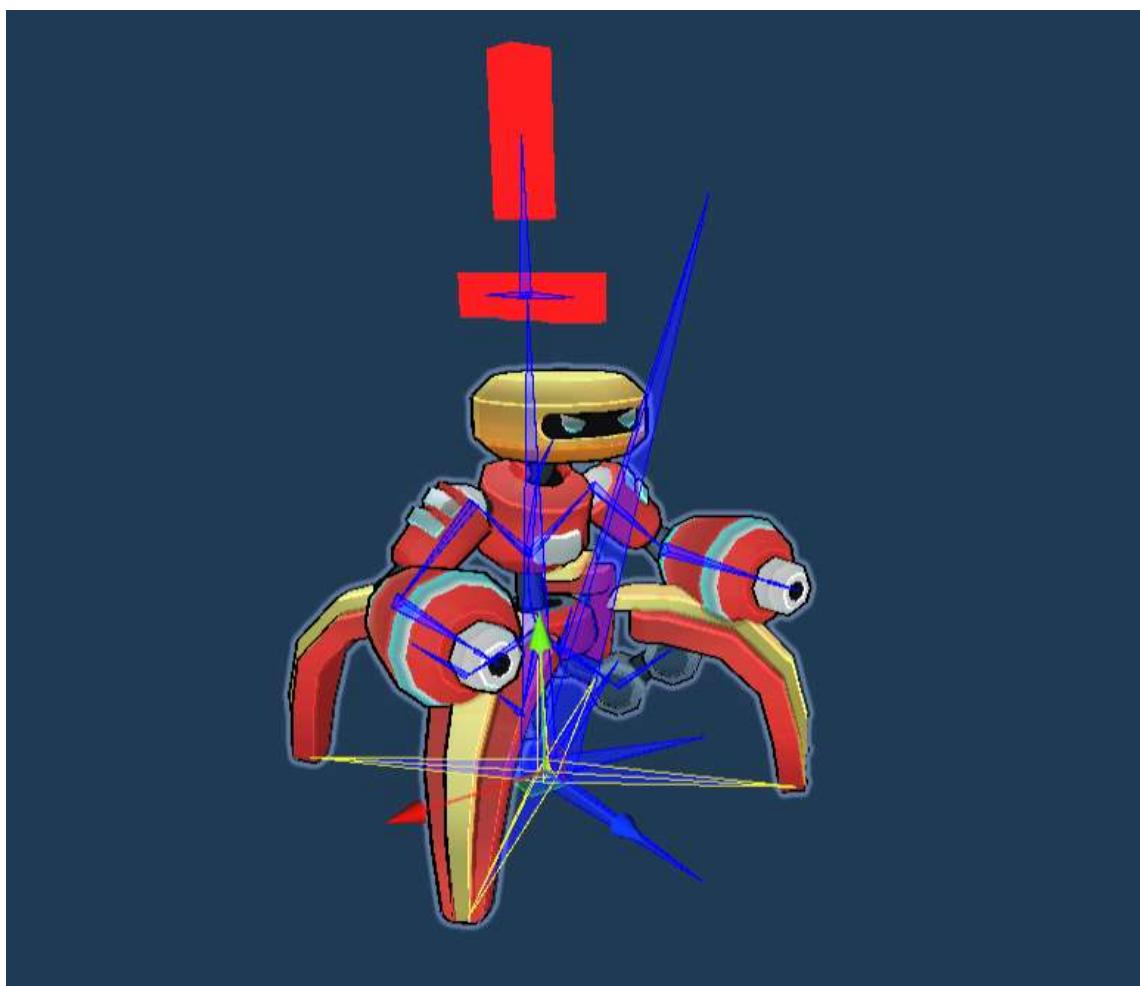


Imagen 106. Representación gráfica de la jerarquía de huesos del NPC: NPC_BigRobotBlast

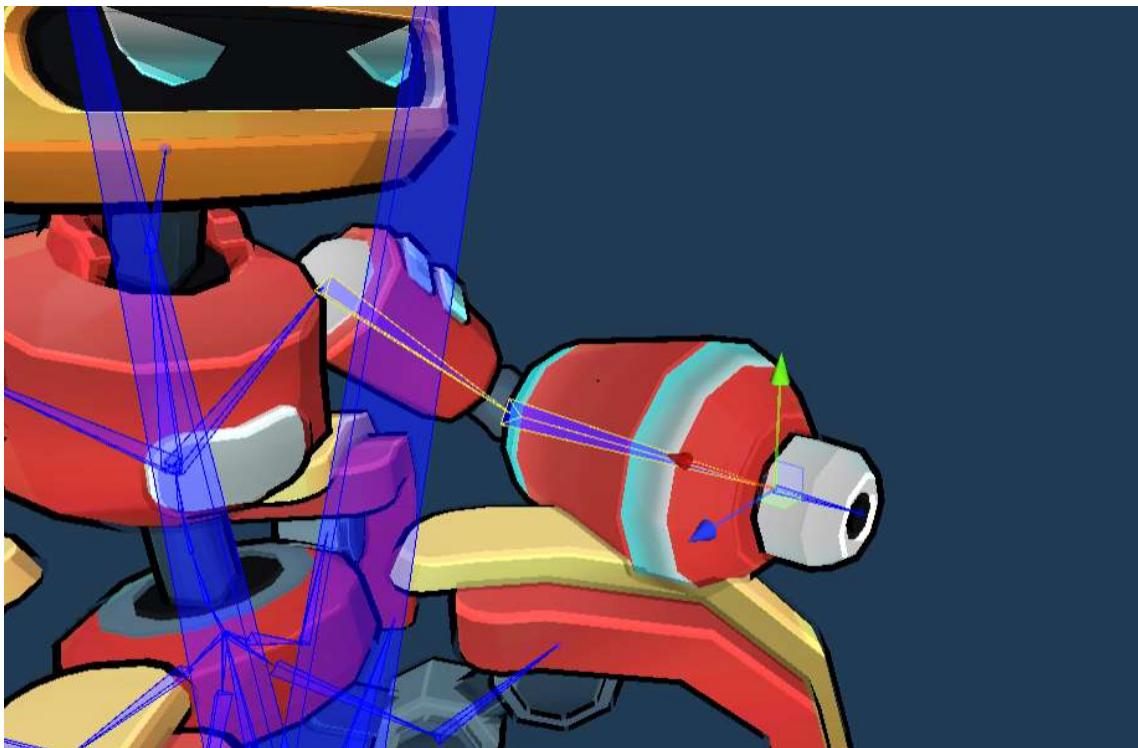


Imagen 107. Huesos asociados al RigBuilder utilizado para la animación por Inverse Kinematics

Como puede apreciarse en la imagen anterior, los huesos del brazo son los que se debían modificar para lograr un efecto de apuntado hacia el personaje. Por ello se construyó el sistema de Inverse Kinematics utilizando únicamente los huesos del brazo.

Utilizando estos InverseKinematics, cuando el NPC detecta al jugador, se realiza una transición suave entre el movimiento del brazo del jugador durante la animación que esté realizando y la posición que indica el Inverse Kinematic indicator que debe colocar el brazo para dirigirlo completamente hacia el jugador.

Esta transición en la fuerza de animación permite lograr un movimiento natural en el que el NPC pasa de mover los brazos con su animación asignada a apuntar directamente al jugador para disparar.

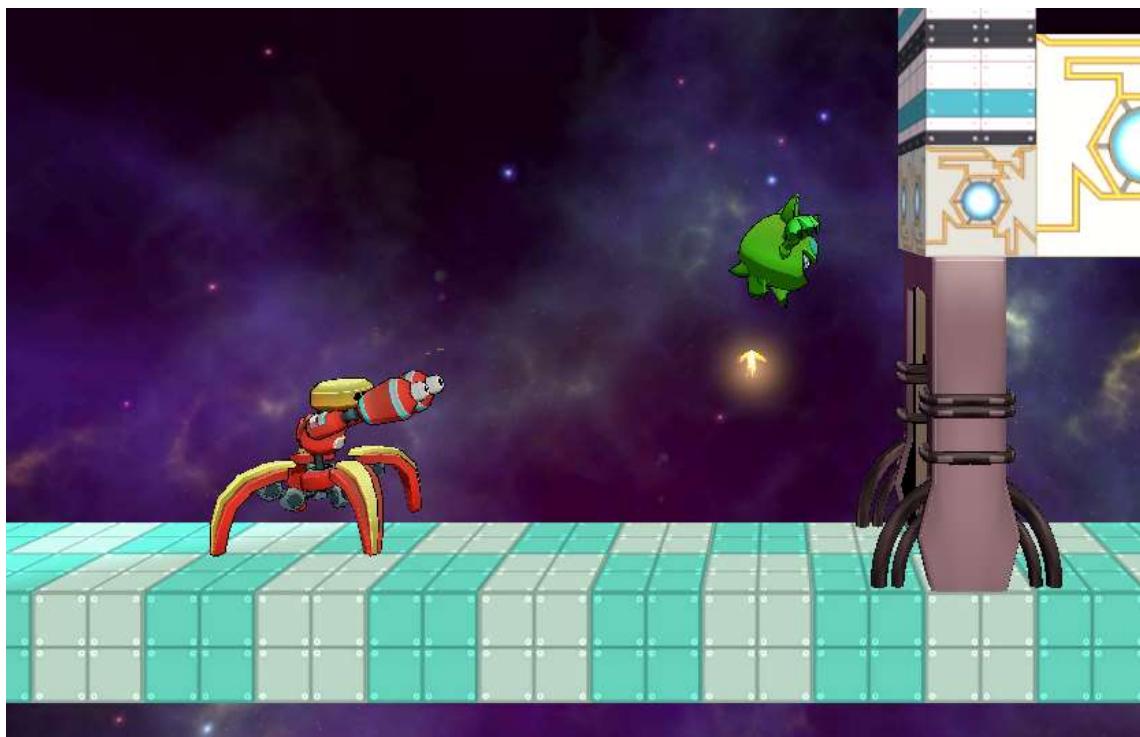


Imagen 108. Apuntado mediante la técnica Inverse Kinematics

2.7.5 Definición del agente inteligente

Para la creación de los agentes inteligentes en el juego se seleccionó utilizar la técnica de Behaviour Trees.

Los Behaviour Trees son una estructura de datos en la que se plasman las reglas de los comportamientos que pueden ocurrir, así como el orden en que estos se ejecutarán. El término árbol proviene de la utilización de la estructura de datos de tipo árbol típica de la informática, donde los nodos están organizados desde el nodo padre (root) hasta los nodos hoja (leaf).

En un Behaviour Tree, además, los nodos intermedios, por lo general, serán nodos de decisión que evaluarán las acciones que se pueden realizar, así como nodos que determinan cómo ejecutar los nodos que aparecen por debajo en la jerarquía del árbol. Por su parte, los nodos hojas, serán tareas específicas que se pueden realizar y que conforman el comportamiento del árbol.

Para analizar por tanto el comportamiento del agente inteligente que se ha propuesto, se deben analizar tanto las tareas que se pueden realizar, así como la topología del árbol de decisiones (Behaviour Tree) que se ha implementado.

A continuación se analizará tanto la tecnología seleccionada para implementar el Behaviour Tree, como las tareas y la misma estructura del árbol que se ha implementado.

PandaBT

A la hora de seleccionar con qué tecnología se iba a trabajar para implementar el Behaviour Tree se analizaron las diferentes opciones de las que se disponía para la implementación. Dado que Unity no implementa un sistema por defecto que se pueda utilizar, se analizaron las siguientes posibilidades:

1. Implementar un Behaviour Tree básico: Este es el ejemplo implementado en el libro Unity AI Game Programming (ver bibliografía). Sin embargo, dada la naturaleza y poco tiempo para realizar la implementación, esta idea se descartó.
2. Behaviour Designer: Este es un sistema disponible para su compra en la Unity Asset Store de Unity. Se puede encontrar en el siguiente link:

<https://assetstore.unity.com/packages/tools/visual-scripting/behavior-designer-behavior-trees-for-everyone-15277>

Este sistema plantea un gran beneficio y es que se realiza un diseño gráfico del sistema, lo que facilita el diseño de la Inteligencia Artificial así como la facilidad para su análisis.

El mayor inconveniente de este sistema es su precio prohibitivo, ya que una licencia de desarrollador cuesta 89,82€

3. PandaBT Free: Este sistema permite definir la estructura de datos del Behaviour Tree, así como controlar su ejecución de una forma sencilla utilizando un DSL⁴² específico con una sintaxis similar a Python. Se puede encontrar en el siguiente link:

<https://assetstore.unity.com/packages/tools/behavior-ai/panda-bt-free-33057>

Esta opción, aunque resulta más difícil de analizar puesto que la definición del Behaviour Tree requiere de la utilización de un lenguaje de programación específico resultó ser la más conveniente puesto que permitía definir el Behaviour Tree y gestionarlo de forma sencilla y sobreponiendo el mayor inconveniente de Behaviour Designer.

Finalmente, se seleccionó utilizar PandaBT como opción para representar el Behaviour Tree asociado al agente inteligente.

Tasks:

Dado que la implementación del Behaviour Tree estaría basado en PandaBT, se definieron todas las tareas que eran necesarias para implementar el comportamiento completo del árbol.

Esto abarcó una serie de tareas que debían resultar en resultados positivos o negativos en función de si la tarea se había considerado que se completó de forma correcta o incorrecta.

Así mismo, las tareas que se han implementado se han categorizado dentro de dos tipos. Tareas de consulta y tareas de acción. Las tareas de consulta son aquellas que se utilizan para, según el resultado de la misma, seleccionar un camino de ejecución dentro del Behaviour Tree. Las tareas de acción son aquellas que llevan la ejecución de una acción específica por parte del agente inteligente.

⁴² DSL: Domain Specific Language. Se trata de un lenguaje de programación de alto nivel utilizado optimizado para representar o solucionar problemas de un dominio determinado.

A continuación se detallan las tareas de consulta que se implementaron para diseñar el Behaviour Tree:

Nombre	Task_CanMoveForward
Descripción	Tarea para comprobar si el personaje se puede mover adelante o no
Resultado Éxito	El personaje puede moverse hacia adelante en función de los valores indicados por los sensores disponibles
Resultado Fallo	El personaje no puede moverse hacia adelante en función de los valores indicados por los sensores disponibles

Nombre	Task_CanMoveForwardIgnoringFall
Descripción	Comprueba si el personaje se puede mover adelante ignorando los sensores de caída
Resultado Éxito	El personaje se puede mover hacia adelante ignorando sensores de caída
Resultado Fallo	El personaje no puede moverse hacia adelante ignorando los sensores de caída

Nombre	Task_HasPlayerBeenDetected
Descripción	Indica si el jugador ya ha sido detectado o no por el NPC. No realiza la comprobación de si es visible o no en la actualidad, sino que es una tarea para comprobar si fue detectado anteriormente
Resultado Éxito	El jugador ha sido detectado
Resultado Fallo	El jugador no ha sido detectado

Nombre	Task_IsPlayerDetected
Descripción	Indica si el jugador es actualmente visible. Realiza la comprobación ejecutando la comprobación con el sensor de visión
Resultado Éxito	El jugador es detectado por los sensores de visión
Resultado Fallo	El jugador no es detectado por los sensores de visión

Nombre	Task_IsAlive
Descripción	Comprueba si el NPC está vivo o no
Resultado Éxito	Si el NPC está vivo
Resultado Fallo	Si el NPC no está vivo

Nombre	Task_CanChasePlayer
Descripción	Indica si el NPC puede perseguir o no al jugador
Resultado Éxito	El NPC puede perseguir al jugador
Resultado Fallo	El NPC no puede perseguir al jugador

Nombre	Task_MustRunAwayFromPlayer
Descripción	Indica si el NPC debe huir del jugador o no
Resultado Éxito	El NPC debe huir del jugador
Resultado Fallo	El NPC no debería huir del jugador

Nombre	Task_CanNPCJumpOnCalmmedState
Descripción	Indica si el NPC puede saltar cuando se encuentra en estado calmado
Resultado Éxito	El NPC puede saltar en estado calmado
Resultado Fallo	El NPC no puede saltar en estado calmado

Nombre	Task_CanNPCJumpOnCombatState
Descripción	Indica si el NPC puede saltar cuando se encuentra en estado de combate
Resultado Éxito	El NPC puede saltar en estado combate
Resultado Fallo	El NPC no puede saltar en estado combate

Nombre	Task_IsPatrollerNPC
Descripción	Indica si el NPC debe patrullar o no en estado calmado
Resultado Éxito	El NPC debe patrullar mientras esté en estado calmado
Resultado Fallo	El NPC no debe patrullar en estado calmado

Nombre	Task_IsFalling
Descripción	Indica si el NPC está cayendo o si no está apoyado en el suelo
Resultado Éxito	El NPC no está apoyado en el suelo. Se encuentra flotando en el aire porque ha saltado o está cayendo.
Resultado Fallo	El NPC está apoyado en el suelo.

Nombre	Task_MustNPCWait
Descripción	Indica si el NPC debe esperar algún tiempo antes de ejecutar otra acción
Resultado Éxito	El NPC debe esperar
Resultado Fallo	El NPC no tiene por qué esperar para efectuar otra acción del Behaviour Tree

Nombre	Task_MustLookAround
Descripción	Indica si el personaje tiene la capacidad de mirar a su alrededor cuando está en estado calmado
Resultado Éxito	El NPC tiene la capacidad de mirar a su alrededor
Resultado Fallo	El NPC no tiene la capacidad de mirar a su alrededor

Nombre	Task_IsCharacterAtStopDistance
Descripción	Indica si el NPC se encuentra a la distancia de parada cuando persigue al jugador
Resultado Éxito	Si se encuentra a una distancia menor o igual que la distancia de parada
Resultado Fallo	Si se encuentra a una distancia superior a la distancia de parada

Nombre	Task_CheckReturnToCalmedState
Descripción	Indica si el NPC debe pasar a un estado de calma
Resultado Éxito	Si el NPC debe cambiar a un estado de calma
Resultado Fallo	Si el NPC aún no debe cambiar a un estado de calma

Nombre	Task_CanShootToPlayer
Descripción	Indica si el NPC puede disparar contra el jugador
Resultado Éxito	Si el NPC puede disparar en el instante de comprobación contra el jugador.
Resultado Fallo	Si el NPC no puede disparar, no lo tiene a tiro o debe apuntar y aún no ha apuntado al jugador.

Nombre	Task_MustAimToShoot
Descripción	Indica si el NPC debe apuntar antes de disparar contra el jugador
Resultado Éxito	Si el NPC tiene la capacidad de apuntar
Resultado Fallo	Si no tiene la capacidad de apuntar

Nombre	Task_CanJumpToAvoidObstacle
Descripción	Indica si, en el estado actual, el NPC puede saltar para evitar un obstáculo.
Resultado Éxito	Si saltando, el NPC podrá esquivar un obstáculo frente a él.
Resultado Fallo	Si saltar no ayudará al NPC a esquivar el obstáculo frente a él.

Como puede observarse, todas estas tareas sirven para realizar la comprobación del estado en que se encuentra el agente inteligente y su utilidad principal será consultar el tipo de tareas que se pueden realizar o si se debe realizar una tarea específica durante la ejecución del Behaviour Tree.

Por otra parte, las tareas de acción son las que, efectivamente, llevan al agente inteligente a realizar una acción determinada o a modificar su estado.

A continuación se muestran las tareas de acción que se han implementado:

Nombre	Task_PreCalculateIsGrounded
Descripción	Tarea para realizar el pre-cálculo que permite al agente conocer si está apoyado en el suelo o no. Al realizar esta tarea, se calcula si el NPC se encuentra apoyado en el suelo.
Resultado Éxito	Siempre. La tarea queda realizada
Resultado Fallo	No aplicable

Nombre	Task_Jump
Descripción	Hace al personaje saltar en función de la fuerza indicada en su NPCConfiguration.
Resultado Éxito	Siempre. La tarea queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_Turn
Descripción	Hace que el NPC cambie el sentido hacia el que está orientado.
Resultado Éxito	Siempre. La tarea queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_WalkForward
Descripción	Hace al NPC moverse hacia adelante a velocidad de andar.
Resultado Éxito	Siempre. La tarea queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_RunForward
Descripción	Hace al NPC moverse hacia adelante a velocidad de carrera.
Resultado Éxito	Siempre. La tarea queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_Stop
Descripción	Hace que el NPC detenga su movimiento.
Resultado Éxito	Siempre. La tarea queda realizada
Resultado Fallo	No aplicable.

Nombre	Task_LookToPlayer
Descripción	Orienta al NPC en la dirección en la que se encuentra el jugador
Resultado Éxito	Siempre. La tarea queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_Aim
Descripción	Hace que el NPC utilice sus armas para apuntar contra el jugador.
Resultado Éxito	Siempre. La tarea queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_Shoot
Descripción	El NPC realiza un disparo de sus armas contra el jugador. Se disparan todas las armas que correspondan.
Resultado Éxito	Siempre. La acción queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_LookOppositeToPlayer
Descripción	Tarea para hacer que el NPC mire en dirección contraria al jugador.
Resultado Éxito	Siempre. La acción queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_SetPatrolWaitingTime
Descripción	Establece un tiempo de espera antes de seguir patrullando.
Resultado Éxito	Siempre. La acción queda realizada.
Resultado Fallo	No aplicable.

Nombre	Task_SetWaitToChaseTime
Descripción	Establece un tiempo que debe esperar el NPC antes de comenzar la persecución del jugador.
Resultado Éxito	Siempre. La tarea queda realizada
Resultado Fallo	No aplicable

Nombre	Task_Alert
Descripción	Tarea para entrar en modo alerta. Como resultado, el personaje también realizará la animación de entrar en estado de alerta.
Resultado Éxito	Siempre. La tarea queda realizada.
Resultado Fallo	No aplicable.

En el caso de las tareas de acción, estas siempre resultan exitosas, puesto que llevan al NPC a realizar una acción y no es posible que fallen. Esto se debe a que, por ejemplo, una vez se realiza el salto, no se espera que el propio salto quede cancelado.

Behaviour Tree (topología):

Con las tareas mostradas en la sección anterior se puede definir el Behaviour Tree. Esta será la estructura de datos en la que quedará almacenado el comportamiento final que exhibirá el agente inteligente.

Para ilustrar este comportamiento se mostrará el código asociado a cada uno de los árboles que se representan y se explicará el comportamiento resultante del árbol de ejecución.

Así mismo, el comportamiento global del personaje, por motivos de simplicidad y para facilitar el análisis y proceso de detección de errores, ha quedado dividido en distintos sub-trees que se representarán uno a uno en esta documentación.

- **tree("root"):**

Descripción:

Este árbol de ejecución determinará si el agente inteligente puede ejecutar alguno de los sub-árboles de comportamiento y en su caso, seleccionará cuál de dichos árboles ejecutar.

Uno de los aspectos más importantes a tener en cuenta en este árbol es el hecho de que, en caso de que el personaje no esté vivo, no se realiza ninguna acción, por lo que no se seleccionará ningún sub-tree de los posibles que se podrían ejecutar.

Definición DSL:

```
// -----
// Root decision tree:
// This tree will decide if the character can execute any
// of the available subtrees and, in that case, will take
// the decision about which tree to execute
// -----
tree("Root")
  while
    Task_IsAlive
    sequence
```

```

Task_PreCalculateIsGrounded
fallback
while
    not Task_HasPlayerBeenDetected
        tree("CALMED_BEHAVIOR")
    while
        Task_HasPlayerBeenDetected
            tree("IN_COMBAT_BEHAVIOR")

```

Representación gráfica del árbol de decisiones:

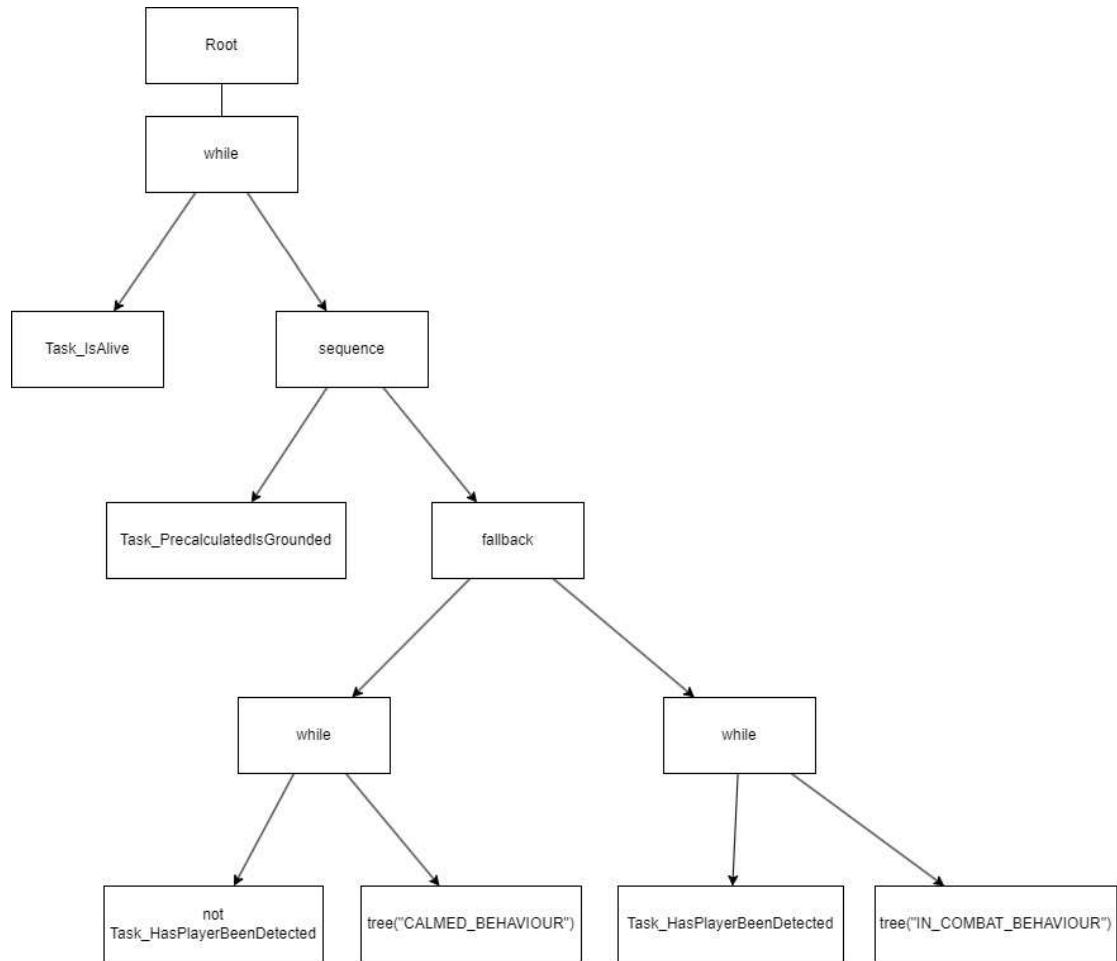


Imagen 109. Representación gráfica del árbol de decisiones root

- **tree("CALMED_BEHAVIOR")**

Descripción:

Este sub-tree define el comportamiento que tendrá el NPC cuando se encuentra en un estado de calma. En este estado, el NPC tomará distintas acciones. Por una parte, si detecta al jugador usando sus sensores se detendrá de inmediato, mirará al jugador, y pasará al modo alerta.

Si por otra parte, el jugador no está visible, patrullará por la zona según su configuración o permanecerá estático mirando a su alrededor.

Definición DSL:

```
// -----
// Calmed behavior tree:
// This sub-tree will define how must the character behave
// if he is in a calmed situation
// -----
tree("CALMED_BEHAVIOR")
fallback
// Check if the player is detected
while
Task_IsPlayerDetected
sequence
    Task_Stop
    Task_LookToPlayer
    Task_Alert
// Otherwise - patrol or idle
while
Task_IsPatrollerNPC
fallback
fallback
// 1.1: The NPC can let himself fall from the platforms
while
sequence
    Task_CanNPCJumpOnCalmedState
    Task_CanMoveForwardIgnoringFall
sequence
    Task_WalkForward
    Task_SetPatrolWaitingTime

// 1.2: The NPC cannot let himself fall from the platforms
while
sequence
    Task_CanMoveForward
    not Task_CanNPCJumpOnCalmedState
sequence
    Task_WalkForward
```

```

Task_SetPatrolWaitingTime
// 2: Character cannot move forward: Wait and change direction
while
not Task_CanMoveForward
sequence
    Task_Stop
    not Task_MustNPCWait
    Task_Turn
    Task_SetPatrolWaitingTime
// If the NPC cannot patrol, stay idle
while
not Task_IsPatrollerNPC
sequence
    Task_Stop
    while
        Task_MustLookAround
        while
            not Task_MustNPCWait
            sequence
                Task_Turn
                Task_SetPatrolWaitingTime

```

Representación gráfica del árbol de decisiones:

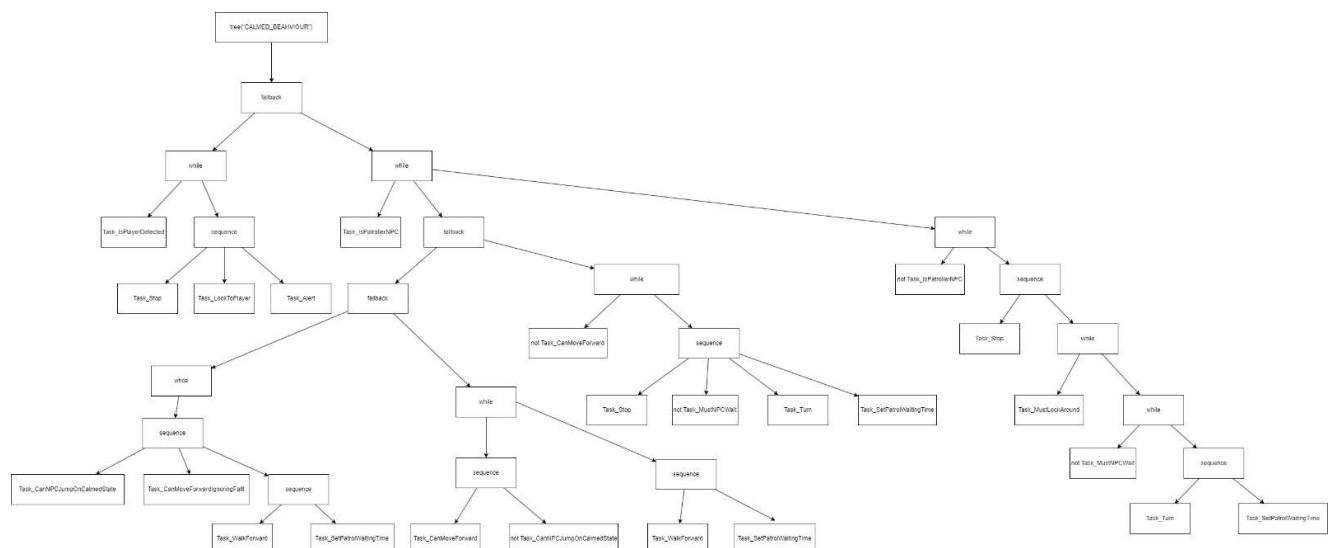


Imagen 110. Representación gráfica del árbol de decisiones CALMED_BEHAVIOR

- `tree("IN_COMBAT_BEHAVIOR")`

Descripción:

Se trata del árbol de ejecución que se ejecutará cuando el NPC entra en combate. En este estado, el NPC tratará de atacar al jugador, perseguirlo, disparar y perseguirlo, huir de él, etc.

El comportamiento que exhibirá aquí dependerá de su NPCConfiguration y sus capacidades.

Así mismo, dada la elevada complejidad de algunas de las acciones que se ejecutarían en este sub-tree, se ha optado por crear árboles de decisión auxiliares que ayudan a exhibir determinados comportamientos.

Definición DSL:

```
// -----
// In combat behavior tree:
// This sub-tree defines how must the character behave if
// a combat situation is reached (the player is detected)
// -----

tree("IN_COMBAT_BEHAVIOR")
fallback
Task_CheckReturnToCalmedState
fallback
// Case 1: Chase the player and shoot to him if the NPC can
while
Task_CanChasePlayer
sequence
// Case 1: Chase the player and stop (if necessary) at
// the maximum stop distance
tree("AUXILIARYCHASE_WITH_STOP_DISTANCE")
// Case 2: Must attack from the distance
tree("AUX_AIM_AND_SHOOT")

// Case 2: NPC can not chase. If it can attack, do it
while
not Task_CanChasePlayer
tree("AUX_AIM_AND_SHOOT")

// Case 3: Run away from the player
tree ("AUXILIARY_RUN_AWAY_FROM_CHARACTER")
```

Representación gráfica del árbol de decisiones:

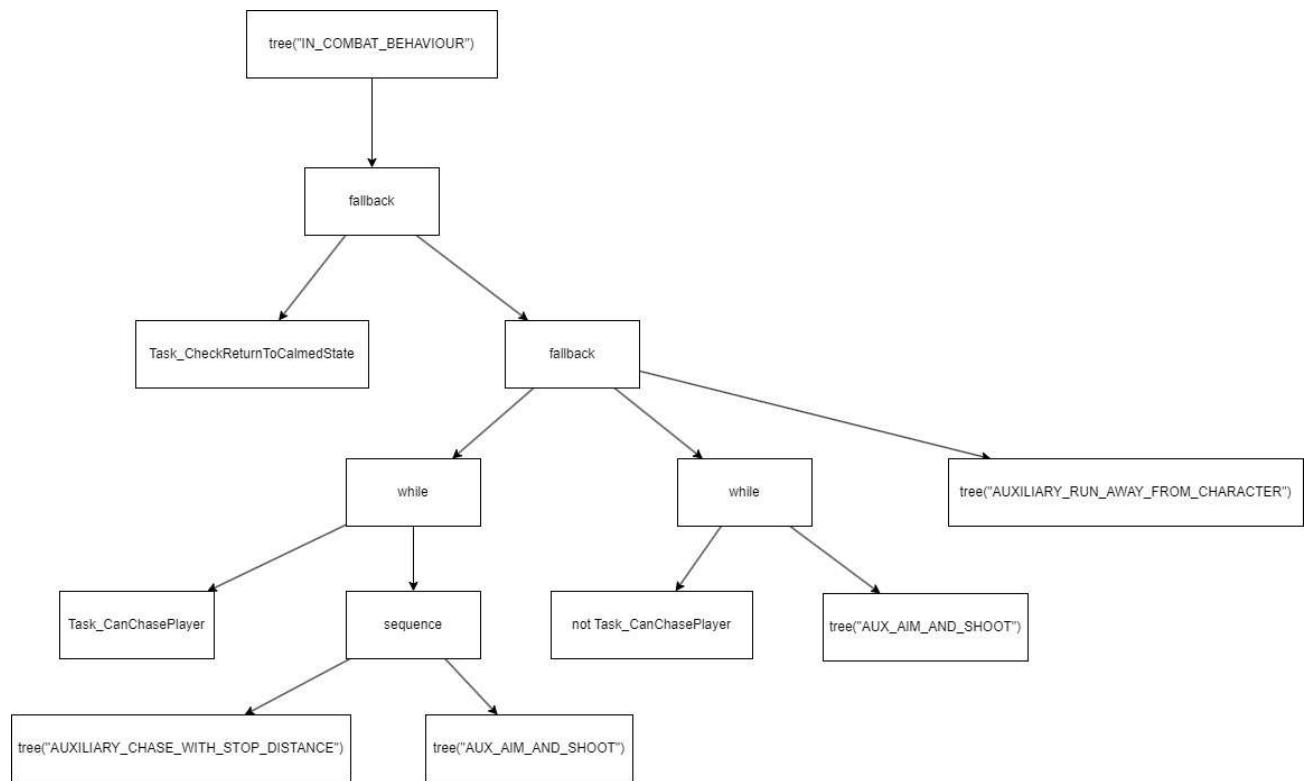


Imagen 111. Representación gráfica del árbol de decisiones IN_COMBAT_BEHAVIOUR

- `tree("AUXILIARY_RUN_AWAY_FROM_CHARACTER")`

Descripción:

Árbol auxiliar que define el comportamiento que hace al NPC huir del jugador.

Este árbol de comportamiento se ejecuta comprobando inicialmente si el jugador puede huir y, en caso afirmativo, hará que el NPC se oriente en dirección contraria a donde está el jugador y corra, salte si puede y huya del mismo.

Definición DSL:

```
// -----
// Auxiliary tree to make the character run away from the
// player
// -----
tree ("AUXILIARY_RUN_AWAY_FROM_CHARACTER")
while
    // Case 2: The NPC must run away from the player
    Task_MustRunAwayFromPlayer
    sequence
        Task_LookOppositeToPlayer
        fallback
        while
            Task_CanNPCJumpOnCombatState
            fallback
            while
                Task_CanMoveForwardIgnoringFall
                Task_RunForward
                while
                    Task_CanJumpToAvoidObstacle
                    Task_Jump
            while
                not Task_CanNPCJumpOnCombatState
                while
                    Task_CanMoveForward
                    Task_RunForward
                    Task_Stop
```

Representación gráfica del árbol de decisiones:

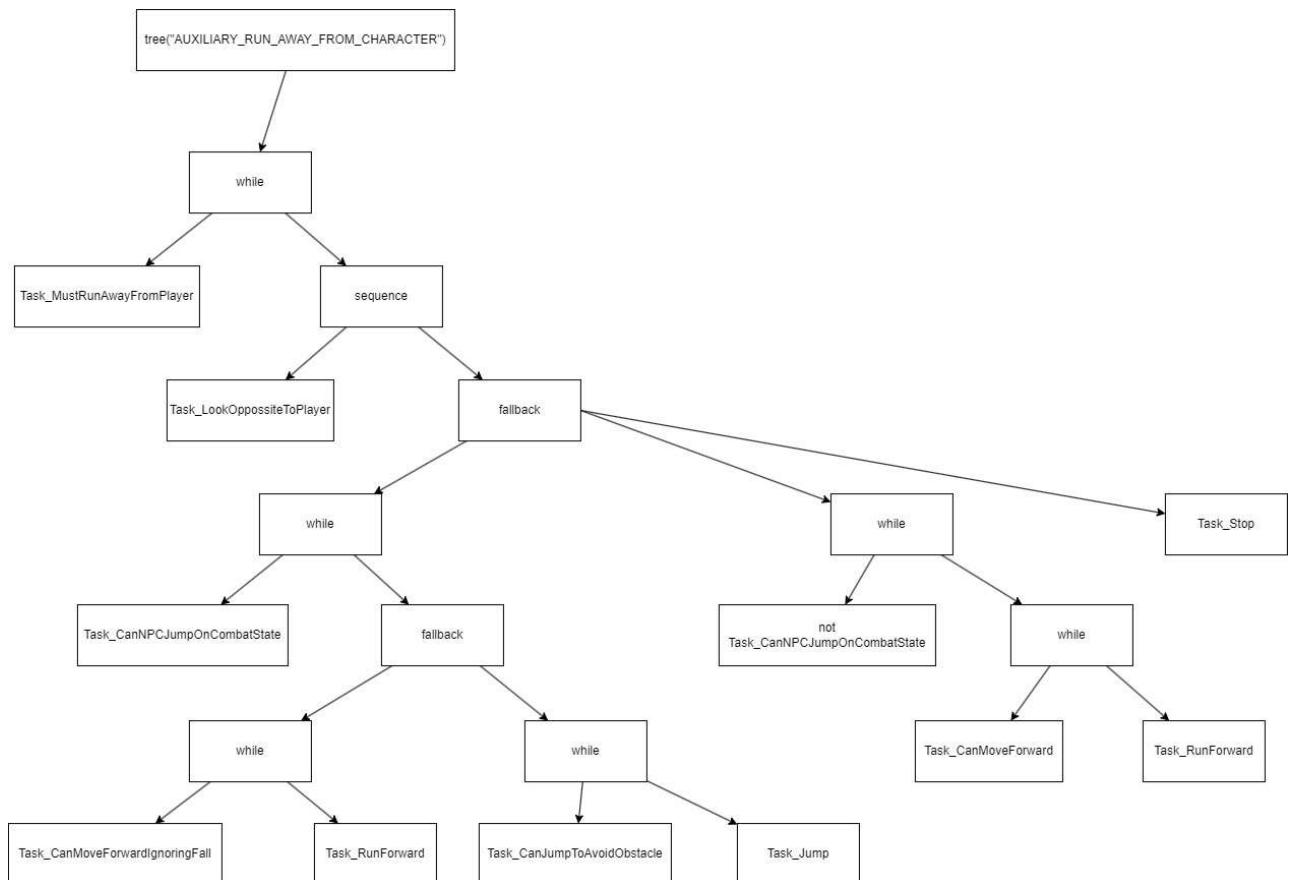


Imagen 112. Representación gráfica del árbol de decisiones

AUXILIARY_RUN_AWAY_FROM_CHARACTER

- `tree("AUXILIARYCHASEWITHSTOPDISTANCE")`

Descripción:

Este árbol auxiliar permite al NPC perseguir al jugador y parar a la distancia adecuada que tiene asignada para ello. Al mismo tiempo, si mientras lo persigue, el NPC tiene la capacidad de saltar para superar obstáculos, realizará el salto.

Definición DSL:

```
// -----
// Auxiliary tree to make the character CHASE the player
// and stop at the stopping distance that this character
// has assigned
// -----
tree("AUXILIARYCHASEWITHSTOPDISTANCE")
fallback
while
  Task_IsCharacterAtStopDistance
  sequence
    Task_LookToPlayer
    Task_Stop
while
  // Case 1: The NPC must chase the player
  Task_CanChasePlayer
  sequence
    Task_LookToPlayer
    fallback
      // 1.1. Chase the player Jumping (or falling) if necessary
      while
        Task_CanNPCJumpOnCombatState
        fallback
          while
            Task_CanMoveForwardIgnoringFall
            Task_RunForward
            while
              Task_CanJumpToAvoidObstacle
              Task_Jump
  // 1.2. Chase the player, don't jump or fall from the platform
  while
    not Task_CanNPCJumpOnCombatState
    while
      Task_CanMoveForward
      Task_RunForward
  // 1.3. Don't move more as the NPC cannot jump or fall from the platforms
  Task_Stop
```

Representación gráfica del árbol de decisiones:

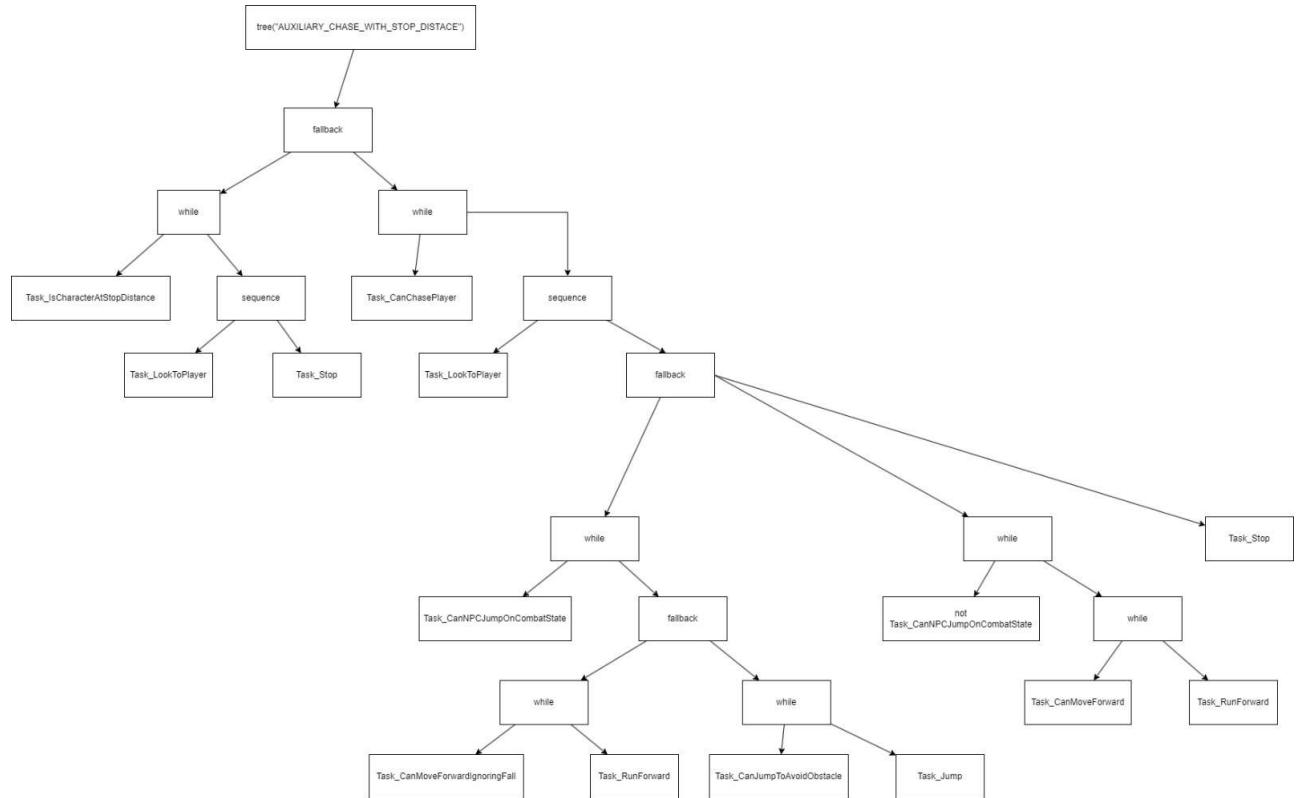


Imagen 113. Representación gráfica del árbol de decisiones

AUXILIARY_CHASE_WITH_STOP_DISTANCE

- `tree("AUX_AIM_AND_SHOOT")`

Descripción:

Árbol auxiliar que permite al NPC apuntar y disparar contra el jugador. En este árbol, el NPC se orientará hacia el jugador, si debe apuntar, tratará de apuntar y si puede disparar realizará el disparo.

Cabe destacar que antes de realizar el disparo se comprueba si el NPC en cuestión puede disparar. Esto se realiza porque, cada vez que se dispara, se ajusta un tiempo de enfriamiento del arma durante el cuál, no se podrá volver a disparar

Definición DSL:

```
// -----
// Auxiliary subtree to make the character aim and shoot
// -----
tree("AUX_AIM_AND_SHOOT")
sequence
sequence
// Case 3: Must attack from the distance
Task_LookToPlayer
fallback
sequence
while
    Task_MustAimToShoot
    Task_Aim
while
    Task_CanShootToPlayer
    Task_Shoot
while
    Task_CanShootToPlayer
    Task_Shoot
```

Representación gráfica del árbol de decisiones:

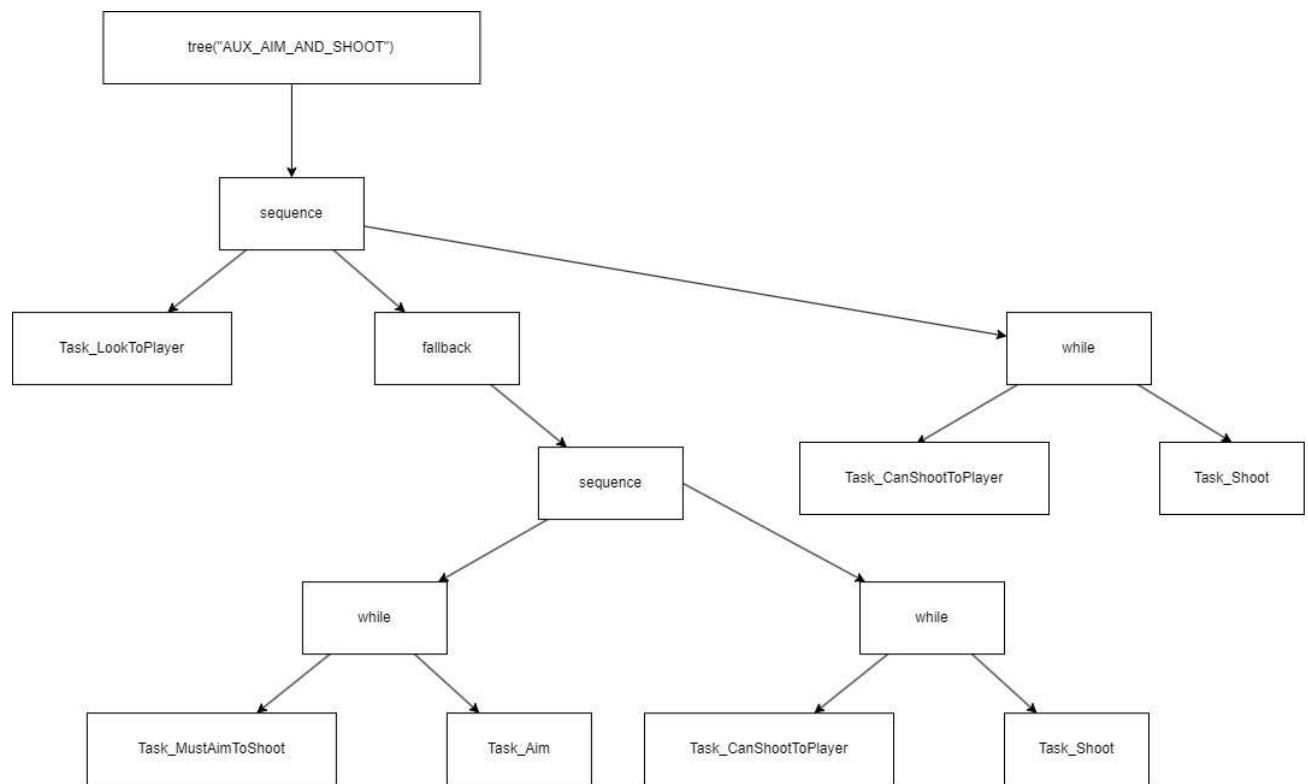


Imagen 114. Representación gráfica del árbol de decisiones AUX_AIM_AND_SHOOT

2.8 Pruebas de Calidad

A lo largo del desarrollo se han ido realizando diferentes tipos de pruebas para asegurar que el software funciona como se espera y responde correctamente. Estas pruebas se han realizado en las diferentes iteraciones del juego a medida que evolucionaba la complejidad de este.

- **Pruebas unitarias:** Las pruebas unitarias son las que se realizan para comprobar la funcionalidad de la unidad más pequeña que conforma el software, son las que realizan los programadores sobre sus propios bloques de código para asegurar su funcionamiento.
- **Pruebas de diseño de niveles:** El tester, sin conocer el diseño e intención de los niveles nuevos, los prueba para acercarse a la experiencia que tendría un jugador primerizo al encontrarse en ese entorno. Este tipo de prueba permite modificar cualquier parte del diseño para facilitar su lectura a ojos externos al desarrollo.
- **Pruebas de colisiones:** Durante la partida el tester hace colisionar contra todas las superficies del mapa al personaje para asegurar que éstas funcionan correctamente.
- **Pruebas funcionales:** Se comprueba que lo que hace el juego se corresponde con las funciones implementadas.
- **Pruebas de navegación:** Se navega a través de las diferentes pantallas del juego, accediendo de diferentes maneras, para comprobar que la navegación entre escenas es fluida y funciona correctamente.
- **Pruebas de compatibilidad:** Se comprueba en distintos tipos de dispositivo (PC, móvil, tableta) y controladores (teclado, mando) que la aplicación de juego funciona fluidamente y el visionado de todos los elementos gráficos en todos ellos es correcto.

BIBLIOGRAFÍA

Animum Creativity Advanced School (2022). Materiales y texturas: introducción al shading en 3D. [En línea].

Disponible:

<https://www.animum3d.com/blog/materiales-y-texturas-introduccion-al-shading-en-3d/>

[Acceso: Jul. 8, 2023].

Centro Universitario de Tecnología y Arte Digital. (2023). ¿Qué son los juegos casuales? Diez recomendaciones para móvil y PC. [En línea].

Disponible:

<https://u-tad.com/juegos-casuales-diez-recomendaciones>

[Acceso: Mar. 7, 2023].

D. González Jiménez, *Arte de Videojuegos. Da forma a tus sueños*. Madrid: Ra-Ma, 2014.

E. Carrasco Rosado (2022, Jul. 21). El UV mapping, ese gran desconocido, Animum Creativity Advanced School. [En línea]

Disponible:

<https://www.animum3d.com/blog/el-uv-mapping-ese-gran-desconocido/>

FILLIBsPix, “Pixel art icon set botones videojuego” *freepik*. Sin fecha de publicación. [JPG].

Disponible:

https://www.freepik.es/vector-premium/pixel-art-icon-set-botones-videojuego_26048050.htm

Freepik, “Fondo de panel de control de nave espacial” *freepik*. Sin fecha de publicación. [JPG].

Disponible:

https://www.freepik.es/vector-gratis/fondo-panel-control-nave-espacial_2814233.htm

Meshtint Studio, "Meshtint Free Boximon Fiery Mega Toon Series" *Unity Asset Store*. 2 de octubre de 2019. [unitypackage].

Disponible:

<https://assetstore.unity.com/packages/3d/characters/meshtint-free-boximon-fiery-mega-toon-series-153958>

beffio, "One Minute GUI" *Unity Asset Store*. 16 de septiembre de 2019. [unitypackage].

<https://assetstore.unity.com/packages/2d/gui/one-minute-gui-32346>

Paul IJsendoorn (Friendly Fonts), "Forced Square" *dafont.com*. 26 de abril de 2014. [ttf].

Disponible:

<https://www.dafont.com/es/search.php?q=forced+square>

Unity. Unity Manual, AnimatorControllerParameterType.Trigger. [En línea]

Disponible:

<https://docs.unity3d.com/ScriptReference/AnimatorControllerParameterType.Trigger.html>

Unity. Unity Manual, Blend Trees (Árboles de Mezcla). [En línea].

Disponible:

<https://docs.unity3d.com/es/530/Manual/class-BlendTree.html>

Unity. Unity Manual, Polar Coordinates Node. [En línea].

Disponible:

<https://docs.unity3d.com/Packages/com.unity.shadergraph@6.9/manual/Polar-Coordinates-Node.html>

Unity. Unity Manual, Time.timeScale. [En línea].

Disponible:

<https://docs.unity3d.com/ScriptReference/Time-timeScale.html>

Steve Rabin, *Game AI Pro: Collected Wisdom of Game AI Professionals*, 2021. [En línea]

Disponible:

<http://www.gameaiopro.com/>

Unity AI Game Programming, Second Edition [libro]

ISBN 978-1-78528-827-2

Importante Classes, Vectors [online]
Disponible: <https://docs.unity3d.com/Manual/VectorCookbook.html>
[Acceso: Ene. 6, 2023]

Unity Inverse Kinematics [online]
Disponible:
<https://docs.unity3d.com/Manual/InverseKinematics.html>

ANEXOS

1. Programario

Motor de juegos: Unity 2021.3.3f1

Texturas:

- Adobe Photoshop 2022
- Adobe Illustrator 2022

Diseño gráfico:

- Adobe Photoshop 2022
- Adobe Illustrator 2022
- Canva versión gratuita

Modelado: Blender 2.90

Mapeado de UV: Blender 2.90

Diagramas: Draw.io