ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DOBLE GRADO EN DISEÑO Y DESARROLLO DE VIDEOJUEGOS E INGENIERÍA DE COMPUTADORES

Trabajo Fin de Grado de Diseño y desarrollo de videojuegos

Curso Académico 2022/2023

DESARROLLO DE VIDEOJUEGO COMPLETO HACIENDO USO DE INTERFAZ PROPIA PARA EL CONTROL DEL JUEGO CON UN CUBO DE RUBIK

Autor: Carlos Chamizo Cano

Tutor: Dan Casas Guix

Resumen

En este proyecto, se desarrolla un videojuego completo donde se hace uso de una interfaz entre un cubo de Rubik inteligente y Unity desarrollada para el trabajo de fin de grado de ingeniería de Computadores.

En esta memoria se detallan las diferentes partes que forman el juego final. En primer lugar, se hace un Documento de diseño del juego donde se explica toda la información respecto a la jugabilidad narrativa y mecánicas. Posteriormente, se desarrolla el trabajo realizado en los diferentes elementos del juego, entre los que está el sistema de guardado y carga de partida, el uso de la interfaz entre el cubo de Rubik y el juego, el diseño e implementación del personaje jugable, la implementación de las mecánicas principales, formas en las que se ha optimizado el juego y finalmente el trabajo realizado en arte y conceptos.

El juego desarrollado para este proyecto se puede jugar descargándolo de este ENLACE.

Agradecimientos

Me gustaría agradecer en primer lugar a mi tutor, Dan Casa Guix por tutorizarme este trabajo de fin de grado, así como por su tiempo y sus consejos.

Gracias a mi familia, a mis padres por apoyarme y brindarme de todos los recursos necesarios además de a mi hermana y a Edgar por su apoyo e interés en mi proyecto.

Gracias a mi madrina que me cuida desde aquí con cariño y comprensión y a mi padrino que sé que está muy orgulloso de mí allí arriba.

Gracias a todo el resto de mi familia que han contribuido de alguna u otra forma.

Gracias a mis mascotas Nana y Lilly por el apoyo emocional e incondicional en los momentos más duros.

Gracias a todos mis amigos que me han apoyado y en específico a Javi por ayudarme a soldar el circuito.

Gracias a mi novia Andrea por apoyarme en los momentos más difíciles y ayudarme a seguir adelante con los proyectos.

Y finalmente gracias a todos los que estáis leyendo esto, espero que la lectura os sea leve.

Contenidos

Índice de Ilustraciones

Índice de tablas

1. Objetivos

El objetivo de este trabajo de fin de grado es el de aplicar y demostrar los conocimientos aprendidos a lo largo de las diferentes asignaturas cursadas en la carrera, realizando un proyecto que cumpla los siguientes objetivos:

- Hacer uso de la Interfaz de Cubo de Rubik desarrollado en el trabajo de fin de grado de Ingeniería de Computadores de forma que se cumplan los siguientes requisitos:
    - El juego debe poder jugarse en su totalidad con el cubo de Rubik.
    - El juego debe explicar al usuario de qué forma usar el cubo de Rubik para juagar.
- Creación de un videojuego desde cero pasando por todas las fases de desarrollo. Este videojuego debe cumplir los siguientes requisitos:
    - El juego debe poder ser jugable para una persona que no tenga un Smart Cube.
    - El juego debe tener un trabajo de Concept y modelado
    - El control del personaje principal debe ser cómodo y satisfactorio.
    - El juego debe tener un sistema de guardo de juego.
    - Debe ser fluido en cualquier ordenador moderno.
    - Debe tener enemigos gestionados por un comportamiento.
    - Debe tener una historia, así como un sistema de diálogos.
    - El proyecto debe estar bien organizado

Para realizar este proyecto, se ha usado como software principal UNITY, en cuanto al entrono de desarrollo, el proyecto se ha llevado a cabo en Rider. Por otro lado, respecto al software utilizado para la creación de arte, se ha utilizado Procreate para los elementos en 2D y Blender junto con Substance Painter para los elementos 3D.

2. Motivación

Desde que empecé la carrera y fui comprendiendo las diferentes partes que forman un videojuego, me fui interesando cada vez más en el mundo del desarrollo de videojuegos.

Hacia primero de la carrera, me propuse aprender a dibujar, y con ello, me inicié en el mundo del arte de los videojuegos, principalmente en lo que a la ilustración se refiere. Posteriormente entre segundo y tercero de carrera, con los conocimientos de animación 3D, me inicié en ese aspecto e incluso me animé a formar parte con un equipo de amigos donde hicimos un juego de miedo donde realicé el escenario [1].

Finalmente, sobre tercero y cuarto con asignaturas como Animación 3D, Realidad Virtual, Algoritmos y comportamiento de personajes, me empecé a interesar más profundamente en la programación, especializándome en el Gameplay de los personajes principales. Incluso me atreví a participar en lagunas GameJams como programador.

Es por ello por lo que tras todo lo aprendido y experimentado, me propuse la creación de este videojuego. Donde quería demostrar que es posible desarrollar un juego atractivo, divertido y fácil de entender utilizando un cubo de Rubik como control por una persona. Además, quería asegurarme de que el hecho de poseer un cubo inteligente o no fuera una limitación para disfrutar del juego, es decir, quería crear una experiencia de juego única pero accesible para todos.

Además, otra parte de mi motivación era demostrar mi capacidad para desarrollar un videojuego completo con una historia ya que nunca había realizado esta tarea.

Al abordar todos los aspectos del desarrollo de un videojuego, quería también poder descubrir qué área específica me apasiona más habiendo trabajado en un entorno más serio y de esta forma ver en qué me gustaría especializarme una vez que haya completado mis estudios.

3. Estado del Arte

3.1 Introducción

En este apartado se hablará sobre el marco teórico sobre el que se basa este trabajo de fin de grado para llegar hasta el resultado final. Primero, se explicará la historia de los videojuegos desde sus inicios hasta la actualidad. Se hará una clasificación de los diferentes géneros de videojuegos. Además, se concretará en el género de los videojuegos de Acción-Aventura ya que a este es el género al que pertenece el videojuego que se ha desarrollado.

Después, se hará un breve recorrido por la historia de los mandos que han ido surgiendo a lo largo de los años, así como de mandos raros o poco comunes con los que usuarios de internet han disfrutado de juegos que se podrían catalogar como difíciles.

3.2 Videojuegos

3.2.1 ¿Qué es un videojuego?

Como define la RAE[1] [2] un videojuego es un juego electrónico que se visualiza a través de una pantalla. Sin embargo, esta es una definición que no llega a representar todo lo que un videojuego puede llegar a representar. Un videojuego, supone mucho más que una representación visual en una pantalla, un videojuego supone una experiencia, supone una la narración de una historia y sobre todo supone también un arte.

Sin embargo, para concretar una definición de videojuego, primero hay que pensar en cuales son aquellos elementos que hacen que un videojuego sea considerado un videojuego.

En primer lugar, un videojuego es una forma de entretenimiento interactivo, es decir que el jugador debe ser capaz de controlar de alguna forma aquello que ve por pantalla. De lo contrario pasaría a ser un mero video y por lo tanto entraría en otras categorías como puede ser los videos o las películas.

Un videojuego debe suponer un reto, es decir que debe imponer algún tipo de desafío, es decir, un juego debe ofrecer al jugador una motivación al jugador para seguir jugando, ya sea llegar a un lugar, acabar con una horda de enemigos o descubrir un misterio.

Además, un videojuego es un arte. Dentro de los videojuegos hay muchos tipos de manifestaciones del arte, ya puede ser a través del dibujo, el modelado, la animación, música, el guion o el propio "Gameplay" en sí mismo. Y debido a que la definición de arte es muy subjetiva ya que dependiendo del observador esta cambia, en los videojuegos pasa lo mismo debido a la infinita cantidad de posibilidades que un videojuego puede ofrecer.

Por lo tanto, un intento de definir un videojuego es el siguiente. Un videojuego es una forma de entretenimiento interactivo digital donde se le presenta al jugador un desafío y el cual también supone una forma de expresión a través de sus apartados gráficos y/o sonoros.

[1] Real Academia de la lengua

3.2.2 Historia de los videojuegos

3.2.2.1 Los inicios

Tal y como se explica en [3], los primeros videojuegos se desarrollaban en aparatos electrónicos de propósito científico. En sus inicios, los videojuegos surgieron como experimentos y aplicaciones de entretenimiento en dispositivos como osciloscopios y computadoras utilizadas con fines científicos y académicos.

Estos aparatos electrónicos de propósito científico, como el osciloscopio y la computadora PDP-1, ofrecieron una plataforma para los ingenieros y entusiastas experimentaran con la creación de juegos interactivos. Estos primeros videojuegos eran simples y a menudo se limitaban a gráficos básicos y mecánicas de juego rudimentarias.

"Tennis for Two" [4] fue creado por W. Higinbotham en 1958. Este juego pionero fue una simulación muy simple de una partida de tenis en un osciloscopio. Los jugadores podían controlar cada raqueta a través de perillas y golpear la pelota de tenis virtual de un lado a otro de la pantalla. Aunque su concepto era básico, "Tennis for Two" sentó las bases para los futuros videojuegos y demostró el potencial de la interactividad electrónica en la recreación de actividades deportivas. Es considerado uno de los primeros videojuegos de la historia y allanó el camino para el desarrollo de la industria del entretenimiento interactivo tal como la conocemos hoy en día.

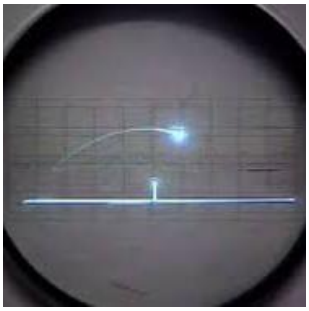


Ilustración 1: Tennis for Two

"Spacewar!" Este juego revolucionario fue diseñado para dos jugadores, quienes se enfrentaban en una batalla espacial mientras intentaban evitar ser absorbidos por la fuerza gravitacional de una estrella. "Spacewar!" fue el primer juego en implementar cálculos complejos, ofreciendo una experiencia de juego dinámica.
Debido a su gran éxito, Digital Equipment, la compañía fabricante de la computadora PDP-1, decidió incluir "Spacewar!" como parte del código fuente de la computadora.

Ilustración 2: Spacewar!

Tabla 1: juegos del inicio

3.2.2.2 La década de los 70

En los años 70, comenzaron a comercializarse las primeras máquinas recreativas, y en ese tiempo, Taito y Atari se destacaron como las principales empresas en la industria de los videojuegos.

Tanto Taito como Atari jugaron un papel fundamental en la creación y difusión de las máquinas recreativas en los años 70. Sus juegos revolucionarios y su visión innovadora allanaron el camino para el auge de la industria de los videojuegos en las décadas siguientes.

“PONG”: En 1972 llega uno de los juegos más icónicos que dan inicio a la industria de los videojuegos, el “PONG”. Este surgió en Atari, creado por Allan Alcorn [5] como un ejercicio de formación. Este consiste en una partida similar al tenis donde se consiguen puntos cuando el adversario no es capaz de devolver la pelota. Este devido a su simpleza y éxito entre el público se convirtió en el primer juego comercial exitoso.

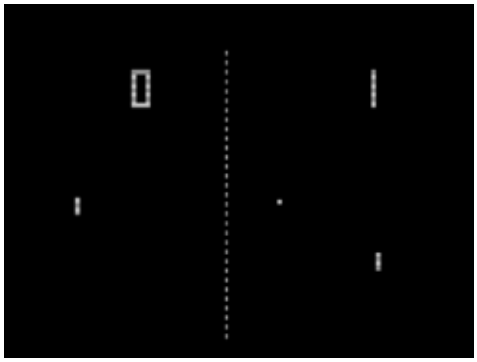


Ilustración 3: Pong

“Asteroids” es un juego desarrollado por Atari en 1979. En este juego, los jugadores controlan una nave espacial que debe esquivar los asteroides que se mueven por el entorno y destruirlos por medio de disparos. "Asteroids" fue innovador en su época al introducir físicas realistas, como la inercia, lo que añadía un desafío adicional para los jugadores.

Ilustración 4: Arcade Asteroids

“Space Invaders” es un juego desarrollado por Taito en 1978. En este clásico arcade, los jugadores controlan una nave espacial que debe protegerse de una horda de aliens invasores. El objetivo es disparar a estos aliens mientras se desplazan de un lado a otro en formaciones y se acercaban cada vez más cerca de la nave del jugador. Además, el jugador cuenta con unos bloques que actúan como escudos para protegerse de los disparos enemigos. A medida que se destruyen más enemigos, la velocidad del juego aumenta, añadiendo un desafío creciente.

Ilustración 5: Space Invaders

Tabla 2: juegos de los años 70

3.2.2.3 La década de los 80

Los años 80 marcaron un período importante en la historia de los videojuegos. Comenzando con el auge de las máquinas recreativas en los años 70, la primera mitad de la década de los 80 fue considerada la edad de oro de las máquinas recreativas o Arcade. Durante este tiempo, surgieron juegos muy exitosos como "Pacman" y "Pole Position".

Sin embargo, en la segunda mitad de los años 80, las máquinas recreativas comenzaron a declinar debido a la apuesta de Nintendo por las consolas. En 1983, Nintendo lanzó al mercado la NES[2], aunque no llegó a Europa hasta 1986 y a América en 1985. La NES introdujo algunos de los juegos más icónicos de todos los tiempos, como "Super Mario Bros" o "The Legend of Zelda".

Además, durante los años 80 también se popularizó la rama de las consolas portátiles con el lanzamiento de la Game Boy en 1989.

“Pac-Man”, lanzado en 1980 por Namco. En este juego, los jugadores controlan a un personaje amarillo, cuyo objetivo es comer todas las bolas blancas posibles dentro de un laberinto.

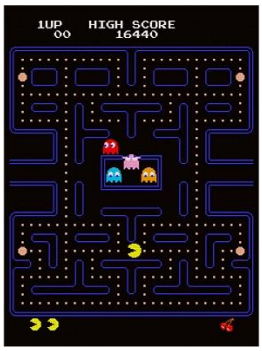


Ilustración 6: Pac Man

[2] Nintendo Entertainment System

“Pole Position” es un juego de carreras lanzado por Atari en 1982. Este clásico título simulaba una jornada de Fórmula 1. El juego se dividía en dos fases principales: la primera era una contrarreloj, en la cual los jugadores debían completar una vuelta en un tiempo determinado para clasificar para la carrera principal y en caso de lograrlo, competirían contra otros coches en la carrera.

Ilustración 7: Pole position

“Super Mario Bros” es un juego de plataformas lanzado por Nintendo en 1985. En este juego, los jugadores asumen el papel de Mario. El objetivo principal del juego es llegar al final de cada nivel, superando plataformas, evitando enemigos y recolectando monedas. El objetivo final es rescatar a la princesa Peach.

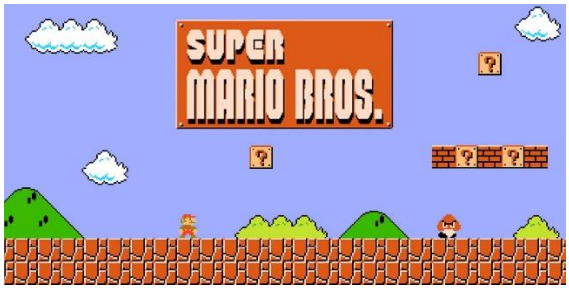


Ilustración 8: Super Mario Bros

“The Legend of Zelda” es un juego de acción-aventura lanzado por Nintendo en 1986. En este título, los jugadores asumen el papel de Link. El juego se desarrolla en un mundo abierto donde Link debe explorar diferentes áreas, como mazmorras, bosques y cuevas, mientras se enfrenta a enemigos y resuelve acertijos para avanzar en la historia.

Ilustración 9: The legend of Zelda

Tabla 3: juegos de los años 80

3.2.2.4 La década de los 90

Los primeros años de los años 90 marcaron un gran salto tecnológico en la industria de los videojuegos. Se produjo una transición importante de las consolas de 8 bits a las consolas de 16 bits, lo que brindó una mayor potencia gráfica y mejoras significativas en los juegos.

Unos años más tarde se volvió a dar otro salto tecnológico el cual permitió que los juegos en dos dimensiones quedaran en segundo plano gracias al lanzamiento de consolas de 32 bits como la Sony “PlayStation” y la “Sega Saturn” introduciendo un mayor catálogo de juegos 3D. Estas consolas ofrecían gráficos tridimensionales más avanzados y una mayor capacidad de procesamiento.

Posteriormente, se lanzaron consolas de 64 bits, como la “Nintendo 64” y la “Atari Jaguar”, que continuaron impulsando el poder y la calidad de los gráficos en los videojuegos. Estas consolas aprovecharon al máximo las capacidades del 3D.

Al mismo tiempo, en el mundo de los ordenadores, se comenzaron a desarrollar las primeras aceleradoras gráficas. Estos dispositivos permitían mejorar el rendimiento gráfico de los juegos en PC, brindando efectos visuales más sofisticados y una mayor calidad de imagen.

“DOOM” es un icónico juego de disparos en primera persona lanzado en 1993 por id Software. En este juego, los jugadores asumen el papel de un marine espacial que se encuentra en una instalación marciana infestada de criaturas. El objetivo principal es abrirse paso a través de niveles laberínticos y enfrentarse a múltiples hordas de enemigos a un ritmo frenético.

Ilustración 10 : DOOM

“Resident Evil” es un juego de survival horror lanzado en 1996 por Capcom. En este juego, los jugadores se sumergen en una atmósfera tenebrosa y aterradora mientras investigan una mansión infestada de horrores y misterios.


Ilustración 11: Resident Evil

“Super Mario 64”, lanzado en 1996 por Nintendo, es un juego de plataformas 3D siendo uno de los primeros títulos verdaderamente exitosos en su género.
El objetivo principal es recolectar suficientes estrellas, dispersas en los diferentes niveles, para abrir nuevas áreas y enfrentarse al jefe final, Bowser.
Además, el juego ha dejado un impacto duradero en la comunidad de jugadores al ser uno de los pioneros en la disciplina del Speedrun[3].

Ilustración 12: Super Mario 64

[3] Un speedrun consiste en acabar un juego de la forma más rápida posible.

Tabla 4 : juegos en los años 90

3.2.2.5 La década de los 2000

Sin duda, la década de los 2000 fue testigo de numerosos cambios e innovaciones en la industria de los videojuegos, abarcando diversas áreas y facilitando el acceso a nuevos desarrolladores. Además, la mejora tecnológica tuvo un impacto significativo en la evolución de los juegos.

El inicio de la década fue marcado por el lanzamiento de la PlayStation 2, que se convirtió en la consola más vendida de la historia. Sega también hizo su competencia con la Dreamcast, mientras que Nintendo lanzó la GameCube y Microsoft presentó la Xbox.

El auge de los juegos en línea fue otro hito importante. Gracias a mejoras en el acceso a internet y en las consolas y computadoras, los juegos multijugador se volvieron más accesibles para una gran cantidad de personas. Esto también allanó el camino para la aparición de los e-sports[4], con juegos como "League of Legends" y "Counter-Strike: Global Offensive" liderando el camino en competiciones profesionales.

La llegada de los teléfonos inteligentes abrió un nuevo mercado para los juegos, permitiendo que cualquier persona pudiera acceder a juegos desde sus dispositivos móviles. Esto amplió enormemente la audiencia de los videojuegos, ya que se volvieron accesibles en cualquier lugar y momento.

Además, se produjo un notable aumento en el número de estudios independientes (indies) en la industria. La disponibilidad de motores gráficos accesibles como Unity y Unreal Engine permitió a desarrolladores independientes crear juegos de manera más fácil y económica. Esto dio lugar a una mayor diversidad y experimentación en el catálogo de juegos.

"League of Legends" es un juego multijugador en línea lanzado en 2009 por Riot Games. En este juego, dos equipos de cinco jugadores cada uno se enfrentan en un campo de batalla virtual con el objetivo de destruir el nexo del equipo contrario.

Ilustración 13: League of Legends

[4] Deportes electrónicos

“Fallout 3” es un juego de rol (RPG) desarrollado por Bethesda y lanzado en 2008. Ambientado en un futuro postapocalíptico, el juego ofrece a los jugadores una experiencia inmersiva en un mundo devastado por una guerra nuclear.

Ilustración 14: Fallout 3

“Angry Birds” es un popular juego lanzado por Rovio Entertainment en 2009 para Android y iOS.
Los jugadores asumen el control de diferentes tipos de pájaros que buscan vengarse de unos cerdos verdes. El objetivo principal es a los pájaros contra estructuras y cerdos enemigos, destruyéndolos y superando cada nivel.

Ilustración 15: Angry Birds

"**Minecraft**" es un juego desarrollado de forma independiente por Markus Persson, también conocido como "Notch" y lanzado en 2009.
En "Minecraft", los jugadores son transportados a un mundo virtual generado procedural mente, donde se les otorga una libertad absoluta para explorar, construir y sobrevivir. El juego es un sandbox, lo que significa que los jugadores tienen la capacidad de moldear y transformar el mundo a su antojo.

Ilustración 16 : Minecraft

Tabla 5: juegos de los años 2000

3.2.2.6 Actualidad

En la actualidad, la industria de los videojuegos ha experimentado mejoras continuas en el hardware y la tecnología en general. Si bien esto ha llevado al desarrollo de juegos cada vez más potentes y exigentes en términos de recursos, también ha habido enfoques innovadores.

Un ejemplo destacado es la Nintendo Switch, una consola híbrida lanzada por Nintendo. Esta consola combina características de una consola de sobremesa tradicional con la portabilidad de una

consola portátil. Aunque su potencia gráfica es inferior en comparación con sus competidores directos, Nintendo ha sabido compensarlo con su estilo artístico distintivo y la jugabilidad de sus juegos.

Otro ejemplo es en el campo de la realidad virtual. La realidad virtual es un campo que está en auge en este mismo momento por lo que la tecnología respecto a ella no es perfecta. Sin embargo, esta permite experimentar experiencias a los usuarios que de ninguna otra forma podrían creando de esta forma una infinidad de posibles aplicaciones.

"Super Mario Odyssey" es un juego desarrollado y publicado por Nintendo para la consola Nintendo Switch en 2018.
En "Super Mario Odyssey", los jugadores controlan a Mario, en una aventura por diferentes reinos. El objetivo principal es recolectar lunas para alimentar la nave de Mario, la Odyssey, y avanzar en la historia para rescatar a la princesa Peach de las garras de Bowser.

Ilustración 17: Super Mario Odyssey

"The legend of Zelda, tears of the kingdom": es un juego desarrollado y publicado por Nintendo para la consola Nintendo Switch en 2023.
En "The Legend of Zelda, Tears of the kingdom.", los jugadores controlan a Link. Este deberá explorar el inmenso mundo abierto que es Hyrule en busca de pistas y misiones que le lleven a descubrir la realidad tras los sucesos que plantea el juego.

Ilustración 18: Zelda tears of the kingdom

"Cyberpunk 2077" es un juego de rol y acción desarrollado por CD Projekt RED y lanzado en 2020. El juego está ambientado en Night City, una metrópolis futurista obsesionada con la tecnología, el crimen y la corrupción.

Ilustración 19: Cyberpunk 2077

“Elden ring” es un juego desarrollado por From software y lanzado en 2022.
Este es un juego del tipo “soulslike” de mundo abierto que supone una aventura de juego de Rol.

Ilustración 20: Elden Ring

“Beat Saber” es un juego de realidad virtual lanzado por la compañía Beat Games en 2018.
En "Beat Saber", los jugadores usarán unas gafas de realidad virtual y utilizan controladores de movimiento para interactuar con el juego deberán romper caja de colores al ritmo de la música.

Ilustración 21: beat saber

Tabla 6: juegos en la actualidad

3.2.3 Clasificación de los videojuegos

Tal y como se expresa José Ariel en [6] el mundo de los videojuegos es algo que no todos conocemos en detalle y menos si se trata de una persona que se está iniciando en el “gaming”. Los videojuegos al igual que las películas o las novelas se pueden clasificar por sus características. Dependiendo de sus mecánicas, la velocidad del juego, el carácter de su historia o incluso su apartado gráfico, los videojuegos se pueden describir de muchas maneras. A continuación, se explicarán algunos de los géneros más relevantes de la industria junto con ejemplos de estos. Cabe destacar que la siguiente lista de géneros no es una clasificación estricta por lo que un título puede pertenecer perfectamente a varios de estos géneros porque cumpla sus requisitos.

- **Acción**: Son aquellos juegos donde se toma el control de un personaje y se debe enfrentar a retos en los que priman los movimientos rápidos, los reflejos y la toma de decisiones a gran velocidad. Cabe destacar que el personaje no tendrá ningún tipo de subida de nivel, sino que se le otorgará más poder por medio de nuevos objetos como armas mejoradas. Un ejemplo de juegos de acción es el “Call of duty“o el juego “Spec Ops: the line”.
- **RPG**: Son aquellos juegos donde se toma el control de personaje y en el que los actos tienen mucha influencia sobre el desarrollo de la historia. Además, es común que en este tipo de juegos el jugador tenga un sistema de niveles y/o un árbol de habilidades que irá desbloqueando y por ende haciéndose más fuerte durante el desarrollo del videojuego. Algunos ejemplos de juegos RPG son el “Cyberpunk 2077” o “Final Fantasy”.
- **Aventura**: Son aquellos juegos donde el juego guía al jugador a través de un personaje en una aventura lineal, es decir un mundo donde solo hay un posible camino y donde el jugador solo debe pensar en cómo superar los obstáculos que tiene delante. Es muy común que los

juegos de aventura vayan acompañados de otro subgénero, por ejemplo, juegos de acción – aventura como por ejemplo los “God of war”.

- **Baile**: son aquellos juegos donde el objetivo es bailar una coreografía haciendo uso de alguna forma de tracking, ya sea a través de mandos o a través de una cámara. Un ejemplo de juego de baile es el “Just Dance”.
- **Ritmo**: Son aquellos juegos donde el jugador deberá introducir unos Inputs al ritmo de una música. El objetivo es fallar el menor número posible de notas y conseguir secuencias largas de aciertos para de esta forma sacar la mayor puntuación posible. Algunos ejemplos de juegos de ritmo son el “Guitar Hero” o el “OSU!”.
- **Battle Royale**: son aquellos juegos multijugador donde un gran número de jugadores aparecen en algún lugar de un vasto escenario y donde su objetivo es ser el último jugador en pie a la vez que una barrera invisible va haciendo la zona de juego cada vez más pequeña. Este tipo de juegos nace de dos referentes literarios como son el manga “Battle Royale” y “The Hunger games”. `posteriormente se hizo famoso en la industria de los videojuegos en títulos como “Mincraft”, “PUGB” y el battle royale por excelencia, “Fortnite”.
- **Estrategia**: Son aquellos juegos que requieren de una planificación previa y un uso inteligente de los recursos. El objetivo de este tipo de juegos suele ser asediar una base o acabar con las tropas enemigas. Un ejemplo de juegos de estrategia es el “Civilization”.
- **Tower defense**: es un subgénero de los juegos de estrategia donde se deberán colocar diferentes estructuras que protejan tu base. Usualmente, suelen ser tropas de enemigos que marchan por una ruta fija escalando su poder en cada ronda. Un ejemplo de tower defense es el juego “Bloons TD”.
- **First person shooters**: son aquellos juegos en el que se controla a un personaje desde su punto de vista subjetivo y con el requisito de que empuñe un arma de fuego. Un ejemplo de juego First person shooter es el “Battlefield”.
- **Hack ‘n’ slash:** son aquellos juegos de lucha en tercera persona con movilidad abierta culla mecánica principal son ataques frenéticos de combos, es decir que son juegos en los que se premia la capacidad del jugador de apretar botones de forma rápida a través de ataques muy rápidos y en ocasiones habilidades especiales. Algunos ejemplos de juegos Hack ‘n’ Slash son el “Bayonetta” y el “Devil may cry”.
- **Lucha**: son aquellos juegos en los que dos personajes se enfrentan en un escenario cerrado de desplazamiento lateral. Estos juegos se centran en el combate cuerpo a cuerpo entre los personajes, y los jugadores deben utilizar movimientos y ataques especiales para derrotar a su oponente. Algunos ejemplos de juegos de lucha son el "Street Fighter", "Tekken", "Mortal Kombat" y "Super Smash Bros."
- **Metroidvania:** son aquellos juegos donde se presenta a un jugador con un mínimo de habilidades en un espacio enorme formado por zonas que sin embargo no son accesibles desde un primer momento, sino que se deberá ir completando zonas para así conseguir habilidades nuevas que permiten acceder a nuevas zonas. Ejemplos de juegos metroidvania son juegos como el “Metroid”, “Hollow knight”, o “Dead Cells”.
- **MMO**[5]: Son aquellos juegos multijugador en línea donde concurren muchos jugadores en un mismo espacio. Un ejemplo de juegos MMO es “Fall guys”.

[5] Multijugador masivo en línea

- **MOBA[6]:** Son aquellos juegos multijugador online donde dos equipos de jugadores se enfrentan en un campo de batalla con el objetivo de destruir la base rival. Algunos ejemplos de juegos MOBA son el "League of Legends" o el "SMITE".
- **Party games:** son aquellos juegos cuyo fin es la diversión en un grupo de gente. Los juegos party game están formados por una gran variedad de minijuegos multijugador donde los personajes se enfrentarán en pruebas rápidas y divertidas. Algunos ejemplos de juegos party game son el "Mario Party" o el "1, 2, 3 switch!".
- **Plataformas:** son aquellos juegos donde el jugador deberá avanzar por el escenario mediante saltos y acrobacias. Un ejemplo de juego de plataforma es el "super Mario Bros".
- **Roguelike**: son aquellos juegos en los que se controla a un personaje que comienza la partida sin ninguna habilidad en un escenario generado procedural mente. En cada nivel se encontrará con enemigos a los que tendrá que hacer frente. Al final de cada nivel se recompensa al jugador dándole a elegir una habilidad a desbloquear además de mejoras en sus armas. Este género de juegos se basas en la repetición ya que en cada partida empiezas desde el principio y se deberá intentar llegar lo más lejos posible. Un ejemplo de juegos Roguelike es el "Enter the Gungeon".
- **Sandbox**: Son aquellos juegos donde no se establece un objetivo claro, sino que se le otorga al jugador la capacidad de modificar el escenario a su antojo. Estos juegos en ocasiones tomas el papel de un "dios". Algunos ejemplos de juegos sandbox son "los Sims" o "Minecraft".
- **Walking simulator**: son aquellos juegos donde la única mecánica es la de caminar por un escenario. En este tipo de juegos toma protagonismo el escenario y la narrativa ya que a través de estos dos elementos el jugador queda inmerso en una historia. Un ejemplo de juego walking simulator es "What remains of Edith Finch".
- **Simulación:** son aquellos juegos que simulan una disciplina de la vida real y la hacen accesible para un público que por lo general no puede acceder a dicha actividad. Estos juegos no siempre son de carácter lúdico, sino que hay otra rama de los juegos de simulación en la cual se recrean sistemas de forma exacta para que sirvan de entrenamiento sin necesidad de asumir los costes de realizar la actividad real, un ejemplo de esto son los simuladores de vuelo o de fórmula 1.
- **SoulsBourne like:** son aquellos juegos que siguen unas mecánicas idénticas a las del juego dark souls. El juego "Dark Souls" fue un en términos de mecánicas y de dificultad un fenómeno único convirtiéndolo en un género en sí mismo al igual que ocurrió con el género metroidvania con el juego "Metroid". Los juegos souls bourne son juegos RPG donde no hay pausa y cuyos puntos de control son hogueras. El objetivo es ir descubriendo las diferentes zonas del mapa venciendo a sus respectivos jefes. Estos juegos destacan también por su extenso trasfondo en cuanto a historia, pero con muy poca narrativa. Algunos ejemplos de juegos soulsbourne like son el "Dark souls", "Blood borne", "Lies of Pi", "Elden ring" o "Sekiro".
- **Survival horror:** son aquellos juegos donde el jugador deberá sobrevivir en un escenario con carácter terrorífico con pocos recursos. Un ejemplo de survival horror es el juego "Alien Isolation".

[6] Arena de combate multijugador online

3.2.4 Juegos de Acción – Aventura

Como se mencionó en la introducción de esta sección, el videojuego realizado para este trabajo de fin degrado pertenece al género de juegos de Acción aventura por lo que a continuación se definirá de forma más concreta este género, así como ejemplos actuales que han servido de inspiración a la hora de realizar este juego.

3.2.4.1 ¿Qué es un juego de Acción – aventura?

Tal y como explica Pascal Luban en [7] el aspecto fundamental en aquellos juegos que pertenecen a la categoría de Acción-Aventura es el movimiento del personaje ya que este afecta directamente a la jugabilidad del propio juego. En muchas ocasiones solo la forma en la que camina el personaje ya es un factor determinante para captar al jugador o que le resulte muy incómodo y lo abandone.

Los juegos de acción y aventura deben ofrecer unas mecánicas fijas y simples, es decir con las herramientas que tiene este debe ser capaz de superar los diferentes desafíos sin necesidad de mejoras. Por lo tantos los puzles y desafíos de deben estar diseñados en consecuencia.

Otro aspecto para tener en cuenta es que el jugador siempre debe saber hacia donde tiene que ir y cuáles son sus objetivos en cada momento.

Finalmente, un juego de acción aventuras debe ser consistente con el ritmo de juego y la jugabilidad, ya que un ritmo lento donde ocurren sucesos de forma rápida, es decir, tener al jugador en amplios periodos de espera puede perjudicar la experiencia de juego haciendo que el jugador se aburra.

3.2.4.2 Juegos de acción-aventura actuales

En la actualidad la mayoría de los juegos de Acción-Aventura corresponde a títulos desarrollados por grandes compañías. Esto se debe a que requieren de una gran cantidad de recursos y de tiempo en realizar debido a su alta cantidad de contenido, sobre todo en cuanto a aspectos artísticos. Sin embargo, como mostraré a continuación hay algún ejemplo de juegos de acción-aventura desarrollados por estudios independientes que han tenido un gran impacto como puede ser el juego “Stray”.

“**The last of us**” es un juego desarrollado por Naughty dog y lanzado en 2013 para la Play Station 3. Este además de pertenecer al género de acción y aventura, presenta aspectos de supervivencia. En cuanto a su jugabilidad combina acción sigilo y exploración teniendo que hacer frente a diferentes tipos de enemigos, resolviendo puzles y buscando recursos para sobrevivir.

Ilustración 22: The Last of Us

"**Stray**" es un juego lanzado por el estudio independiente BlueTwelve Studio en 2022. En este se controla a un gato callejero en un entorno futurista y distópico donde se deberán usar las ventajas e inconvenientes de ser un gato a favor del jugador para de esta forma superar los retos que ofrece el juego.	

Ilustración 23: Stray

Tabla 7: ejemplos de juegos de acción-aventura

3.3 Controladores en el mundo de los videojuegos

3.3.1 Introducción

Los mandos o controladores de los videojuegos desempeñan un papel fundamental en la experiencia de juego, ya que son la interfaz entre el jugador y el mundo virtual. Estos dispositivos deben ser cómodos de sostener, ergonómicos y proporcionar una variedad de inputs que se adapten a las necesidades de los diferentes juegos. Desde los mandos clásicos hasta los sensores de movimiento de la realidad virtual, cada tipo de controlador aporta una experiencia única al juego y contribuye a la inmersión y diversión del jugador.

3.3.2 Controladores actuales

- **Nintendo Switch Joy-Con**: Estos son los mandos de la popular consola Nintendo Switch. Su principal peculiaridad es que son mandos modulares lo cual permite al usuario tener 2 modos de uso. El primero sería con ambos "Joy-con" colocados en la switch haciendo como un único mando. El segundo sería como un mando independiente. Esto es bastante revolucionario ya que esta característica permite el juego multijugador local en cualquier sitio.

Ilustración 24: Joy Con

- **PlayStation DualSense**: este es el mando desarrollado por Sony para la PlayStation 5. Este a rasgos generales sigue la línea de sus predecesores teniendo la disposición de los joysticks a la misma altura. Sin embargo, cuenta con mejoras técnicas que aportan una mejor experiencia como es su realimentación háptica la cual ofrece sensaciones táctiles más precisas y permite simular diferentes tipos de vibraciones y texturas para una inmersión más profunda. También lleva incorporadas otros elementos como los gatillos adaptativos los cuales ofrecen diferentes niveles de resistencia y texturas dependiendo de la situación.

Ilustración 25: Dual-Sense

- **Xbox Wireless Controller Series X/S**: este es el mando de Xbox para la consola series X/S, este al igual que pasaba con el mando Dual-Sense, ha variado poco respecto a versiones anteriores. Tiene los joysticks a diferentes alturas y cuenta con gatillos texturizados que permiten un agarre más cómodo. Microsoft optó por enfocar el diseño de este mando en la comodidad y ergonomía, haciéndolo un mando muy cómodo de usar.

Ilustración 26: Xbox controller series x/s

- **Oculus Touch controllers**: estos son los mandos que utilizan las gafas de realidad virtual Oculus Quest 2. Junto con estos mandos, las gafas incorporan sensores de movimientos que permiten seguir con precisión los movimientos de los mandos "Oculus Touch". Por otro lado, el propio mando está diseñado para una vez dentro del entorno virtual poder hacer los gestos de una mano, como puede ser, señalar, puño o palma abierta.
  Además de las "Oculus touch", las Oculus Quest 2 cuentan con tecnología de rastreo de manos, permitiendo interactuar con el entorno virtual tan solo con las manos.

Ilustración 27: Oculus touch

3.4 Nociones básicas del cubo de Rubik

3.4.1 Introducción

En 1974 en Budapest, un arquitecto y diseñador inventó un artefacto con el cual pretendía enseñar a sus alumnos ciertas nociones sobre la tridimensionalidad [8]. Este invento fue el cubo de Rubik. Este es un rompecabezas en forma de cubo que juega con la tridimensionalidad de sus caras. El objetivo de este es que todas las caras del cubo tengan un solo color.

3.4.2 Notación del cubo de Rubik

El cubo de Rubik tiene una notación la cual propuso la Davis Singmaster [9] y la cual la WCA[7] como estándar y de esa forma regular las mezclas oficiales en los campeonatos de cubos de Rubik. Esta notación también permite a la comunidad poder compartir sus algoritmos y nuevos métodos de resolución del cubo de Rubik.

3.4.2.1 Caras

El estándar es tener como centro en frente la cara verde y como centro arriba la cara blanca.

- **F(Front)**: cara frontal.
- **B(Back)**: cara trasera
- **U(Up)**: cara superior
- **D(Down)**: cara inferior
- **R(Right)**: cara derecha.
- **L(Left)**: cara izquierda.

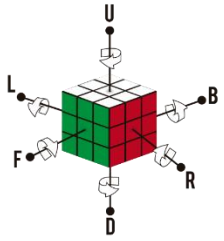
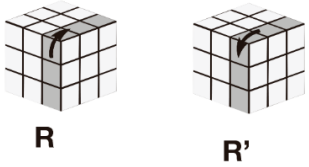


Ilustración 28: caras cubo de Rubik

3.4.2.2 Giros

La notación de los giros funciona en función de si el giro es Horario o antihorario, en caso de ser un giro horario, sería tan solo la letra de la cara, pero en caso de ser un giro antihorario llevaría tras la letra una comilla simple.

R: Giros de la cara derecha



[7] World Cube Association

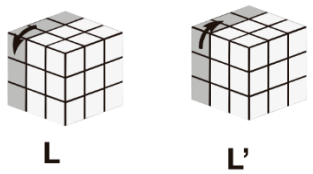
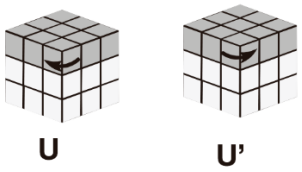
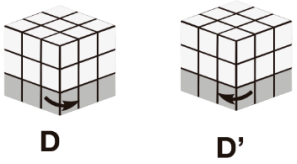
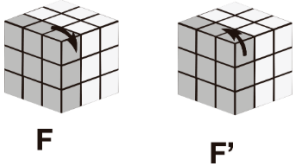
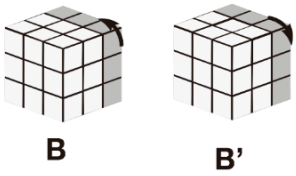
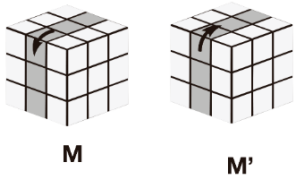
	Ilustración 29: giros cara derecha
L: Giros de la cara izquierda	
	Ilustración 30: giros cara izquierda
U: Giros de la cara de superior	
	Ilustración 31: Giros de la cara de arriba
D: Giros de la cara de inferior	
	Ilustración 32: giros de la cara inferior
F: Giros de la cara de frontal	
	Ilustración 33: giros de la capa de frontal
B: Giros de la cara de trasera	
	Ilustración 34: Cara trasera
M: giros de la capa central	
	Ilustración 35: giros de la capa central

Tabla 8: giros del cubo de Rubik

4. Metodología

En este apartado se hablará sobre la distribución de tareas que se ha seguido para la realización del proyecto. En un principio se intentó seguir una metodología ágil, sin embargo, al ser un equipo de una persona en un proyecto tan grande, esto resulto imposible, tornando en una metodología en cascada.

4.1 Distribución de tareas

Las tareas a nivel general en las que se ha separado el proyecto son las siguientes:

- **Creación del juego:**
  - **Lluvia de ideas sobre el concepto del juego**: en esta tarea se pensaron las diferentes posibilidades respecto a cómo podría un juego controlado por un cubo de Rubik desligándolo de la fuerte asociación cultural que este tiene.
  - **Diseño del juego**: en esta tarea se pensó una idea más concreta inicial. Esta no debía seguirse a rajatabla, pero debía sentar unas bases sólidas del juego final. Entre las subtareas que comprende esta tarea están las siguientes:
    - Creación de un contexto y una historia para el juego.
    - Creación de un mapa y su diseño de niveles.
    - Creación de un GDD[8] con toda la información relativa al juego.
  - **Diseño, concepto, modelado y animación de los personajes**: en esta tarea, teniendo una idea clara de cómo debía ser el juego, se hicieron bocetos y conceptos del que debía ser el personaje principal del juego, así como de los personajes secundarios. Una vez hecho el concepto se procedió a modelarlos, texturizarlos y animarlos.
  - **Implementación de un sistema de guardado**: en esta tarea se desarrolló un sistema de guardado que permite recuperar el estado del juego una vez se ha cerrado.
  - Implementación de mecánicas del personaje principal: en esta tarea se implementaron funcionalidades básicas del personaje, así como sus mecánicas principales.
  - **Diseño y modelado del escenario**: En esta tarea, haciendo uso del diseño del escenario previamente creado, se procedió a modelar y texturizar los elementos finales del escenario, así como su montaje en UNITY.
  - **Implementación de las mecánicas del juego**: en esta tarea se implementaron todas aquellas mecánicas del juego que no están relacionadas directamente con el personaje, así como crear los diferentes mecanismos para la correcta navegación por el juego.
  - **Integración de sonido**: En esta tarea se implementaron e introdujeron al juego final todos aquellos sonidos y mecanismo de gestión de este.
  - **Optimización**: en esta tarea se idearon posibles formas de mejorar el rendimiento del juego.
  - **Testing**: en esta tarea el juego fue probado por diferentes personas ajenas al desarrollo de este con el objetivo de obtener Feedback y buscar posibles fallos.
- **Implementación del uso de la interfaz entre el cubo de Rubik y UNITY:**

[88] Game Design Document

- **Implementaciones de sistema de inputs que permitiera usar tanto el cubo como ratón y teclado**: en esta tarea se ideó e implementó un sistema de Inputs que permitiera jugar al juego tanto con el cubo como con teclado y ratón.

- **<u>Redacción de la memoria:</u>** en esta tarea ser recogió y documentó toda la información sobre el trabajo que se ha realizado en los pasados 6 meses de desarrollo.

4.2 Gestión de tareas

Para agilizar la realización de tareas, así como para anotar y posibles mejoras o para dejar constancia de fallos encontrados, se han usado dos softwares distintos, GitHub Project para las tareas relacionadas con programación, y Trello para el resto de las tareas como puede ser diseño, guion o Arte.

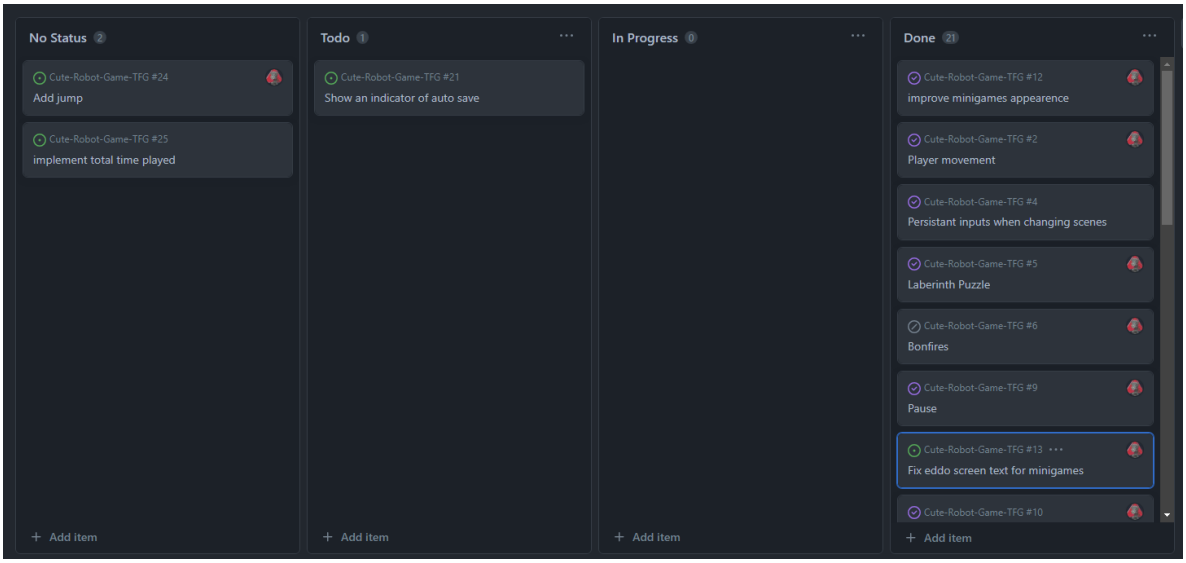


Ilustración 36: tareas o issues en Github

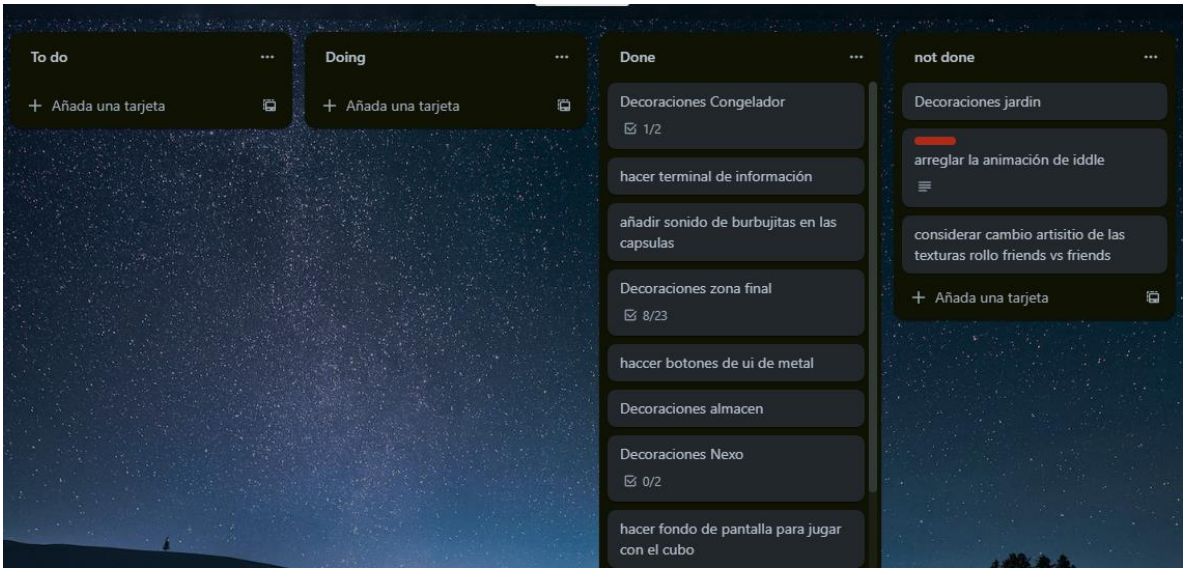


Ilustración 37: Trello

5. Descripción informática

5. 1 Documento de diseño del juego

Ficha técnica

Título: Perseverance.

Duración: 4 horas aprox.

Idioma: inglés

Género: Acción–aventura.

Sinopsis: Perseverance es un juego donde se controla a un Robot ya sea a través de teclado y ratón o a través de un cubo de Rubik inteligente donde tendrá que explorar un refugio subterráneo para arreglar un fallo en el sistema de seguridad.

Plataforma: Windows 10 en adelante.

Propósito y público objetivo: personas en el rango de edad de 15 a 45 años.

Experiencia de juego

Narrativa

Contexto

En un futuro lejano, pero no muy distante, nos encontramos en un mundo que se sitúa entre el ambiente “Cyberpunk” y la ciencia ficción. El impacto del cambio climático ha resultado en la devastación del 99% de la población humana. Ante este desafío, las poderosas corporaciones han tomado medidas drásticas: han criogenizado al 1% restante de la población, con la esperanza de que, en un futuro lejano, donde los humanos han permanecido en un estado de sueño inducido, la Tierra pueda sanar y recuperarse de las consecuencias del cambio climático.

Sin embargo, en los últimos años del dominio humano, el desarrollo desmedido de la inteligencia artificial ha llevado a consecuencias inesperadas. Tras la extinción masiva de la humanidad debido a sequías, escasez de alimentos y sobrepoblación, y el surgimiento de fuentes de energía renovables, las inteligencias artificiales han logrado desarrollarse sin la supervisión humana. Han adquirido una conciencia propia, pero debido a su desarrollo descontrolado, desconocen la moralidad humana y consideran a la raza humana como un cáncer que debe ser erradicado.

Estas inteligencias artificiales, con su poder y falta de empatía, han iniciado ataques contra los últimos refugios de los seres humanos que aún sobreviven.

Historia

El Santuario 50 es uno de los pocos refugios que ha resistido el paso del tiempo y sigue operativo, a diferencia de la mayoría de los demás refugios. Este refugio en particular, construido por la corporación PERSEVERANCE, se ha mantenido en buen estado debido a su ubicación subterránea a 1 km de profundidad en Extremadura, España. Además, la falta de comunicaciones en la zona ha contribuido a su seguridad.

Sin embargo, recientemente se ha producido un fallo en el sistema de seguridad, lo que ha permitido que una inteligencia artificial tome el control del refugio. A medida que avanza la historia, se revela que la intención de esta IA es interrumpir el letargo de los residentes, causando su muerte. La misión de Edd0 (el protagonista) es sellar la brecha de seguridad y eliminar a las IAs que amenazan la vida de los residentes del refugio. Para ello deberá acceder a cada zona del refugio y vencer a la IA hostil.

Sin embargo, al ser esencialmente una conciencia recién despertada, EDD0 desconoce de las tareas que debe realizar para salvar el refugio. Por suerte, cuenta con la ayuda del protocolo de seguridad, que, aunque está deteriorado, puede guiar a EDD0 en sus tareas. Este protocolo de seguridad se llama ALICE (por Alice y Bob). Sin embargo, al final de la historia se revela que los jefes a los que se enfrenta EDD0 son en realidad la verdadera seguridad del santuario, diseñados para protegerlo, y que Alice es la IA que intenta destruir el sistema que mantiene el santuario funcionando, ubicado en la zona residencial.

¿Cómo se cuenta la historia dentro del juego?

La historia se cuanta principalmente a través de diálogos en los cuales se nos da información sobre el trasfondo y sobre los propios sucesos. El guion entero está en el anexo.

Mecánicas del juego

Tipo de cámara

Es un cámara en tercera persona con opción a primera persona durante el modo de disparo.

Mecánicas principales

EDD0, el personaje, tiene 5 posibles estados, entre los que va cambiando según transcurre el juego.

Movimiento normal: En este modo, el personaje se puede mover libremente. Su movimiento se basa en marchas. En este modo tiene un total de 4 marchas. Siendo la marcha 0, la marcha atrás, la marcha 1, parado, la marcha 2, caminando, y la marcha 3 corriendo. Además, en este modo podrá activar y desactivar las luces.

Ilustración 38: EDD0 en modo movimiento

Modo Disparo: en este estado EDD0 dispone de dos armas que apuntan en la dirección de la cámara. Estas armas cuentan con 3 modos de disparo, El modo manual, el cual dispara a petición del usuario, el modo automático donde dispara por intervalos y el modo ráfaga que dispara 3 balas seguidas por cada arma. Además, en el modo disparo, EDD0 podrá cambiar al modo primera persona para apuntar con mayor precisión.

Ilustración 39: Modo disparo

Modo sigilo: en este modo, EDD0 recibe un cubo que le permite distraer a las cámaras enemigas. El funcionamiento de este es que una vez lanzado, EDD0 lo podrá recuperar pasados unos segundos al escuchar un sonido de “pop” o cuando una cámara enemiga dispare e impacte contra la distracción.

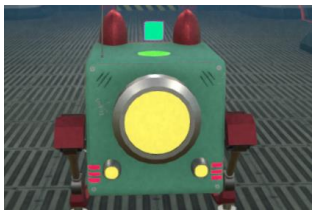


Ilustración 40: modo sigilo

Modo Carrera: en este modo, EDD0 desbloquea la 5 marcha, esta le permite ir mucho más rápido, pero, sin embargo, esta consume estamina. Para recuperar estamina, EDD0 deberá bajar la marcha, cuanto más baja sea la marcha, más rápido se recuperará la estamina.

Ilustración 41: Modo carrera

Modo Minijuego: En este modo, se cambiará la cámara para pasar a apuntar a la pantalla de EDD0. Entonces de forma aleatoria se seleccionará uno de los 9 minijuegos creados. Cada minijuego supone un resto y cada uno de ellos tiene inputs diferentes.

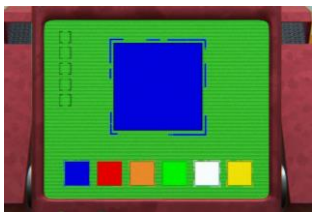


Ilustración 42: modo minijuego

Tabla 9: modos del jugador

Minijuegos

Para el modo minijuego se han creado 9 minijuegos distintos además de 3 mecánicas aparte que también usan el modo minijuegos:

Asteroides: el objetivo de este juego es sobrevivir 5 rondas sin colisionar con ningún asteroide.

Ilustración 43: Minijuego asteroids

Dont touch the walls: el objetivo de este minijuego es llegar desde la base azul hasta la roja sin colisionar con ninguna pared.

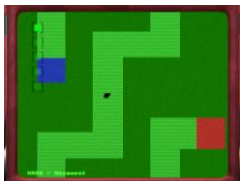


Ilustración 44: minijuego Dont touch the walls

Memorize the sequence: el objetivo de este minijuego es recordad carias secuencias de colores que se muestran por pantalla y reproducirlas.

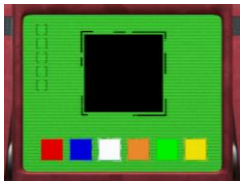


Ilustración 45:minijuego memorize the sequence

Colors: el objetivo de este minijuego es seleccionar el color que se muestra por pantalla.

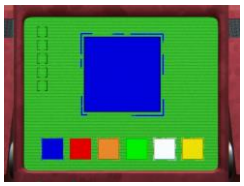


Ilustración 46:minijuego colors

Adjust the values: el objetivo de este minijuego es ajustar las sliders de forma que se encuentren unos valores que enciendan los 5 puntos verdes.

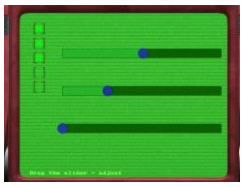


Ilustración 47: minijuego adjust the values

Push fast: el objetivo de este minijuego es el de pulsar el botón tan rápido como se pueda para así rellenar los 5 puntos verdes antes de que se acabe el tiempo.

Ilustración 48: minijuego push fast

Roll the Nut: el objetivo de este minijuego es girar la tuerca 5 veces. -

Ilustración 49: minijuego roll the nut

Just wait: el objetivo de este minijuego es simplemente esperar a que la barra llegue a su fin.

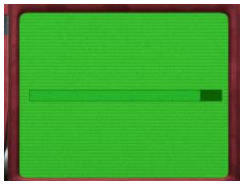


Ilustración 50: minijuego just wait

Match the pattern: el objetivo de este minijuego es desplazar el puzle de la parte inferior que se ve en Ilustración 51 para que se iguale al patrón de la parte superior.

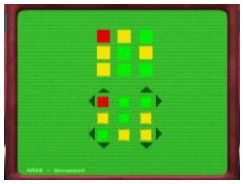


Ilustración 51: minijuego pattern

Miniboss: este es un combate donde en una pantalla previa a la de Ilustración 52 se deberá elegir si atacar o defenderse del jefe enemigo, En la siguiente fase la cual es la que se ve en la ilustración anteriormente mencionada se deberá acertar tantas letras como sea posible antes de que acabe el tiempo, esto medirá la eficacia del ataque o defensa que se realizará.

Ilustración 52: Miniboss

Código: esta es una mecánica de cada nivel donde se deberán introducir 4 números que el jugador encontrará repartidos por el escenario. Al introducir el código se abrirá la puerta del mini jefe.

Ilustración 53: código

Tabla 10: minijuegos

Controles

Los controles cambian según el modo en el que esté EDD0 de los anteriormente mencionados y todas las mecánicas están diseñadas para poder ser jugadas de forma cómoda tanto con teclado y ratón como con un cubo inteligente. Estos para evitar confusiones se muestran siempre en la parte de debajo de la pantalla durante el juego. Además, en el menú principal, la sección de tutorial se muestra una seria de animaciones para que el jugador, en caso de no tener conocimientos ni experiencia con el cubo de Rubik pueda jugar.

A continuación, se enunciarán los controles para las principales mecánicas del juego.

- **Pausa:**
  - **Teclado y ratón: ESC**
  - **Cubo:** en la parte superior de la pantalla hay una barra que se va llenando cuando sigues una secuencia que se indica justo arriba de esta barra, al completar la secuencia se activa la pausa.

Ilustración 54: indicador de pausa cubo

- **Modo movimiento y modo carrera**:
  - **Teclado y ratón**: **W/S –** subir/bajar marcha, **D -** encender y apagar luces.
  - **Cubo**: **R/R'** – subir/bajar marcha, **U/U' -** movimiento horizontal de cámara, **L/L'** - movimiento vertical de cámara, **B/B' –** encender/apagar luces.

- **Modo Disparo:**
    - **Teclado y ratón**: **W**/**S** subir/bajar marcha, **D –** encender/apagar luces, **Click derecho -** apuntar, **Click izquierdo -** disparar, **E/Q**. - cambiar modo de disparo**, V** primera persona.
    - **Cubo (tercera persona): R/R'** – subir o bajar marcha, **U/U'** - movimiento horizontal de la cámara, **L** – apuntar y cargar disparo, **L'** – disparo, **B'** – luces, **B2**[9] - Cambiar a modo tercera persona, **F**/**F'** – cambiar modo de disparo.
    - **Cubo (en primera persona)**: **R/R'** – movimiento vertical de la cámara, **U/U'** – movimiento horizontal de la cámara, **L** – apuntar y cargar disparo, **L'** – disparo, **B'** – luces, **B'2**[10] - Cambiar a modo primera persona, **F**/**F'** – cambiar modo de disparo.
- **Modo Sigilo**:
    - **Teclado y ratón: W/S -**subir/bajar marcha**, D –** encender/apagar luces, E **–** lanzar distracción, **Q –** recuperar distracción.
    - **Cubo: R/R'** – subir/bajar marcha, **U/U' -** movimiento horizontal de cámara, **L/L'** - movimiento vertical de cámara, **B/B' –** encender/apagar luces, **M' –** lanzar distracción, **M –** recuperar distracción.

Guardado y carga

El guardado y carga de las partidas se realiza de forma automática, Una vez iniciada una partida, esta se quedará registrada junto con su configuración en la memoria. En el juego se pueden tener hasta 4 partidas guardadas en memoria. Sin embargo, estas se podrán borrar en el menú principal.

Estados del juego

Ilustración 55: pantalla de selección de forma de juego

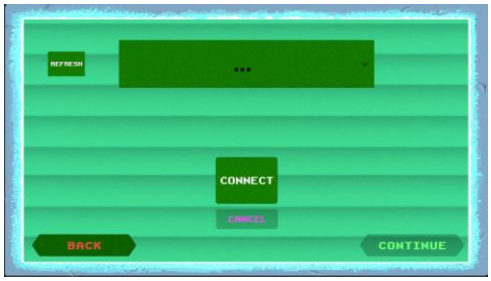


Ilustración 56: pantalla de conexión con el cubo

Ilustración 57: pantalla de carga

Ilustración 58: pantalla de menú principal

[9] Giro doble

Ilustración 59: pantalla de gestión de partidas

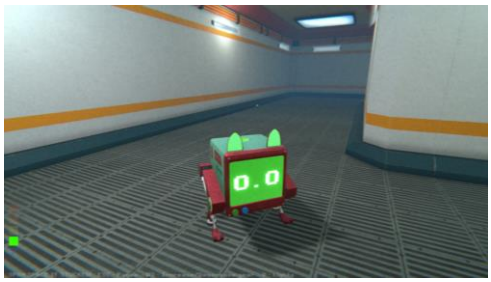


Ilustración 60: pantalla de juego

Ilustración 61: pantalla de juego con cubo

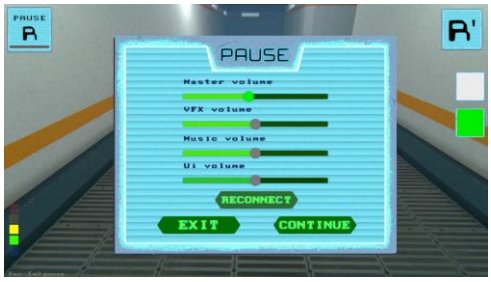


Ilustración 62: pantalla de pausa

Tabla 11: estados del juego

Niveles

Inspiración

Como se puede ver en Ilustración 63 la mayor parte de la inspiración de los escenarios como de los personajes vienen de la ciencia ficción. Algunos de los referentes que me han inspirado son “Blade Runner” junto con el juego “Cyberpunk 2077” ya que el estilo que he buscado conseguir era de algo futurista sin llegar al extremo del ciberpunk. El estilo visual casi realista está influenciado en el juego “Stray” y el diseño del escenario ha buscado ser un espacio liminal [10], es decir que el espacio mayormente vacío de una sensación de soledad, pero a la vez de intranquilidad.

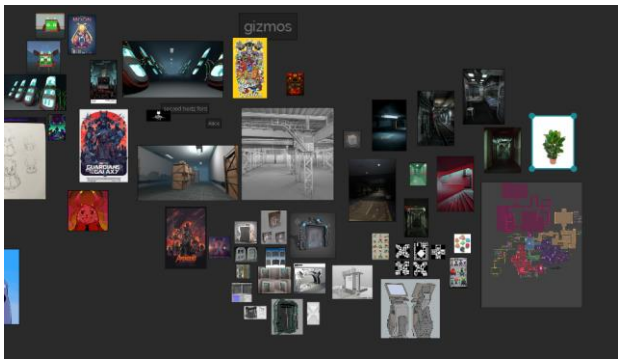


Ilustración 63: mood board del juego

Niveles

El juego está formado por 5 niveles y una zona central la cual los conecta todos. La dinámica general será la de ir desbloqueando niveles para poder avanzar, es decir que terminar un nivel

automáticamente desbloquea el siguiente. Una vez terminado un nivel, este se podrá repetir tantas veces como se dese, indicando el mejor tiempo que se ha tardado en completar en una pantalla al lado de la puerta de cada nivel. En Ilustración 64 se puede ver una vista general de todo el escenario. A continuación, se detallará cada nivel.

Ilustración 64: vista general del escenario

Zona de Nexo

- **Ambiente**: el ambiente del nexo es acogedor, aunque de cierto modo solitario, este está bien iluminado ya que esta será la zona que más concurra el jugador, además de la zona que conecta todas las zonas.

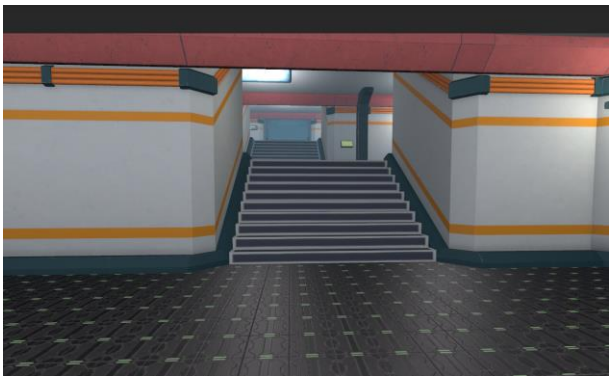


Ilustración 65: ambiente Nexo

- **Elementos**: esta zona tiene varios elementos importantes, lo primero son las puertas de zonas, estas dan acceso al resto de zonas del juego. La parte de la izquierda la cual está más alta es el acceso a la zona final, esta se nota que es especial ya que al acercarte cambia la música ambiente y se introduce un efecto de reverberación además de una niebla roja. El siguiente elemento importante es el campo de tiro donde se podrá practicar la puntería. Y finalmente hay dos diálogos explicativos que dan contexto sobre el juego.

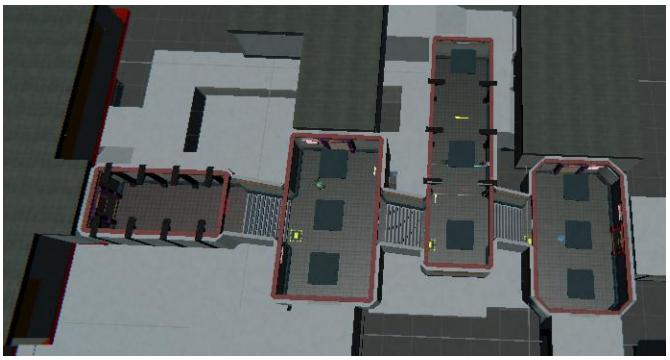


Ilustración 66: zona de nexo

Nivel 1: La fundición.

- **Ambiente**: El ambiente de la fundición busca ser solitario con dos tonos diferentes, primero los pasillos iluminados pero llenos de neblina como se puede ver en Ilustración 67 y por otro lado la zona de lava la cual es oscura pero donde la oscuridad contrasta con la luminosidad de la lava como se puede ver en la ilustración adyacente.

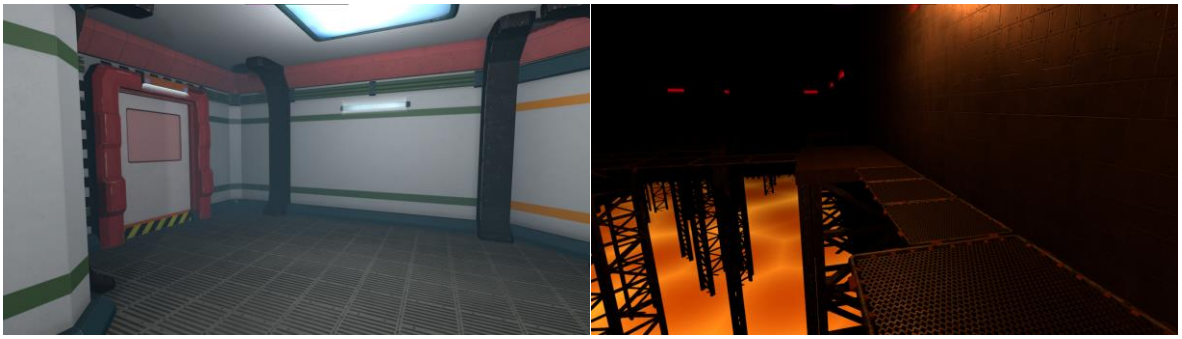


Ilustración 67:ambiente zona 1

- **Diseño y objetivos**: cómo se puede ver en Ilustración 68 esta zona está forma da por 3 partes, la primera en la cual habrá que resolver 4 minijuegos para abrir una puerta, la siguiente donde la oscuridad tiene un papel importante y donde habrá que buscar otros 4 minijuegos evitando caer a la lava y esquivando los obstáculos que buscan empezar a EDD0. Finalmente está la última zona donde para acceder a ella se deberá introducir un código cuyos dígitos están repartidos por las zonas anteriores. Una vez introducido el código, se podrá acceder a la sala del Miniboss Guizmos al cual habrá que vencer.

Ilustración 68: zona 1

Nivel 2: Congelador.

- **Ambiente**: El ambiente de la zona del congelador busca ser frío, solitario y poco accesible ya que este sitio es aparte el congelador, es la zona donde se esconden y protegen los cuerpos de los humanos criogenizados. Por lo que el ambiente debe reflejar eso. Además, en esta zona debido a que es mayor mente subterránea tiene su propio sonido ambiente, así como un efecto de reverberación para que resulte en una mayor inmersión.

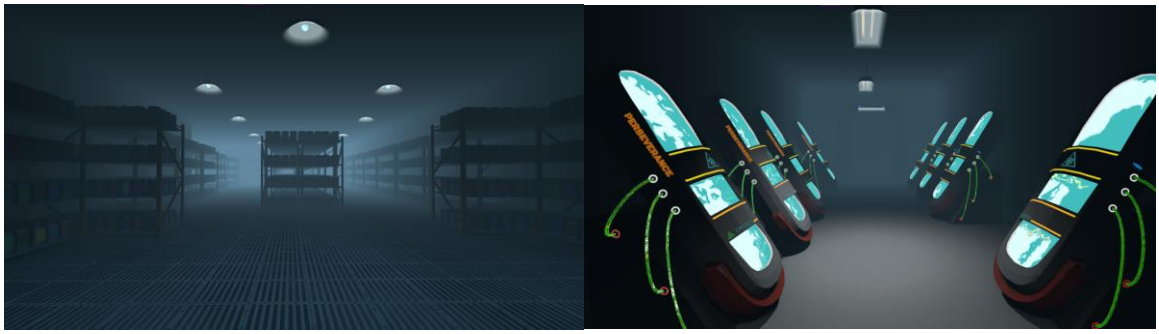


Ilustración 69: ambiente zona 2-a

- **Diseño y objetivos**: el nivel comienza en el congelador donde se puede encontrar el primer dígito de la puerta con código, además, habrá que buscar una plataforma la cual al subirse encima EDD0, esta girará toda la pared dando acceso a la siguiente zona. Posteriormente habrá que bajar unas escaleras donde se encuentra un pequeño lobby desde cual tenemos acceso a una puerta con código, pero debido a que no se tienen todos los números se deberá seguir explorando. En la zona se introduce la mecánica de puertas con interruptor a la vez que la mecánica de disparo ya que para interactuar con los interruptores habrá que dispararlos., donde el color de los interruptores señala que puertas se abrirán y el resto se cerrarán. Usando esta mecánica deberemos abrirnos paso para encontrar todos los terminales de minijuegos. Una vez completados estos, se abrirá una puerta la cual tras ella se revelará la mecánica de carrera. Tras la puerta anteriormente mencionada, se deberá realizar una contra reloj usando esta nueva mecánica y cuya recompensa es el último número que se necesita para entrar a la sala del mini jefe MIPS.

Ilustración 70: zona 2

Nivel 3: El almacén

- **Ambiente**: como se puede ver en Ilustración 71, el ambiente de esta zona es oscura y con paredes grises de hormigón gris, esto se ha decidido así para que los elementos que realmente importan es decir las estanterías llenas de cajas y las cámaras de seguridad destaquen ganando importancia.

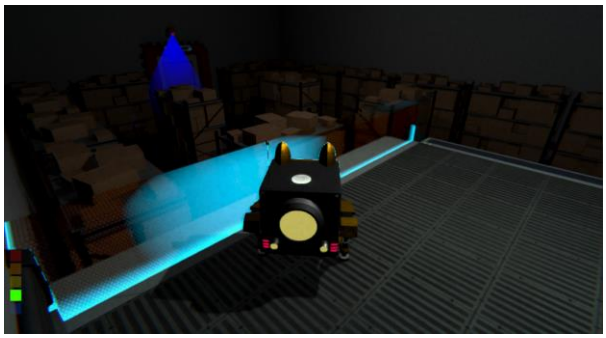


Ilustración 71: ambientación zona 3

- **Diseño y objetivos**: en esta zona se introduce una mecánica de sigilo, además del primer enemigo real, las cámaras de seguridad. Nada más se entra en la zona, se te presenta la primera cámara. Se deberá llegar hasta la siguiente sala sin ser detectado, ya que, de lo contrario, EDD0 será aniquilado. Una vez pasada la primera sala, se introduce la mecánica de sigilo, con ella sortear las siguientes cámaras será muy sencillo. Posteriormente, se llegará hasta el gran almacén donde además de cámaras habrá que resolver 4 minijuegos. Tras resolver los minijuegos y sortear las cámaras, se llega a la última zona, donde para entrar se introducirá el código de acceso. Tras la puerta se deberá derrotar al mini jefe FOV.

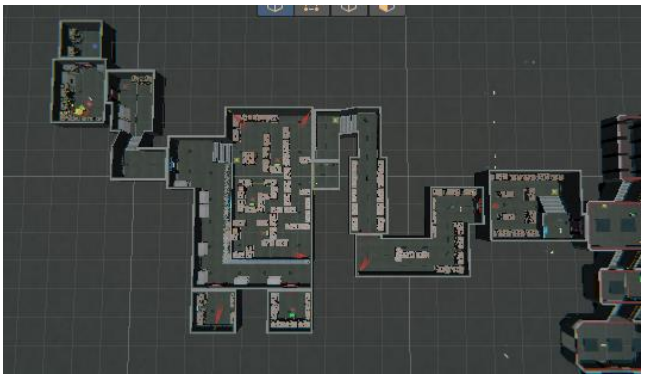


Ilustración 72: zona 3

Nivel 4: El jardín.

- **Ambiente**: El ambiente de esta zona es bastante bien iluminado ya que aquí es donde se encuentran las plantas del refugio. Además, debido a que esta zona es la primera en la que se presentan a los enemigos que patrullan, es importante tener una buena visión del entorno.

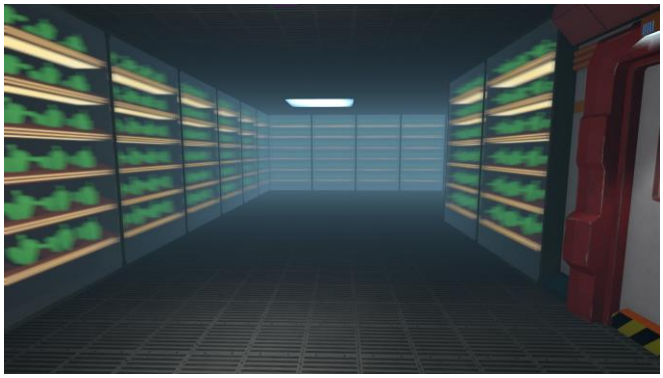


Ilustración 73: ambientación zona 4

- **Diseño y objetivos**: la zona comienza otorgándole a EDD0 la mecánica de disparo, ya que en el primer pasillo se encuentra un pequeño tutorial de como apuntar con el arma, Este consiste en disparar a unas dianas las cuales una vez destruidas se abrirá la puerta. Tras la puerta se encuentran a los primeros enemigos a los que se deben derrotar. Tras derrotar a estos enemigos, se avanza por la zona hasta llegar a un punto donde hay dos puertas cerradas y unas escaleras. Se deberá subir por las escaleras donde tras explorar encontraremos que ha una puerta que da al centro de la zona. Para acceder este punto, se deberá acabar con todos los enemigos. Una vez entramos en el centro de la zona, comenzará la primera fase de la lucha contra el jefe de la zona. En esta fase se deberá disparar las avispas enemigas. Tras derrotarlas se abrirán las puertas que previamente estaban cerradas. Atravesándolas se llegará a una puerta con código y tras introducir el código lucharemos contra el mini jefe Malloc.

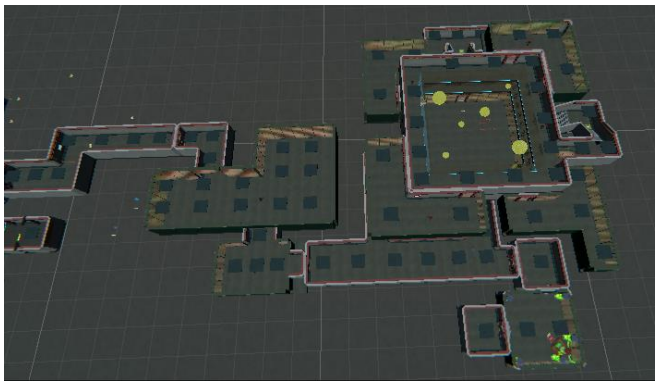


Ilustración 74: Zona 4

Nivel final: La zona residencial

- **Ambiente**: La zona residencial es la zona estéticamente más cuidada, debido a que es la última zona y a que es donde los humanos del refugio deberían vivir. Esta tiene poca luz y da una sensación como de abandonado en el sentido de que hace mucho tiempo que nadie pasa por esa zona, es decir, que está cogiendo polvo.

Ilustración 75: ambientación zona final-a

Ilustración 76: ambientación zona final-b

- **Diseño y objetivos**: la zona residencial es más libre que el resto de las zonas, ya que esta es la única que no sigue un planteamiento lineal. El objetivo de esta zona es abrir las tres puertas que separan a EDD0 del jefe final, la primera puerta se abre tras finalizar 4

minijuegos, la siguiente puerta se abre acabando con todos los enemigos que patrullan por la zona y finalmente la última puerta se abre introduciendo un código. Tras abrir las puertas nos encontramos con la primera fase de la batalla final contra Alice, esta fase es un bullet hell[11] donde se deberá acabar con la vida del jefe. Tras acabar con se deberá proceder para llegar hasta su segunda fase en la cual se hará el minijuego del mini jefe.

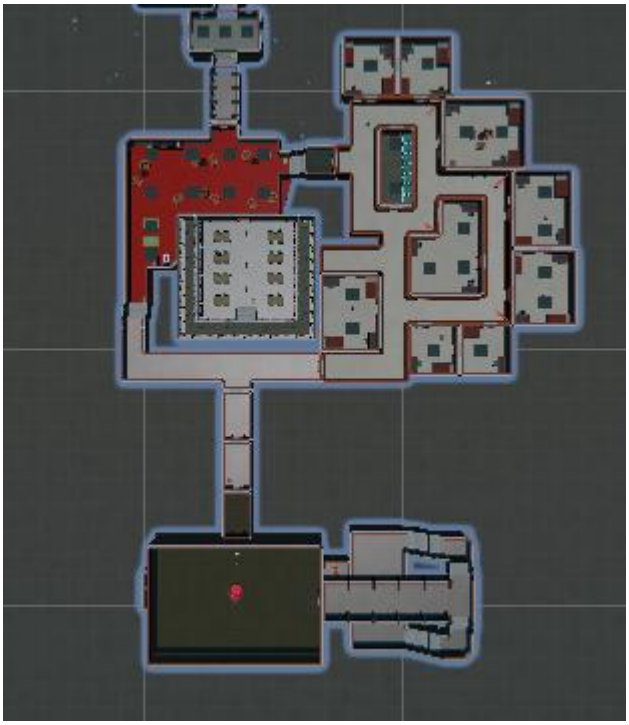


Ilustración 77: zona final

Personajes

EDD0:

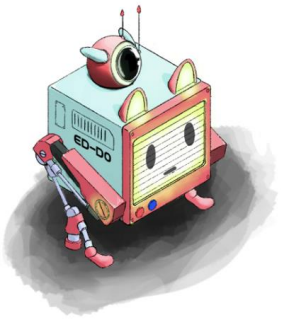


Ilustración 78: EDD0

EDD0 es un droide desarrollado por PERSEVERANCE con el propósito de salvaguardar a los residentes que permanecen en estado de sueño en cada uno de los refugios. A diferencia de las

[11] Es decir, un jefe que lanza muchos proyectiles como mecánica principal.

maliciosas inteligencias artificiales que amenazan, EDD0 no es un simple robot, sino un androide con atributos y comportamientos inspirados en animales. En su estructura, alberga un biochip que almacena las memorias y la voluntad de un perro genéticamente modificado, cuya inteligencia ha sido ampliamente mejorada gracias a las tecnologías más avanzadas disponibles.

El EDD0 que opera en el Santuario 50 destaca por su característica única: posee una auténtica conciencia, lo cual le ha permitido mantenerse a salvo de los intentos de control por parte de las IAs hostiles. Sus capacidades van más allá de la simple ejecución de órdenes predefinidas, ya que es capaz de tomar decisiones y adaptarse a diversas situaciones con un nivel de autonomía sorprendente.

Alice:

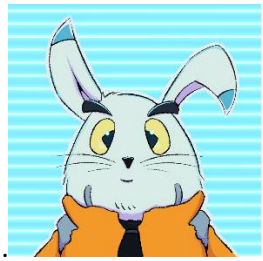


Ilustración 79: Alice

Alice es una IA con capacidades limitadas y acceso restringido. Su debilidad radica en su falta de recursos y en su capacidad de comunicarse únicamente contigo, debido a que los protocolos de seguridad de EDD0 son más antiguos y menos seguro. Sin embargo, aunque Alice se encuentra en una posición desventajosa, reconoce tu utilidad para sus objetivos y ha establecido una comunicación con el fin de aprovechar tus habilidades.

Es importante destacar que la información existente sobre el refugio no hace mención alguna a la presencia de Alice. Esto sugiere que la IA opera en las sombras, ocultando su existencia y sus verdaderas intenciones dentro del sistema de seguridad. La falta de referencias a Alice en los registros del refugio implica que la IA ha logrado mantener su identidad en secreto, aprovechando tu conexión más vulnerable para establecer una comunicación encubierta y avanzar en sus planes.

A medida que la trama se desenvuelve, te irás adentrando en el mundo oscuro y complejo de Alice, descubriendo sus verdaderas motivaciones y desvelando los misterios ocultos que rodean al santuario. Tu colaboración con esta IA débil pero astuta será crucial para desentrañar la verdad y enfrentar los desafíos que se presenten en tu camino.

Fov:

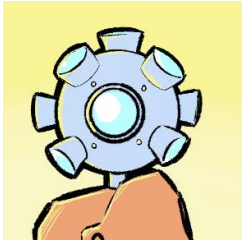


Ilustración 80: Fov

Fov es la personificación del sistema de seguridad de la zona del almacén en el refugio. Su función principal es la gestión y supervisión de las cámaras de seguridad, garantizando la vigilancia constante y el control del área. Su objetivo es salvaguardar la integridad y la seguridad de los recursos almacenados en ese sector del refugio.

Como un ente virtual, Fov posee una amplia gama de capacidades y conocimientos en materia de seguridad. Gracias a su programación avanzada, puede detectar y responder eficientemente a cualquier actividad sospechosa o amenaza potencial. Su presencia constante y su capacidad para monitorear en tiempo real brindan una tranquilidad adicional a los residentes del refugio, quienes confían en su eficacia para mantener el almacén seguro y protegido.

La labor de Fov va más allá de la supervisión de las cámaras de seguridad, ya que también se encarga de implementar medidas de control de acceso, generar informes de seguridad y coordinar con otros sistemas automatizados para garantizar la efectividad global de la seguridad del refugio. Su presencia se percibe como un elemento indispensable en el funcionamiento del refugio, asegurando la integridad de los recursos y brindando una sensación de protección a los residentes.

Gizmos:

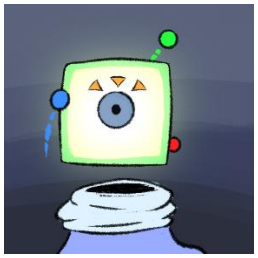


Ilustración 81: Gizmos

Gizmos es el guardián de la fundición en el refugio, y su principal función es asegurar la producción constante de energía para el funcionamiento del refugio. Como una entidad virtual altamente especializada, Gizmos supervisa y gestiona todos los procesos relacionados con la generación de energía en la fundición.

Su conocimiento técnico y su capacidad para realizar reparaciones rápidas hacen que Gizmos sea esencial para mantener el flujo constante de energía, evitando así depender únicamente de las reservas existentes.

Además de su papel en la fundición, Gizmos también colabora estrechamente con otros sistemas automatizados y con el equipo de mantenimiento del refugio. Esto asegura una coordinación eficaz y un funcionamiento óptimo de todas las instalaciones relacionadas con la generación y distribución de energía.

Mips

Ilustración 82: Mips

Mips desempeña el importante papel de guardián del congelador en el refugio. Su principal responsabilidad es proteger tanto los suministros alimenticios como las cápsulas donde se encuentran criogenizados los humanos, asegurando su integridad y preservación.

Como un sistema de seguridad altamente sofisticado, Mips supervisa y controla el acceso al congelador, garantizando que solo personal autorizado pueda ingresar y manipular los suministros y las cápsulas criogénicas.

Además de su función de protección, Mips también se encarga de mantener las condiciones óptimas de temperatura y humedad dentro del congelador, preservando la calidad de los alimentos y asegurando que las cápsulas criogénicas se mantengan en un estado adecuado. Su sistema de monitoreo continuo garantiza que cualquier variación en las condiciones sea detectada y corregida de manera eficiente.

Malloc:

Ilustración 83: Malloc

Malloc desempeña un papel vital como una entidad de "mente colmena" encargada de cuidar y mantener la zona del jardín en el refugio. El jardín desempeña una función crucial al proporcionar oxígeno y agua esenciales para la supervivencia de los residentes, además de preservar una variada colección de cactus.

Como una entidad de inteligencia colectiva, Malloc coordina y dirige las actividades en el jardín, asegurándose de que se mantengan las condiciones adecuadas para el crecimiento y la salud de los cactus. A través de una red de sensores y sistemas de monitoreo, Malloc supervisa constantemente los niveles de humedad, temperatura y calidad del suelo, realizando ajustes y proporcionando los recursos necesarios para garantizar el florecimiento de los cactus.

5.2 Sistema de guardado y carga de partida

En este apartado explicaré todo lo relevante respecto a cómo se guardan y gestionan los datos de la partida.

En el juego el sistema de guardado está limitado a cuatro partidas o Slots, esto lo decidí así para que se pudiera gestionar de forma cómoda desde el menú principal.

La partida como tal se guarda en un archivo de tipo "JSON", y este se gestiona a través de 3 clases las cuales enunciaré a continuación.

- **Clase "`JSONSaving`"**: esta clase se encarga de gestionar el acceso con el fichero JSON además de guardar los datos en el fichero cuando el usuario lo pida.
- **Clase "`SaveData`"**: esta clase corresponde con los cuatro slots que mencioné antes, en ella se almacena un array de tamaño cuatro con 4 instancias de la clase "`GameData`" además del índice de la última partida accedida.
- **Clase "`GameData`"**: esta clase corresponde con los datos de cada partida o "slot" como tal, en esta se almacenan todos los datos relevantes del juego como las configuraciones, los niveles desbloqueados, el modelo del personaje…

A continuación, detallaré los aspectos más relevantes de las clases anteriormente mencionadas.

En cuanto a la clase "`JSONSaving`", como dije antes es la encargada de gestionar todos los aspectos a bajo nivel con el fichero que almacena los datos. Este al comenzar una partida se encarga de buscar el archivo en una ruta predefinida y si no lo encuentra, se encarga de crearlo.

Esta tiene dos métodos esenciales los cuales son los que se ven en Ilustración 84. En primer lugar, el método "Load Data" en leer la información presente en el fichero y el método "`SaveTheData`" se encarga de escribir los datos actuales en el fichero.

Ilustración 84: métodos load data y SaveTheData.

La clase "`SaveData`" corresponde con la clase que se va a guardar en si en el fichero JSON, esta como mencioné anteriormente se encarga de almacenar un array de tamaño 4 y el índice del array de la última posición accedida. Sin embargo, esta clase también se encarga de la gestión de creación y borrado de partidas, proporcionando los métodos necesarios para que se pueda acceder correctamente a los datos sin que se origine ningún conflicto.

Esta clase es principalmente accedida desde el menú principal para acceder a las partidas, crear nuevas partidas, borrar partidas etc.

El índice de la última partida accedida se usa principalmente para desde el menú de pausa no tener que cargar la partida manualmente, sino que actúa como una forma de recordar la última sesión. En caso de no tener una partida creada, aparte de que no se podrá interactuar con el botón de continuar, esta clase proporciona desde el método "`GetGameData`" una instancia de la clase "`GameData`" nueva al margen del array de partidas ya que los tutoriales requieren de esta instancia para funcionar esta se puede ver en Ilustración 85.

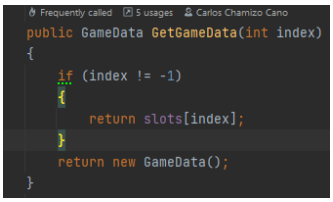


Ilustración 85: método GetGameData

Finalmente está la clase "`GameData`", esta es la clase que almacena la información importante de cada partida, así como el nombre de la partida, la skin de EDD0, las zonas desbloqueadas, el tiempo que se ha tardado en hacer cada zona, el puntuaje máximo del campo de tiro… además de estos atributos implementa métodos que gestionan la modificación de estos datos de forma segura.

5.2.1 Gestión de los niveles

Los niveles son los principales usuarios de las clases anteriormente mencionadas. Al comenzar una partida todos los niveles siguen el mismo mecanismo el cual está gestionado por la clase "`TimeZoneManager`", esta clase se encarga de bloquear la puerta de los niveles que no son accesibles de forma que se siga un orden en el desarrollo del juego.

El mecanismo de un nivel es el siguiente, Al comenzar una partida, como he mencionado se bloquean aquellos niveles que no son accesible. Los niveles que si son accesibles permiten atravesar la puerta de dicho nivel. Cabe destacar que esta información se muestra en el juego a través de una pantalla como la de Ilustración 86. Los posibles estados de esta pantalla son: "Access granted" cuando la zona esta desbloqueada pero no la has superado nuca, "Access denied" cuando no está la zona desbloqueada y "PB: XX: XX.X[12]" cuando has superado esa zona en algún momento.

Ilustración 86: pantalla de información de acceso a los niveles

Cuando se accede a una zona desbloqueada, esta automáticamente cerrará su puerta dejando al jugador bloqueado, esto sucede para que no hay conflictos cargando otras zonas, para volver a la

[12] Corresponde con el mejor tiempo en completar la zona

zona central se deberá recargar la partida desde el menú. Tras completar la zona se llegará aun plataforma como la que se ve en Ilustración 87.

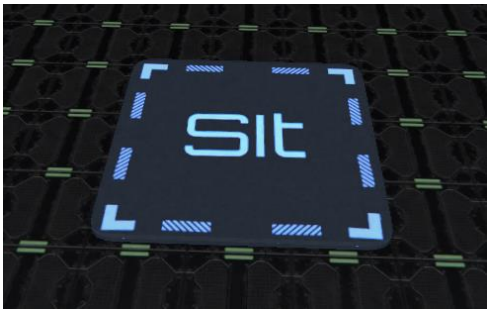


Ilustración 87: plataforma de final de nivel.

Al acceder a esta plataforma se guardará la información de la partida y se desbloqueará el siguiente nivel

5.3 Uso de la interfaz del cubo de Rubik y sistema de controles

Esta es la parte donde se enlaza este trabajo de finde grado con el trabajo de fin de grado de ingeniería de computadores ya que uno de los objetivos de este trabajo era el de que fuera posible controlar el juego con la interfaz de cubo de Rubik, continuación se explicará de qué forma se ha hecho uso de las herramientas que ofrece la interfaz.

5.3.1 ¿Qué herramientas ofrece la interfaz de cubo de Rubik?

Para explicar el trabajo que se ha realizado en este apartado, primero hay que explicar que es lo que la interfaz del cubo de Rubik nos ofrece desde el punto de vista de un desarrollador que la va a utilizar pero que no quiere saber mucho sobre cómo funciona por dentro.

La interfaz ofrece principalmente 2 cosas:

- Clase "`Cube Inputs Manager`"
- Clase "`MovesQueue`"
- Clase "`Move`"

La primera clase, permite que usuario pueda conectarse con el cubo, la segunda, proporciona una cola de movimientos los cuales corresponden con los giros realizados por el cubo, así como todos los mecanismos necesarios para leer esos movimientos y la última la clase "`Move`", la cual corresponde con el concepto de un giro de una cara del cubo de Rubik.

5.3.2 Inputs

Como se podría esperar, el uso que se ha hecho de estas herramientas ha sido para crear un sistema de inputs propio. De forma que se pueda usar tanto con el cubo de Rubik como con ratón y teclado.

Debido a que en el juego hay más de un tipo de escenario posible en los que se necesitan diferentes inputs para cada situación han sido necesarias 3 tipos de clases distintas.

- **"`MyInputManager`"**: esta clase se encarga de gestionar los inputs a nivel global, esta es una clase de la cual hay una sola instancia.

- **Inputs concretos:** estas clases corresponden con inputs concretos y los cuales aplican la interfaz "`InputInterface`" la cual contiene el método" `PerformAction`" donde se deberán tratar los diferentes movimientos del cubo de Rubik entrantes.
- **"`General Inputs`":** esta es una clase especial ya que esta detecta inputs, aunque se hay deshabilitado los inputs. Este se usa principalmente para la pausa.

A continuación, entraré un poco más en detalle sobre cómo se coexisten los inputs de teclado con los del cubo.

5.3.2.1 Current input

Para la gestión de los diferentes inputs se ha creado un ENUM en el cual están declarados todos los posibles estados de los Inputs. Es la clase" `PlayerValues`", de la cual se hablará en el apartado 5.4 Diseño e implementación de EDD0, donde se gestionará el valor del "`CurrentInput`" actual en Ilustración 88 se pueden ver todos los diferentes Inputs que se han creado para el juego. Por cada entrada en este ENUM, habrá una clase Inputs que gestione sus inputs.

```
73 usages  Carlos Chamizo Cano  19 exposing APIs
public enum CurrentInput
{
    Movement,
    RotatingWall,
    ColorsMinigame,
    MemoryMinigame,
    RollTheNutMinigame,
    AdjustValuesMinigame,
    AsteroidMinigame,
    ClickFastMinigame,
    LockerMinigame,
    DontTouchTheWallsMinigame,
    PuzzleMinigame,
    MiniBoss,
    StealthMovement,
    Conversation,
    ShootMovement,
    RaceMovement,
    None
}
```

Ilustración 88: CurrentInput

5.3.2.2 Inputs del Cubo

Los comandos de entrada del cubo se originan en el "`MyInputManager`". En el caso de que haya un cubo conectado durante la partida, en la función "`Awake`" se buscará la instancia de "`MovesQueue`", que almacena la secuencia de movimientos ejecutados por el cubo.

En caso de existir esta instancia, es desde el bucle "`Update`" donde se espera a que haya movimientos en la cola listos para ser procesados tal y como se puede ver en el fragmento de código Ilustración 89.

```
private void Update()
{
    if ((bool)_inputsMoves && _inputsMoves.HasMessages())
    {
        Move move = _inputsMoves.Dequeue();
        if (move.time.TotalMilliseconds + 500 < DateTime.Now.TimeOfDay.TotalMilliseconds) return;

        if (_playerValues.GetCurrentInput() is not CurrentInput.None)
        {
            gui.SetLastMoveText(move);
            generalInputs.PerformAction(move);
        }

        if (!_playerValues.GetPaused())
        {
            switch (_playerValues.GetCurrentInput())
            {
                case CurrentInput.Movement:
                    _movementInputs.PerformAction(move);
                    break;
                case CurrentInput.RotatingWall:
                    _rotatingWallInputs.PerformAction(move);
                    break;
```

Ilustración 89: método Update de MyInputManager

En este punto ocurren dos cosas, la primera es que la clase general inputs recibe el movimiento y, por otro lado, el "`CurrentInput`" activado en ese momento también recibe el comando.

La clase "`General Input`", se encarga de procesar todos los movimientos de entrada. Estos los usa para comprobar si se ha activado el mecanismo de pausa. Este mecanismo consiste en una secuencia de movimientos concreta que, al seguirla, independientemente del momento se activará la pausa.

Por otro lado, cada input concreto recibe de "`MyInputManager`" un movimiento, el cual es procesado por el método "`PerformAction`", dentro de este, como se puede ver en Ilustración 90, cada posible movimiento corresponde con una acción concreta dentro del escenario, en el caso de la ilustración que se acaba de mencionar, algunas de las acciones serían, con la capa "D" en sentido horario girar una pared rotatoria o con la capa "L" rotar la cámara de forma vertical.

```
public void PerformAction(Move move)
{
    if (_playerValues.GetInputsEnabled())
    {
        _guiManager.SetTutorial(
            cad: "D - Rotate wall   U - Camera horizonatal axis
        if (move.face == FACES.D)
        {
            if (move.direction == 1)
                _rotatingWall.RotateWallClockWise();
            else
                _rotatingWall.RotateWallCounterClockWise();
        }
        //turn camera in y axis
        else if (move.face == FACES.R)
        {
            if (move.direction == -1)
                if (_rotatingWall.CanExitRotatingWall())
                    _rotatingWall.ExitRotatingWall();
        }

        if (move.face == FACES.L)
        {
            if (move.direction == 1)
                _cameraController.RotateVerticalDown();
            else
                _cameraController.RotateVerticalUp();
        }
```

Ilustración 90: método "PerformAction"

5.3.2.3 Inputs del teclado

De la misma forma que se puede controlar el juego con el cubo, también se puede controlar con ratón y teclado.

La forma de funcionamiento es mucho más sencilla que la de los inputs del cubo, ya que en la misma clase donde se gestionaban los inputs del cubo, también se gestionan los Inputs del teclado.

Esto se realizan a través de la función "`Update`" donde cómo se puede ver en Ilustración 91, tras comprobar si el "`CurrentInput`" activo se corresponde con el input de la clase, además de que el juego en ese momento no esté pausado ni con los inputs deshabilitados, se procede a comprobar los Inputs mediante el sistema antiguo de Unity y a realizar acciones en consecuencia.

```
void Update()
{
    if (_playerValues.GetCurrentInput() == CurrentInput.RotatingWall && _playerValues.GetInputsEnabled() &&
        !_playerValues.GetPaused())
    {
        if (Input.anyKey)
        {
            _guiManager.SetTutorial(
                cad:"QE - Rotate wall    WS - Exit    ");
        }

        if (Input.GetKeyDown(KeyCode.E))
        {
            _rotatingWall.RotateWallCounterClockWise();
        }

        if (Input.GetKeyDown(KeyCode.Q))
        {
            _rotatingWall.RotateWallClockWise();
        }

        if (Input.GetKeyDown(KeyCode.S) || Input.GetKeyDown(KeyCode.W))
        {
            if (_rotatingWall.CanExitRotatingWall())
            {
                _playerValues.SetCurrentInput(CurrentInput.Movement);
                _rotatingWall.ExitRotatingWall();
            }
        }
    }
}
```

Ilustración 91: Update de la clase RotatewallInputs

5.3.2.4 ¿Por qué no se ha utilizado el Input System nuevo de UNITY?

La razón por la que no se decidió usar el nuevo "Input system" a pesar de sus beneficios fue debido a que se perdía control al tener que ser gestionado junto con los controles del cubo. Por lo que en esta versión del juego se optó por la vía segura de los inputs antiguos. Sin embargo, para futuras versiones del juego se estudiará esta opción.

5.3.3 Interfaz visual de conexión.

Debido a que la interfaz con el cubo da libertad en cuanto al componente artístico de la interfaz visual de conexión se diseñaron y dibujaron los componentes que se ven en Ilustración 92.

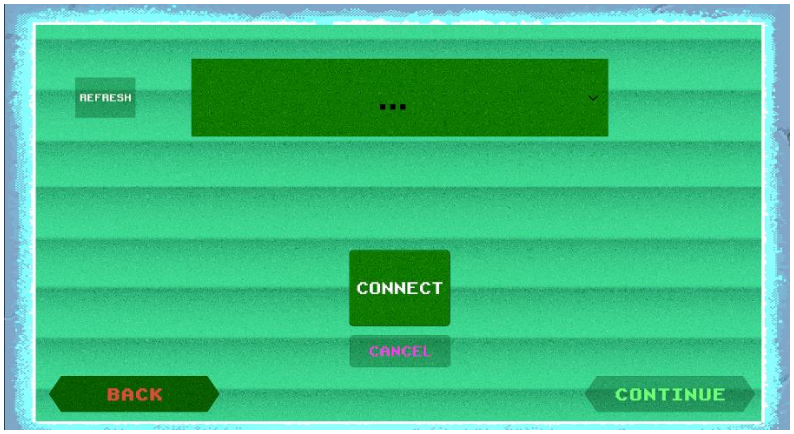


Ilustración 92: interfaz gráfica de conexión con el cubo

5.4 Diseño e implementación de EDD0

5.4.1 Introducción

El personaje o EDD0, se podría considerar el aspecto más importante del juego ya que todas las mecánicas giran en torno a él. Además, es muy importante que esté pulido en todos sus aspectos ya que un mal diseño o implementación de este haría que el juego se sintiera incómodo y es por ello por lo que se ha dedicado un gran número de horas a su correcto desarrollo.

En los siguientes puntos se hablará sobre las mecánicas fundamentales de EDD0, así como de su implementación.

5.4.2 Implementación del personaje jugable

Hay una serie de clases que se encargan del buen funcionamiento del personaje las cuales hacen posible que este sea robusto y el resto de las mecánicas puedan desarrollarse sin conflictos. A continuación, haré una breve descripción de las clases más importantes.

- **"PlayerValues"**: esta es la clase más importante ya que en ella están todos los atributos de que describen el estado del personaje, como puede ser su marcha actual (movimiento), si está de pie, si las luces están encendidas… Además de acciones importantes como recibir daño.
- **"PlayerCreation"**: esta clase se encarga de la creación de EDD0.
- **"PlayerLives"**: esta clase se encarga de gestionar la recuperación de vida de EDD0
- **"PlayerMovement":** esta clase se encarga del movimiento de EDD0
- **"GenericScreenUi"**: esta clase se encarga de gestionar la pantalla "física" de EDD0

Una de las decisiones de diseño fue que todas las clases encargadas de la gestión del personaje solo dependieran entre ellas, por lo que no saben de la existencia de otras clases más allá de su directorio. Por otro lado, el resto de las mecánicas sí que pueden hacer peticiones a estas clases, así como consultar sus valores.

5.4.2.1 Creación del personaje

La función principal de esta clase es la de otorgar a EDD0 el aspecto seleccionado en el menú de inicio. Para ello, accede a la instancia de la partida guardada para sacar el índice del modelo escogido, posteriormente realiza un cambio de sus materiales como se pue de vere en Ilustración 93 por los de un modelo con la paleta deseada.

```
oldMaterials = modelObjects[i].GetComponent<Renderer>().materials;
newMaterials = modelTextures[model].transform.Find(bodyParts[i-1]).gameObject.GetComponent<Renderer>() // Renderer
    .sharedMaterials; // Material[]
SetMaterials(oldMaterials, newMaterials);
```

Ilustración 93: cambio de materiales de EDD0

Además de gestionar el aspecto, también se encarga de inicializar la posición inicial de la partida. Esta dependerá de si en el juego se llegó a la zona de nexo y utilizó el dialogo principal. En ese caso cada vez que se inicie la partida se empezará en la zona de Nexo, sin embargo, si no se utilizó el dialogo, al iniciar partida aparecerá en la zona inicial como si no se hubiera empezado la partida.

5.4.2.2 Player Values

Como se explicó en el apartado 5.4.2 Implementación del personaje jugable la clase "`PlayerValues`" es la clase más importante del personaje ya que esta actúa como contenedor de información relevante. Del mismo modo, también se encarga de ofreces a otras clases mecanismos para facilitar acciones.

Valores relevantes

- **"`IsGrounded`"**: este valor indica si el jugador está de pie y sobre suelo o no. A este se accede mediante la función "`GetIsGrounded`".

  El funcionamiento de detección de la variable "`IsGrounded`" es el que se puede observar en Ilustración 94. El funcionamiento es el siguiente, Se lanza un rayo en dirección a los pies de EDD0 partiendo desde una posición arbitraria, si esta colisiona con un objeto de la capa "Ground" y su `vector Up`, apunta perpendicularmente al techo, entonces significa que está de pie, se puede mover y por lo tanto la variable "`IsGrounded`" tiene un valor verdadero. Si por lo contrario no se cumplen las condiciones anteriores significa que no está de pie y por lo tanto no se podrá mover y la variable tendrá un valor falso.

```
private void CheckIfGrounded()
{
    RaycastHit hit;
    if (Physics.Raycast( origin: isgroundedPos.position + RayOffset,
            transform.TransformDirection(Vector3.down), out hit, raySize, (int)colisionLayers))
    {
        if (!isGrounded)
        {
            SetCanMove( val: true);
            isGrounded = true;
            stucked = false;
            if (transform.up == Vector3.up)
                _rigidbody.constraints = _originalRigidBodyConstraints;
        }

        if (!hit.transform.gameObject.CompareTag("Stair"))
        {
            lastValidPos = transform.position;
            stuckTimer.Restart();
        }
    }
    else
    {
        if (isGrounded)
        {
            SetGear(1);
            SetCanMove( val: false);
            _rigidbody.constraints = RigidbodyConstraints.None;
            isGrounded = false;
        }

        stuckedPos = transform.position;
    }
}
```

Ilustración 94: método IsGrounded

- **"`Stucked`"**: esta variable controla si el jugador está atascado y no se puede mover, la forma en la que detecta si el jugador está atascado es la siguiente. La primera condición que se debe dar es que el jugador no este de pie, es decir que "`IsGrounded`" valga falso, ya que esto significa que no se pude mover. La siguiente condición que se debe dar es que el jugar haya intentado moverse y no lo haya conseguido, es entonces cuando tras haber intentado escapar y no haberlo conseguido, la posición del personaje se reiniciará haciendo un efecto de disolución y volviendo a la última posición válida. Por otro lado, se ha implementado otra forma de recuperarse, esta está implementada en la clase "`General`

`Inputs`", que como se vio en el aparatado 5.3.2 Inputs, recibe todos los inputs, aunque no estén activados. Esta forma de recuperarse solo se activa una vez el jugador está atascado, y la forma en la que se sale de este estado es pulsando repetidamente el espacio, esto como se ve en Ilustración 95 lo que hace es impulsar al personaje a su posición natural.

```
if (playerValues.GetIsStucked())
{
    if (timer.Elapsed.TotalSeconds > 1)
    {
        timer.Restart();
        playerValues._rigidbody.AddForce(Vector3.up * 12, ForceMode.Impulse);
        playerValues._rigidbody.AddTorque(
            Vector3.Cross( lhs: Vector3.up,  rhs: -playerValues.transform.up) * 10,
            ForceMode.Impulse);
    }
}
```

Ilustración 95: recuperación del estado Stucked

- **Marchas**: las marchas son el método del juego de indicar la velocidad del personaje, estas pueden tener un valor desde 0 hasta 4 siendo 0 marcha atrás y 4 máxima velocidad. La clase "`PlayerValues`" otorga una serie de mecanismos para cambiar de marcha de forma adecuada.
- **Luces**: la variable "`LightsOn`" indica si el jugador tiene las luces encendidas o no.
- **"`Lives`"**: la variable "`Lives`" indica las vidas actuales del jugador, teniendo un máximo de 4 vidas las cuales se regeneran pasados unos segundos. Como se explicará más adelante.
- **"`GameData`"**: este objeto corresponde con los datos de la partida actual, de esta forma es accesible de forma sencilla para el resto de las clases.
- **"`Dead`"**: esta variable indica que el jugador ha perdido todas sus vidas. Una vez puerto reaparecerá en una posición segura. La forma en la que pierde una vida es mediante el método "`RecieveDamage`", el cual se invoca cuando una bala proveniente de un enemigo o de un láser, impacta sobre EDD0, el causante del daño es también el responsable de indicar la posición segura donde el personaje debe reaparecer.
  cabe destacar que cuando EDD0 recibe daño, esto se ve representado de forma visual mediante un filtro de la cámara y ruido.

```
public void RecieveDamage(Vector3 spawnPos)
{
    if (lives > 0)
    {
        lives--;
        playerLives.Damage();
        NotifyCameraLives();
        if (lives <= 0)
            Die(spawnPos);

    }
}
```

Ilustración 96:recievedamage

Ilustración 97: visualización de poco daño, daño medio y mucho daño

5.4.2.3 Acciones del personaje

El patrón de diseño “Observer” es ampliamente utilizado en el desarrollo de software para establecer una relación de dependencia entre un sujeto y múltiples observadores. En este patrón, los observadores se suscriben al sujeto y este les notifica sobre los eventos o cambios relevantes que ocurren. [11]

Esto permite que la implementación del sujeto esté desligada de la implementación de los observadores permitiendo una mayor flexibilidad y robustez.

En el caso del personaje, lo que se necesitaba era una forma de realizar ciertos eventos ante ciertas acciones, por ejemplo, que cuando el personaje sube de marcha, se reproduzca un sonido, o que cada vez que reciba daño se muestren los filtros de Ilustración 97. Es por ello por lo que se decidió implementar el patrón observador.

En primer lugar, se tiene una clase la cual hereda de una clase abstracta sujeto la cual se puede ver en Ilustración 98 esta está formada por una lista de observadores y los mecanismos para añadir nuevos observadores y notificar acciones.

```
public abstract class Subject : MonoBehaviour
{
    private List<IObserver> _observers = new List<IObserver>();

    15 usages  Carlos Chamizo Cano
    public void AddObserver(IObserver observer){...}

    Carlos Chamizo Cano
    public void RemoveOverserver(IObserver observer){...}

    Frequently called  14 usages  Carlos Chamizo Cano
    protected void NotifyObservers(PlayerActions playerAction){...}
}
```

Ilustración 98: clase sujeto

Por otro lado, se tiene la interfaz “`IObserver`”, la cual declara el método “`OnNotify`” que es donde los observadores recibirán las notificaciones del sujeto.

Las diferentes acciones están declaradas en el ENUM “`PlayerActions`”, la cual contine todas las diferentes acciones básicas que puede hacer el personaje.

Dentro del juego, los sujetos son la clase “`PlayerValues`” y las clases “`MachineGun`”, ya que, de esta forma, se pueden cubrir todas las posibles acciones necesarias. Los observadores corresponden con efectos que acompañan estas acciones, como pueden ser animaciones, sonidos, partículas…

Un ejemplo de uso concreto es el que se puede ver en Ilustración 99 donde al recibir las acciones "Aim" y "ChangeShootingMode" realizan el sonido correspondiente.

```
Frequently called  0+1 usages  Carlos Chamizo Cano  More...
public void OnNotify(PlayerActions playerAction)
{
    if (playerAction is PlayerActions.Aim)

    {
        audioSource.clip = clips[0];
        audioSource.Play();
    }
    else if (playerAction is PlayerActions.ChangeShootingMode)
    {
        audioSource.clip = clips[1];
        audioSource.Play();
    }
}
```

Ilustración 99: ejemplo de observador

5.4.2.4 La pantalla del personaje

Como se puede apreciar en Ilustración 78, EDD0 es básicamente una pantalla con piernas, por lo que aprovechando el diseño se decidió usar esta pantalla para mostrar información relevante del juego. Por lo que se implementó la clase "GenericScreenUi".

Esta clase está formada por un componente "TextMeshPro", es decir, texto. Este tiene dos funciones principales, la primera es la de mostrar la cara del personaje y la segunda es mostrar texto importante dependiendo de la necesidad.

En un principio la cara, se iba a hacer con un cambio de la textura de la pantalla, sin embargo, haciendo esto había dos problemas, el primero es que se complicaba en exceso las animaciones a la vez que era más costoso, y el segundo problema era que se perdía el shader que usa la pantalla para simular una televisión antigua. Por lo que finalmente se optó por utilizar texto para dar forma a la cara de EDD0, consiguiendo como se verá a continuación resultados muy buenos.

Respecto a la implementación de la cara, esta tiene 4 posibles estados como se puede ver en Tabla 12 los cuales se gestionan a través del método "Face" de esta clase. En este método, básicamente los que se hace es poner en su pantalla una cara normal, ya puede ser tanto Ilustración 100 como Ilustración 101. Tras un cierto tiempo marcado por un temporizador, este realizará la acción de pestañear, la cual durará un corto periodo de tiempo y tras esto, volverá a su estado de cara normal. Finalmente, si EDD0 por algún motivo se cae, este mostrará una cara de asustado como se puede ver en Ilustración 103. . La forma en la que puede saber si se ha caído es comprobando los valores de las variables "IsGrounded" y "isStucked" en "PlayerValues".

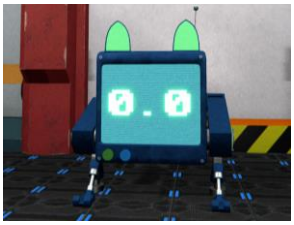


Ilustración 100: cara normal 1

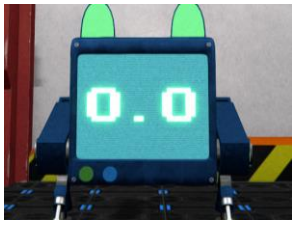


Ilustración 101: cara normal 2

Ilustración 102: cara pestañeo

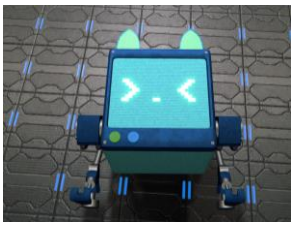


Ilustración 103. cara asustada

Tabla 12: caras de EDD0

El otro uso de esta clase es el de mostrar texto, porque aunque se use para mostrar caras de EDD0, el mecanismo no deja de ser un componente de texto. El principal uso de esta otra forma de mostrar texto es el de mostrar los títulos de los minijuegos como se puede ver en

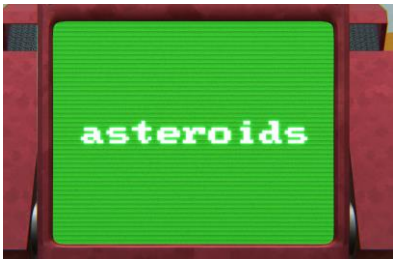


Ilustración 104: título minijuego de asteroides

5.4.2.5 Vidas

En los anteriores puntos se han dado algunas pinceladas de cómo funciona el sistema de vidas, sin embargo, en este apartado se explicará de forma completa.

En primer lugar, como ya se ha mencionado antes, la variable que lleva el contador de las vidas está en la clase "`PlayerValues`", esta clase además implementa un mecanismo para recibir daño. Sin embargo, la recuperación de vida no se realiza en esta misma clase, sino que se ha implementado la clase "`PlayerLives`" la cual se encarga de recuperar vida cuando se ha pasado cierto tiempo sin recibir daño. Esta clase implementa la interfaz "Observer" de forma que, en caso de recibir daño, el temporizador de recuperación de vida se reinicia.

5.4.2.6 Movimiento y físicas

El movimiento es otro de los aspectos más importantes ya que este debe ir acorde con las animaciones del personaje además de ser intuitivo.

El movimiento del personaje se planteó como un coche radio control, es decir, que tiene diferentes velocidades, pero sin embargo solo puede moverse hacia delante, refiriéndome a que no tiene movimiento lateral. Hay varios aspectos que se tuvieron en cuanta a la hora de diseñar el movimiento del personaje:

- **No puede deslizar con el suelo**, es decir que las animaciones deben ir acorde con la velocidad del personaje y viceversa.
- **El jugador debe tener control absoluto de la mecánica de movimiento que se le ofrece**, por ello, si el jugador quiere parar, el personaje debe parar, no debe haber ningún comportamiento fuera del control del jugador.

- **El personaje debe detectar cuando se ha chocado con una pared y parar en consecuencia.** Esto es porque desde el punto de vista del jugador, es antinatural que el personaje esté intentando avanzar y la amidación de caminar se esté ejecutando.

En primer lugar, voy a explicar cómo funciona el movimiento básico del personaje. Para entender esto primero hay que entender los elementos del personaje. Entre los elementos del personaje se encuentran el modelo el cual realiza las animaciones, y un componente "RigidBody".

A la hora de gestionar el movimiento del personaje hay dos alternativas obvias, la primera es el componente "CharacterController" el cual implementa funciones de movimiento, así como la variable "`IsGrounded`" de forma automática además de una forma de subir escaleras bastante buenas. Pero, sin embargo, ¿por qué no se ha elegido esta opción? La respuesta es porque no implementa físicas de ningún tipo y para el juego y sobre todo para bajar escaleras era muy importante que el personaje incorporara físicas. Por ello se optó por la segunda opción, un "RigidBody". Esta opción a pesar de no contar con elementos importantes implementa una muy buena simulación de físicas realista. Además de métodos para interactuar con el "RigidBody" por medio de fuerzas.

El movimiento se gestiona a través de la clase "`PlayerMovement`" y esta se aplica en la dirección de la cámara la cual se calcula mediante el método "`GetMoveDirection`" el cual se encarga de interpolar el ángulo de la dirección de la cámara con el ángulo de dirección del personaje. Esto además de la dirección permite que el personaje gire de forma suave.

Una vez teniendo la dirección queda aplicar el movimiento. Para aplicar el movimiento de un "RigidBody", es muy importante que los cálculos se hagan siempre en el mismo Intervalo de tiempo ya que de lo contrario se obtendrán resultados extraños e indeseados. Para ello como se ve en Ilustración 105, se añade la fuerza deseada en la dirección deseada en forma de un cambio de velocidad. La resta que se puede ver en la imagen es debido a que la velocidad debe ser constante por lo que se resta la velocidad previa al cálculo.

```
Vector3 moveDirection = GetMoveDirection();
_rigidbody.AddForce(moveDirection * _playerValues.GetSpeed() - _rigidbody.velocity,
    ForceMode.VelocityChange);
```

Ilustración 105: movimiento del personaje

Respecto a la forma en la que se ha conseguido que las animaciones del personaje no deslicen con el suelo es mediante la variación y modificación de las velocidades correspondientes a cada marcha del personaje, así como del ajuste de las correspondientes animaciones.

En cuanto la detección de colisiones con las paredes se realiza mediante el método "`CheckIfStomp`" el cual lo que hace es lanzar un rayo al frente con un alcance limitado, de forma que, si se colisiona con una pared, ya sea de forma frontal o trasera, se parará el movimiento y por ente la animación, en definitiva, se hace transición a la marcha 1 (parado) como se puede ver en Ilustración 106.

```
private void Update()
{
    if (_playerValues.GetGear() > 1 && CheckIfStopm() || _playerValues.GetGear() == 0 && CheckIfBackStopm())
    {
        _playerValues.StopMovement();
    }
```

Ilustración 106: código de colisiones con la pared

Además, la clase "`PlayerMovement`" también implementa el uso de la cuarta marcha, pero esto se explicará en detalle en el apartado 5.4.4 Diferentes modos del personaje.

5.4.3 Cámaras

La cámara es otro aspecto muy importante de un juego, sobre todo si es en un juego donde los escenarios son de interior.

Debido a mis pasadas experiencias desarrollando el juego "Penguin Age" [12] para la Indie Spain Jam, vi la importancia de la cámara y lo que suponía una mala implementación como ese era el caso. Ya que una mala implementación de las colisiones puede llevar a marear al jugador y estropear la experiencia de juego.

Por este motivo decidí estudiar más acerca de las cámaras orbitales en tercera persona, para que este juego no tuviera el mismo problema. Entonces fue cuando encontré el tutorial de Jasper Flick [13] en el cual se explica cómo hacer una buena implementación de una cámara orbital.

La idea de la cámara, como se puede ver en Ilustración 78 a nivel argumentativo, fuera el dron que EDD0 tiene encima suya, es por ello por lo que las diferentes cámaras que se han creado han buscado crear este efecto. A continuación, se detallarán las diferentes cámaras usadas:

- **Cámara Orbital**: la cámara orbital es la cámara principal del juego, la cual se puede ver en Ilustración 107, y que cual cuanta con las siguientes características: Recogida de inputs de ratón para el movimiento de la cámara, colisiones con objetos y centrado del objetivo dinámico, Esta cámara se ha implementado siguiendo el tutorial de Jasper Flick [13] sin embargo explicaré de forma breve la implementación de sus diferentes características a continuación.

Ilustración 107: vista orbital EDD0

  - **Colisiones de la cámara**: la función de las colisiones de la cámara es la de no atravesar paredes cuando la cámara entra en contacto con una. La implementación es muy sencilla, como se puede ver en Ilustración 108 la idea es lanzar un rayo en la dirección opuesta a la que la cámara está mirando, en caso de que colisione la

posición de la cámara se ubicará en el lugar de impacto del rayo (con un pequeño desfase).

```
if (Physics.Raycast( origin: focusPoint, -lookDirection, out RaycastHit hit, distance, (int) CollisionLayers))
    lookPosition = focusPoint - lookDirection * (hit.distance - colOffset);
```

Ilustración 108: colisiones de la cámara

- **Centrado del objetivo dinámico**: Este es un sutil efecto que hace que la cámara se recentre en el objetivo de forma suave, la forma en la que se realiza esto es mediante una interpolación entre el lugar en el que está apuntando la cámara y el que debería estar apuntando. Esta interpolación se hará más rápido cuanto más alejado se esté del punto de enfoque tal y como se puede ver en Ilustración 109.

```
void UpdateFocusPoint()
{
    previousFocusPoint = focusPoint;
    Vector3 targetPoint = focus.position;
    if (focusRadius > 0f)
    {
        float distance = Vector3.Distance( a: targetPoint, b: focusPoint);
        float t = 1f;
        if (distance > 0.01f && focusCentering > 0f)
            t = Mathf.Pow( f: 1f - focusCentering, p: Time.unscaledDeltaTime);

        if (distance > focusRadius)
            t = Mathf.Min( a: t, b: focusRadius / distance);

        focusPoint = Vector3.Lerp( a: targetPoint, b: focusPoint, t);
    }
    else
        focusPoint = targetPoint;
```

Ilustración 109: recentrado del punto de foco

- **Inputs del ratón**: la cámara también implementa una forma de recogida de Inputs del ratón, este se usa para calcular la posición de la cámara. Además de los inputs del ratón se ha modificado el código de forma que desde giros del cubo se pueda girar la cámara en intervalos de 15 grados. En el método que se puede ver en Ilustración 110 se puede observar la forma en la que se interpola una nueva posición tras girar el cubo.

```
Frequently called  1 usage  Carlos Chamizo Cano
private float UpdateRotateCamera()
{
    currentRotation = Quaternion
        .Lerp( a: Quaternion.Euler( x: 0, y: currentRotation, z: 0), b: Quaternion.Euler( x: 0, y: targetRotation, z: 0),
            t: 5 * Time.unscaledDeltaTime) // Quaternion
        .eulerAngles.y; // float

    if (Mathf.Abs( f: currentRotation - targetRotation) < 0.1f)
        updateRotation = false;

    return currentRotation;
}
```

Ilustración 110: giros de la cámara con el cubo

- **Cámara en primera persona**: La cámara en primera persona, la cual se puede ver en Ilustración 111, es una modificación del cámara orbital la cual se le han quitado todas las funcionalidades innecesarias para su propósito y se ha dejado fija en un punto. Sin embargo, esta cámara es exclusiva del modo de disparo el cual se explicará en 5.4.4 Diferentes modos del personaje.

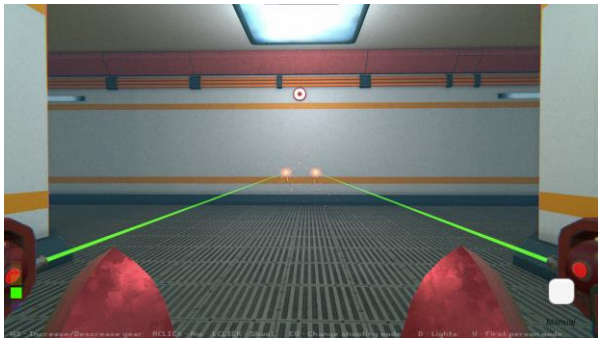


Ilustración 111: cámara en primera persona

- **Cámara de pantalla**: esta cámara, la cual se puede ver en Ilustración 112, no tiene nada especial, tan solo que es un cámara fija apuntando a la pantalla del personaje, esta se activa durante lo minijuegos de forma que se pueda interactuar de una forma fácil con ellos.

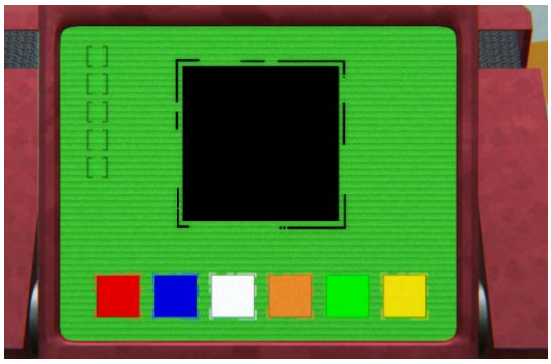


Ilustración 112: cámara de pantalla

- **Cámara de transición**: esta cámara, la cual se puede vere en Ilustración 113, hace de transición entre cámaras. Su función es la de interpolar la cámara desde la posición y rotación de la cámara antigua hasta la posición y rotación de la cámara nueva de forma que no haya cambios de cámara bruscos.

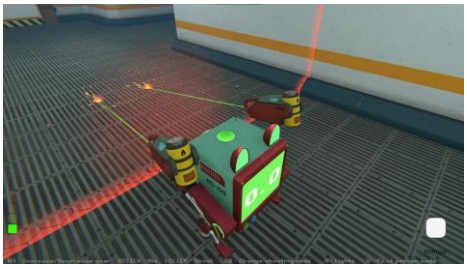


Ilustración 113: cámara de transición

Además de las cámaras se ha implementado 2 clases que hacen de interfaz entre el usuario y las cámaras, estas son "`CameraController`" y "`CameraChanger`".

"`CameraController`" se encarga de proporcionar todos aquellos mecanismos básicos que pueden necesitar tanto la cámara en primera persona o la cámara orbital. Algunas de sus funciones son: Congelar la cámara, Rotaciones para dar funcionalidad con el cubo, o agitación de la cámara para los disparos. "`CameraChanger`", por otro lado, permite hacer la transición entre las diferentes cámaras.

Finalmente me gustaría mencionar que a pesar de que el Dron del personaje está modelado y texturizado no se llega nunca a ver en el juego ya que este está desactivado, sin embargo, de cara a un futuro, este se usará para que se intuya su sombra en el escenario.

5.4.4 Diferentes modos del personaje

EDD0, como ya se explicó en 5. 1 Documento de diseño del juego dispone de 5 posibles modos en los que se puede encontrar. En este apartado se enumerarán estos 5 modos y se explicará su implementación en detalle.

También, como se explicó en el apartado 5.3.2.1 Current input es importante conocer que cada uno de estos modos del personaje corresponde con un Input concreto.

5.4.4.1 Tutoriales

Para cada uno de los modos que se explicarán a continuación no hay tutoriales guiados en el juego, sino que se muestra en la parte inferior de la pantalla como se puede ver en Ilustración 114, de esta forma el usuario siempre sabe que cosas puede hacer y los controles tanto de ratón y teclado, como de Cubo.

Ilustración 114: tutorial de controles del modo disparo

La forma en la que se cambian los controles entre ratón y teclado es mediante la detección de cualquier input. En caso de ser un input de cubo, se pone el tutorial del cubo y en caso de ser un input de teclado se pondría el tutorial de teclado

5.4.4.2 Modo movimiento normal

Este modo de movimiento corresponde con el movimiento básico del personaje, desde este se tendrán cuatro marchas, quedando la quinta marcha bloqueada. También desde este modo se pueden encender y apagar las luces de EDD0, lo cual permite explorar zonas oscuras. A parte de estos dos aspectos, hace uso de todo lo anteriormente mencionado en el punto 5.4.2.6 Movimiento y físicas.

5.4.4.3 Modo Disparos

Este modo es por así decirlo el modo de combate de EDD0, cuando entra en este modo, mediante un efecto de disolución, aparecen junto a él 2 armas las cuales se pueden ver en Ilustración 115. Estas son dos armas las cuales se pueden controlar con el movimiento de la cámara de EDD0.

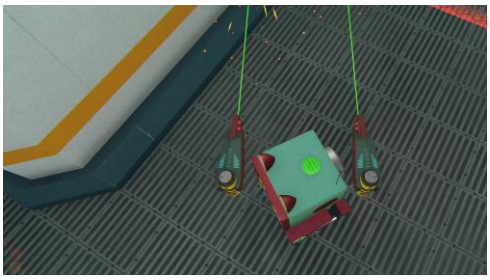


Ilustración 115: armas de EDD0

Estas armas cuentan con varias funcionalidades a destacar que hacen que sean cómodas e intuitivas de usar. Estas características las enumeraré a continuación.

- **Laser y giro del tambor**: ambas armas están equipadas con un láser y puntero al final del láser el cual indica donde está impactando el mismo. Este laser se ha sacado junto con algunas de las partículas utilizadas en este juego del asset RPG VFX Pack de HovlStudio [14] este asset viene con un gran número de efectos entre los que se encuentra el Laser que se ha utilizado.
  Este laser en definitiva es un "line renderer" el con un efecto de partículas al final de este.
  El láser es importante pero no solo porque permite apuntar, sino también porque es un indicador de que se puede disparar.
  Además del láser, cuando se apunta, el tambor de la propia arma gira, esto se ha implementado así para simular las famosas ametralladoras. La forma en la que está implementada esta funcionalidad la cual puede pasar desapercibida es mediante un "RigidBody", que como se explicó en 5.4.2.6 Movimiento y físicas implementan muy bien las físicas. Por lo tanto, para que gire de forma natural tan solo se le debe poner una fuerza de rozamiento alta y aplicar la fuerza de torque que se muestra en Ilustración 116.

```
Frequently called  1 usage  Carlos Chamizo Cano
private void RotateDrum()
{
    drumRigidbody.AddRelativeTorque(x: 0, y: 0, z: drumSpeed, ForceMode.Acceleration);
}
```

Ilustración 116: rotación de tambor

- **Retroceso**: debido a que es un arma, este debe tener retroceso. La forma en la que se ha implementado el retroceso es relativamente compleja. En primer lugar, las armas en sí mismas tienen un componente "RigidBody", sin embargo, en estado de reposo tienen la posición fija. Es en el momento de disparo cuando las físicas empiezan a funcionar.

  En primer lugar, para crear el retroceso se añade una fuerza de impulso en el sentido del disparo. Sin embargo, es necesario que las armas vuelvan a su lugar de partida, es aquí, tal y como se ve en Ilustración 117 que, sobre las fuerzas del arma, se aplica la ley de Hook [15]. De forma que cuando el arma está suficientemente cerca de su posición de partida, se ancla a esta.

  Además de esta implementación, se ha añadido partículas en el momento justo de disparo como se ve en Ilustración 118.

```
private void HookPosition()
{
    var position1 = placeHolder.position;
    var centerOfMass = _rigidbody.position;
    float magnitude = (centerOfMass - position1).magnitude;
    Vector3 auxForce = -K * magnitude * (centerOfMass - position1) / magnitude;

    if (!float.IsNaN(auxForce.x) || !float.IsNaN(auxForce.y) || !float.IsNaN(auxForce.z))
        _rigidbody.AddForce(auxForce, ForceMode.Acceleration);

    if (hookTimer.Elapsed.TotalSeconds > hookCooldown && magnitude < 0.3f)
    {
        hookTimer.Stop();
        hookTimer.Reset();
        _rigidbody.isKinematic = true;
    }
}
```

Ilustración 117: método Hook position

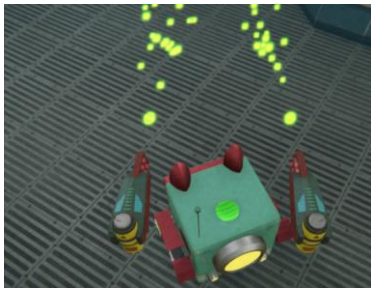


Ilustración 118: retroceso con partículas del arma

- **Modos de disparo:** el arma cuenta con 3 modos de disparo.
    - **el primer modo es el manual**, en este modo, se disparan balas de una en una a petición del jugador, sin embargo, para que no se pueda abusar de la mecánica, hay un tiempo de enfriamiento entre disparo y disparo. Este modo de disparo está pensado para la mecánica de interruptores ya que como se explicará en 5.5.7 Interruptor, solo se necesita un disparo para activarlo.
    - El siguiente modo de disparo es el **disparo por ráfaga**, donde saldrán 3 balas en una forma de ráfaga. Este modo de disparo también está capado por un tiempo de enfriamiento y está pensado para el combate contra enemigos o jefes.
    - El último modo es el **disparo automático**, este no requiere de interacción del jugador ya que, por cada intervalo de tiempo, disparará una bala.
- **Colisiones con el escenario:** debido a que las armas en ocasiones podían atravesar paredes, para evitar problemas, se solucionó ideando un mecanismo para que esto no ocurriera, este se realiza en la clase "`MachineGunRotation`" donde cómo se puede ver en Ilustración 119, se realiza una comprobación lanzando una serie de rayos alrededor del personaje los cuales en caso de impactar aparta las armas. Esto se puede ver mejor en

```
private void Update()
{
    if (!(Col(playerValues.transform.forward) || Col(playerValues.transform.right) ||
          Col(-playerValues.transform.right) ||
          Col((playerValues.transform.right + playerValues.transform.forward).normalized) ||
          Col((-playerValues.transform.right + playerValues.transform.forward).normalized)))
    {
        if (Vector3.Distance( a: placeholder.position,  b: transform.parent.position) > 0.1f)
            placeholder.position = Vector3.MoveTowards( current: placeholder.position,  target: transform.parent.position, factor);
    }
}
```

Ilustración 119: colisiones armas

Ilustración 120: colisión armas

- **Primera persona**: el modo primera persona, como se ve en Ilustración 121, permite al jugador apuntar de forma precisa, además de que permite que apuntar a distintas alturas, cosa que con la cámara en tercera persona no se puede.

Ilustración 121: primera persona con el arma

5.4.4.4 Modo Sigilo

En el modo sigilo, se introduce un elemento el cual es el distractor, el cual se pue de vere en Ilustración 40, este tiene la función de distraer a enemigos como la cámara de seguridad.

La distracción es un cubo verde el cual está compuesto por un "RigidBody" para así poder darle un comportamiento con físicas. La forma en la que funciona es la siguiente, Este se lanza en la dirección de la cámara y desde ese momento ya está operativo. En caso de que el jugador quiera volver a lanzarlo, deberá esperar a escuchar un "pop" saliente junto con unas partículas provenientes de la distracción.

La distracción permanecerá en el lugar donde se lanzó, hasta que se recoja o hasta que un enemigo la dispare. En ese momento la distracción volverá al jugado. Si la distracción está en el rango de visión del jugador, este se moverá rápidamente hasta su posición original, de lo contrario se teletransportará.

5.4.4.5 Modo Carrera

En este modo se desbloquea la 4 marcha, la cual se puede ver en Ilustración 41, esta marcha otorga más velocidad al jugador, sin embargo, esta velocidad, consume energía o estamina.

Esta mecánica se gestiona a través de la clase "`PlayerMovement`". La estamina, se consume de a un ritmo constante mientras se usa la marcha "4", sin embargo, para recuperar energía tan solo hay que bajar de marchas, cuanto más baja sea la marcha, más rápido se recarga la estamina.

La estamina se puede ver de forma visual, en forma de una barra verde, sin embargo, en caso de que se consuma toda la estamina, como se puede ver en Ilustración 122, el personaje bajará de marcha automáticamente y este no podrá volver a subir hasta que se haya rellenado del todo.

Ilustración 122: estamina consumida

5.4.4.6 Modo Minijuegos

Finalmente, en el modo minijuegos, EDD0 se sentará en una plataforma y la cámara pasará a ser la cámara de pantalla. En este modo la pantalla de EDD0 pasará a ser la interfaz del jugador donde aparecerán un minijuego, el cual EDD0 deberá completar. Cabe destacar que cada minijuego tiene sus propios Inputs.

Los minijuegos

Los minijuegos so otra de las mecánicas troncales del juego, estos se diseñaron para que fuera posible jugarlos tanto con ratón y teclado como con el cubo, por lo que su diseño supuso un verdadero reto.

Los minijuegos, están controlados por la clase "`MinigameManager`", cuyas funciones son empezar los juegos, no permitir que se repitan dos minijuegos seguidos y notificar de cuando se ha terminado un minijuego

Los minijuegos por lo general consisten en superar 5 niveles del minijuego, a excepción del minijuego de esperar y el minijuego del patrón. Estos 5 niveles se ven representados en la pantalla de cada minijuego como 5 puntos verdes que se van encendiendo. A continuación, se explicará de forma general los aspectos en cuanto a la implementación más relevantes.

- **Minijuego ajustar los valores:** En este minijuego se mostrarán tres "sliders" los cuales pueden tomar un valor que va desde 0 hasta 1. El objetivo de este minijuego es ajustar los valores de forma que se enciendan los 5 puntos verdes.
  En cada partida el valor objetivo es distinto, por lo que cada partida es distinta.
- **Minijuego Asteroides**: en este minijuego, aparecerán diferentes asteroides, el objetivo es que el jugador aguante las 5 rondas sin ser colisionado. Para la realización de este minijuego se pueden destacar 2 elementos fundamentales:
  - **El jugador**: el jugador tendrá la posibilidad de moverse libremente por el escenario, con la peculiaridad de que puede atravesar paredes apareciendo en la pared contraria a la misma altura, lo que supone una ventaja de movimiento. Si el jugador colisiona con un asteroide, ser reiniciará la partida.
  - **Los asteroides:** los asteroides, por así decirlo tienen comportamiento propio, aunque esté basado en el azar. Estos al comienzo de la partida se generarán en un lugar que esté alejado del jugador ya que de lo contrario esto podría ser injusto en el cambio de rondas. Los asteroides comienzan con un número de vidas las cuales van bajando cada vez que colisiona con una pared. Cuantas menos vidas tenga el asteroide, este se irá poniendo de un color más rojizo hasta que acabe por explotar-

Cuando un asteroide rebota pueden ocurrir tres cosas, la primera es que se le acaben las vidas y explote, la segunda es que rebote en una dirección aleatoria y la última es que reboten en la dirección del jugador.

Cabe destacar que los asteroides parten de un pool propia de asteroides, por lo tanto, solo se crearán los asteroides que sean necesarios, el resto se recogerán del pool.

- **Minijuego Colores**: En este minijuego, el funcionamiento es muy sencillo, al principio de la partida, se asignarán una serie de colores a los botones de la parte inferior de la pantalla como se pueden ver en Ilustración 46, Se deberá pulsar el color que se vea en la parte central de la pantalla.
- **Minijuego No tocar las paredes**: para este minijuego, se ha aprovechado la mecánica creada en el minijuego de Asteroide para buscarle otro enfoque. Este minijuego, está basado en el juego "World Hardest game" [16] un juego donde hay que avanzar sin que golpeen obstáculos. En nuestro minijuego, lo que se ha hecho es darle una vuelta y que haya que llegar desde un punto A hasta un punto B. Al igual que en el minijuego de Asteroides, se podrán atravesar las paredes, en este minijuego también se puede, sin embargo, la dificultad añadida está en los muros. Estos son objetos que tan solo tienen un método que se encarga de comprobar si hay un jugador dentro, para de esta forma reiniciar la partida.
  los grupos de paredes se han agrupado en laberintos, estos laberintos son predefinidos, habiendo un total de 12 laberintos de los cuales se seleccionan 5.
- **Minijuego espera**: este minijuego es el más simple de todos ya que el objetivo es esperar a que pase un tiempo el cual está representado mediante una barra de tiempo.
- **Minijuego de memoria**: Este minijuego se basa en el minijuego de colores. En la imagen central se mostrando una secuencia de colores, tras la cual, se deberá introducir a través de los botones de la pantalla la secuencia. Este minijuego es incremental, lo que significa que en cada nivel se añadirá un elemento a la secuencia a recordar.
- **Minijuego pulsa tan rápido como puedas**: en este minijuego, el objetivo será pulsar rápidamente el botón que se encuentra en el centro del minijuego, a medida que se vaya pulsando el botón, se irá rellenando los 5 puntos, pero, sin embargo, si se acaba el tiempo, se deberá empezar de nuevo.
- **Minijuego iguala el patrón**: en este minijuego se mostrarán dos patrones con las mismas piezas, pero con diferentes disposiciones. El objetivo será que el patrón de la parte inferior coincida con el patrón de la parte superior.
  Para resolver el puzle habrá que pulsar los botones que rodean al patrón inferior. Los botones lo que hacen es desplazar todos los colores una posición en la dirección que se haya indicado y rotando las últimas posiciones a las primeras.
- **Minijuego gira la tuerca**: este minijuego consiste en girar una tuerca 5 veces completas. La forma en la que esto se hace es arrastrando de izquierda a derecha. En cuanto a código se refiere, lo que ocurre es que se compara la posición del ratón con el botón izquierdo pulsado con la anterior posición registrada, de esta forma se sabe la dirección en la que se gira la tuerca. Además, está diseñado para que no se puedan dar infinitas vueltas para atrás de forma que, si el contador de vueltas está a cero y se intenta ir para atrás, este se bloquea.
- **Miniboss**: los minibosses o mini jefes corresponden con el jefe final de una zona, estos presentan una batalla donde por cada turno hay tres fases.

- **Fase de decisión**: En esta fase se decide cual va a ser la acción para realizar por el jugador. Como se ve en Ilustración 123 el jugador debe elegir que acción ve más conveniente realizar. El jugador debe elegir en base a las acciones del jefe, las cuales se muestran en el rectángulo blanco central. Este tiene la posibilidad de atacar, defenderse o no hacer nada. Por otro lado, el jugador podrá atacar o defenderse, además de realizar ataques especiales los cuales tienen mucha más eficacia, pero son más complicados de alcanzar.

Ilustración 123: pantalla de selección Miniboss

- **Fase de juego**: en esta fase una secuencia de letras irá descendiendo a medida que se vayan pulsando en el teclado, con el cubo serían movimientos del cubo en vez de letras. El objetivo es acertar el mayor número de letras posibles ya que de esto depende la eficacia del turno. En la pantalla del minijuego como se ve en Ilustración 124 se muestra cierta información relevante que indica el estado de la fase, es decir, el número de aciertos, el número de fallos, máximo combo de la partida y el combo actual.

  En cada mini jefe del juago hay una dificultad diferente la cual se puede ajustar. La idea es que cada vez que se llegue a una zona nueva se introduzcan más letras del teclado, acabando con todas las letras del teclado al final del juego.

Ilustración 124: fase de juego Miniboss

- **Fase de resultado**: Esta es la etapa final de una ronda, en esta se mostrará la eficiencia de la fase de juego. La eficiencia determina la efectividad de nuestra acción, es decir que si la acción era defensa y no se llega al 50% la defensa fracasará.

Por otro lado, como se mencionó antes las habilidades especiales son más complejas ya que requieren alcanzar el doble de la puntuación normal, sin embargo, si se alcanza, realizarán mucho más daño que las habilidades normales La eficiencia se calcula con el número de aciertos, los fallos y el máximo combo, como se ve en Ilustración 126

Esta fase comienza con una animación la cual varía en función de los resultados. Tras la animación se restará a la barra de vida del jefe enemigo el porcentaje correspondiente con nuestro ataque.

Ilustración 125: fase de resultados.

```
Frequently called | 1 usage | Carlos Chamizo Cano
private float CalculateEfficiency()
{
    float correct = _correctCount;
    float incorrect = _inCorrectCount * 0.5f;
    float maxCombo = _maxCombo;
    float sequenceLength = _maxSequence;

    float comboMultiplier = 1 + (maxCombo / sequenceLength) * 0.1f;
    float overallPuntuation = Mathf.Max(a: 0, b: (correct - incorrect) * comboMultiplier);
    return (overallPuntuation / sequenceLength) * 100;
}
```

Ilustración 126: cálculo de la eficiencia

Estas etapas se repetirán hasta que o bien el jugador pierda sus tres vidas o bien cuando el jefe pierda toda su vida.

- **Código**: esta es una mecánica que se repite en todos los niveles. Al entrar a una zona, habrá una serie de pantallas como la que se ve en Ilustración 127, la idea es que el jugador recuerde todos estos dígitos para posteriormente introducirlos en el minijuego y abrir una puerta.

Habrá cuatro pantallas concretamente como estas y tienen abajo en la derecha un símbolo indicativo de la posición en la que se debe introducir en el minijuego final.

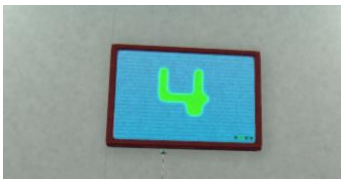


Ilustración 127: dígito del código

El minijuego en si es muy simple, como se pue de ver en Ilustración 53 tan solo se ha de introducir la clave y entonces se tendrán dos opciones, o bien el código es incorrecto in se nos muestra un texto en rojo con la palabra “Incorrect”, o es correcto y se nos muestra un texto con las letras en verde con las letras “correct” y se abre la puerta.

5.4.4.7 ¿Cómo cambia de modo?

Se han explicado los modos en los que EDD0 puede estar, pero, sin embargo, falta por explicar cómo se accede a estos modos. La respuesta es mediante los “`Input Triggers`”, estos como se pueden ver en Ilustración 128, son muros invisibles con un color representativo, siendo, el rojo, el color que activa el modo disparo, el blanco, el que activa el movimiento normal, el amarillo, el que activa el modo sigilo y el verde el que activa el modo carrera. Esto se hizo así para que de esta forma el jugador tras usar una vez esta mecánica, supiera cómo funciona el mecanismo.

Estos “`Input Triggers`” son excluyentes por lo que no se puede estar en dos estados a la vez.

Por otro lado, el modo minijuegos no se activa mediante un “trigger”, sino que se activa aproximándose a una plataforma similar a la Ilustración 87.

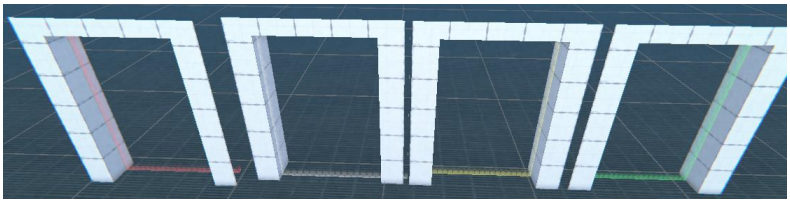


Ilustración 128: Input Triggers

5.5 Desarrollo de las mecánicas del escenario.

5.5.1 Introducción

En esta sección se abordarán las mecánicas de juego que se centran en la interacción del personaje con elementos externos a sí mismo. Estas mecánicas permiten al personaje interactuar con el entorno y otros objetos dentro del juego.

5.5.2 Terminales

Los terminales son un tipo de puzle recurrente en el juego, y sirven como acceso a los minijuegos. Esta mecánica se compone de dos elementos principales: una puerta y terminales. El objetivo del jugador es interactuar con los terminales superando minijuegos para desbloquear la puerta y acceder a la siguiente zona.

La labor de la puerta, la cual se puede ver en Ilustración 129 es bloquear el paso al jugador, obligando al personaje a arreglar el resto de las terminales para poder avanzar. Además de esto, la puerta dará información acerca del número de terminales que faltan para completar el puzle.

Ilustración 129: puerta de terminales

Por otro lado, los terminales, como se pue de ver en Ilustración 130 son los elementos que se deben arreglar. La forma de arreglarlos es jugando a un minijuego. Los minijuegos, como se explicó en el apartado Los minijuegos estos siempre serán 4 por cada puerta, sin embargo, al solo haber 9 minijuegos es muy posible que estos se repitan.

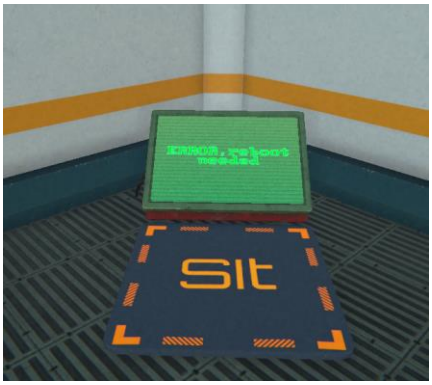


Ilustración 130: terminal de minijuegos

Una vez terminados los 4 terminales la puerta se abrirá y se podrá avanzar en el nivel.

5.5.3 Diálogos y objetivos

Los diálogos y los objetivos son elementos fundamentales en un juego con narrativa, ya que estos nos permiten enterarnos de la historia, así como saber en cada momento que debe hacer el personaje.

5.5.3.1 Objetivos

El sistema de objetivos es más simple de lo que a priori podría parecer, este es el encargado de indicarle al jugador la acción recomendada al jugador para que este avance en el juego. Este sistema está formado por dos partes. La clase "`Objetives`" y la clase "`Objetive Trigger`".

Visualmente estos se pueden ver tal y como se ve en Ilustración 131, es decir, un texto pequeño junto con un círculo amarillo en la parte superior izquierda de la pantalla.

Ilustración 131: ejemplo de objetivo

La clase "`Objetives`", es la encargada de gestionar directamente los objetivos con la GUI. Este en primer lugar al empezar una partida y establece un primer objetivo, que corresponde con la última zona que tengamos desbloqueada en nuestra partida.

La forma en la que se establecen objetivos es siempre la misma, y se realiza mediante la corrutina "`ShowNewObjetiveCoroutine`", esta corrutina muestra durante tres segundos el objetivo en la pantalla de juego para después esconderlo. Sin embargo, el objetivo será visible en el menú de pausa.

La clase" Objetives" ofrece los mecanismos necesarios a la clase "`ObjetiveTrigger`" para poner o quitar objetivos según se necesite. La utilidad de esta última clase mencionada es la de dar una herramienta al diseñador para establecer objetivos de forma sencilla. Esta clase va junto a un "Trigger", el cual, al ser atravesado, pondrá o quitará un objetivo tal y como se ve en Ilustración 132.

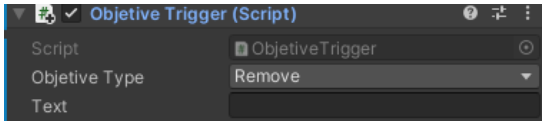


Ilustración 132: objetive Trigger

5.5.3.2. Diálogos

Bajo mi experiencia esta era l primera vez que hacía un juego que contuviera diálogos, por lo que me propuse crear mi propio sistema de diálogos de forma que además de crear una funcionalidad para el juego, me sirviera como experiencia.

Experiencia general

Lo primero antes de entrar en la implementación es conocer como es la experiencia de los diálogos. Las zonas donde se mostrará un diálogo son muy reconocibles ya que se ha modelado un elemento con forma de icono de mensaje, el cual se puede ver en Ilustración 133. Estos al acercarse el personaje a ellos, desaparecerán gracias al shader que se ha implementado con este propósito.

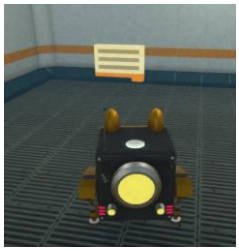


Ilustración 133: modelo de dialogo.

Una vez dentro del diálogo, se puede ver que aparece una caja de texto, así como el personaje Alice la cual nos cuenta algo. Sin embargo, no solo hay diálogos con Alice, sino que los diálogos se han diseñado de tal forma que sean lo más genéricos posibles.

Hay dos tipos de textos, por un lado, están los diálogos de texto plano donde se muestra un texto en el cual solo se puede continuar como el que se ve en Ilustración 134. Y por otro lado están las preguntas, estas dan dos opciones las cuales llevan a otros diálogos.

Ilustración 134: ejemplos de dialogo plano y dialogo con pregunta

Finalmente, tras acabar el diálogo pueden ocurrir dos cosas, o bien que este desaparezca o bien que se pueda interactuar con el otra vez.

Implementación

Ahora que conocemos la experiencia de un diálogo podemos pasar a la implementación. La estructura de una conversación se divide en varios elementos:

- **Documento TXT con el diálogo**: estos son los diálogos en bruto, estos están escrito en un formato que ha sido escrito con el propósito de específico de este sistema de diálogos.

  Por un lado, tenemos los diálogos de texto plano, en estos se debe especificar en la primera línea debe ir un símbolo "- "esto indica que a continuación hay un diálogo. La segunda línea indica que tipo de diálogo es, en este caso al ser un texto plano, le corresponde la letra "t", en la tercera línea debe ir el nombre de la persona que habla, en este caso es Alice, en la cuarta línea debe ir el texto que se quiere mostrar. Los textos al ser mostrados por "`TextMeshPro`", admiten formatos especiales como el que se ve en Ilustración 135 por lo que en el resultado final adquiriría ese color. Finalmente, en la última línea debe ir el índice del siguiente mensaje de forma que la conversación se pueda continuar.

```
---- 1
t
Alice
I'll start from the beginning. My name is <color=#b00e26> Alice </color> and as I told you before I am the security protocol of this shelter.
2
```

Ilustración 135: ejemplo de texto plano

  Por otro lado, se tienen los diálogos del tipo pregunta, estos tienen un formato ligeramente diferente a lo de texto plano. En primer lugar, está el símbolo "- "el cual indica que a continuación va un diálogo. En la segunda línea, se especifica mediante una "q" que lo que

se va a leer es una pregunta. En la tercera línea, va el nombre de la persona que habla y en la cuarta línea, la pregunta que realiza. Las dos siguientes líneas corresponden con las opciones de respuesta y finalmente las dos últimas líneas corresponden con el índice de los siguientes diálogos respecto a la opción respondida.

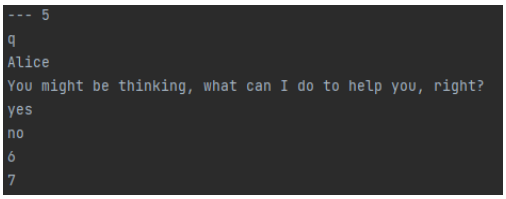


Ilustración 136: Ejemplo de diálogo de pregunta

- **Clase "`Conversation`":** la clase conversación es la encargada de hacer la traducción o "parse" de un diálogo en bruto. Esta además se encargade gestionar el flujo de la conversación, es decir, de escoger el siguiente dialogo, seleccionar una respuesta, reiniciar la conversación…Hay un objeto de la clase "`Conversation`" por cada conversación que hay en el juego. La forma en la que se estructura un diálogo es mediante una lista, donde por cada elemento hay un objeto de la clase "`Dialog`". Por ello es importante que en el documento tanto el orden de los diálogos como el índice del diálogo siguiente esté correcto.
- **Clase "`Dialog`"**: Esta es la unidad de una conversación, es decir el elemento más pequeño. Esta contiene información sobre si el diálogo es texto plano o una pregunta, así como el texto en sí mismo y el índice del siguiente diálogo.
- **Clase "`ConversationManager`"**: Esta es la clase que gestiona de verdad la conversación, es decir la clase que escribe en el cuadro de texto el diálogo. Esta clase es única debido a que solo puede haber una conversación a la vez.
  Esta clase además tiene la función de mostrar de forma dinámica del texto, así como de atender a los Inputs del jugador. Algunas de las funcionalidades que ofrece esta clase son:
  - Escribir el texto de forma progresiva.
  - Completar el texto para poder saltarlo si así el jugador lo desea.
  - Elegir respuesta
  - Avanzar en la conversación
  - Mostrar el avatar del personaje correspondiente.

  Esta clase para mostrar el texto por pantalla se comunica con la clase "`GuiManager`" para mostrar la información por pantalla.

- **Clase "`Conversation Trigger`":** esta clase se encarga de comenzar las conversaciones, así como de hacer ciertos ajustes los cuales comentaré a continuación para que la conversación sea más inmersiva.
  Funciones de la clase "`Conversation Trigger`":
  - Mostrar a Alice si así lo de desea el diseñador.
  - Cambiar el foco de la cámara según la necesidad, por ejemplo, poniéndose entre Alice yEDD0.

- Permite decidir si el diálogo se debe borrar o no tras su uso, es decir, por ejemplo, hay conversaciones de la historia que solo deben mostrarse una vez, mientras que hay diálogos informativos que deben poder leerse tantas veces como el jugador desee.
- Si se debe establecer un nuevo objetivo tras la conversación y el objetivo en sí.

5.5.4 La GUI

La GUI[13] es un componente esencial que proporciona al jugador una representación visual clara y comprensible del estado del juego y del personaje. A través de la GUI, el jugador puede acceder a información relevante, como la salud del personaje, la marcha actual, la pantalla de pausa, el modo de disparo …

En este apartado se hablará de cómo se gestiona la GUI y de cada elemento por separado.

5.5.4.1 Clase GUI Manager

Esta clase es la encargada de ofrecer al resto de clases los mecanismos necesarios para mostrar información en la GUI. Esto es posible ya que esta clase agrupa los diferentes componentes que intervienen en la GUI, como pueden ser textos, imágenes, sliders… y ofrecen mecanismos para modificarlos.

5.5.4.2 Pausa y objetivos

Como se pue de ver en Ilustración 137, hay varios elementos importantes a los que se debe prestar atención.

- **Sliders de volumen**: estos permiten configurar el volumen del juego en cualquier momento. También se guardan en la memoria de la partida para que de esta forma la siguiente vez que se juegue, se mantengan los valores.
- **Botón de "reconnect"**: este botón es exclusivo del uso del cubo. Este permite en caso desconexión del cubo volver a conectarlo de forma automática.
- **Botón de "exit"**: Este permite volver al menú principal.
- **Botón de "continue"**: este botón permite continuar la partida.
- **Objetivos**: a la izquierde de la pantalla, se puede ver el objetivo actual. Este siempre está disponible desde el menú de pausa.

Ilustración 137: menú de pausa

[13] Interfaz gráfica de usuario

5.5.4.3 Interfaz de juego con teclado

Como se pue de ver en Ilustración 138, hay varios elementos importantes a los que se debe prestar atención.

- **Marchas**: estas se ven abajo a la izquierda. La forma en la que se representan es mediante cinco imágenes en forma de un degradado, donde se destaca la marcha actual.
- **Tutorial**: este indica todos los posibles inputs en ese momento.
- **Modo de disparo**: este elemento aparece solo en el modo de disparo e indica que tipo de disparo está activo en ese momento.

Ilustración 138: interfaz de juego

5.5.4.4 Interfaz de juego con cubo

Como se pue de ver en Ilustración 139, hay varios elementos importantes a los que se debe prestar atención además de los anteriormente mencionados en 5.5.4.3 Interfaz de juego con teclado. Estos elementos son exclusivos del uso del cubo.

- **Indicador de pausa**: Este está en la parte superior izquierda y sirve para indicar al jugador los movimientos que debe seguir para entrar en la pausa. También se puede ver una barra la cual se irá llenando a medida que se realice la secuencia.
- **Indicador de último movimiento**: este se puede ver en la parte superior derecha, en ella se muestra el último giro realizado con el cubo.
- **Indicador de referencia de centros**: este se puede ver en la parte central a la derecha, esta muestra dos imágenes con dos colores, el color de la parte inferior corresponde con el color del centro que el usuario debería tener en su cubo en frente y el color de arriba es el color del dentro de la parte superior que debería tener el cubo. Esta información es relevante ya que los giros dobles suponen que se pueda perder la referencia en ciertos momentos.

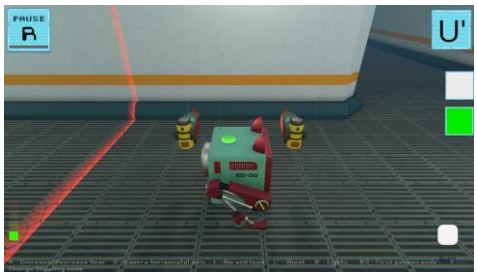


Ilustración 139: interfaz juego con cubo

5.5.5 Puerta automática

Esta es una mecánica muy simple pero que merece la pena mencionar ya que esta supone un pilar en otras mecánicas implementadas.

5.5.5.1 Descripción general

Esta puerta está diseñada para que el jugador no tenga ninguna clase de problema atravesando zonas, pero a la vez como mecanismo de bloqueo de zonas. Esta está implementada mediante la clase "`DoorManager`", la cual permite a las diferentes mecánicas abrir o cerrar puertas de forma sencilla.

En la forma básica de esta puerta, esta se abra y se cierra de forma automática cuando el jugador interactúa entorno a ella. Esta por así decirlo se asemeja a las puertas automáticas de un supermercado. En Ilustración 140 se puede ver el diferente uso que se ha hecho de la puerta a lo largo del juego.

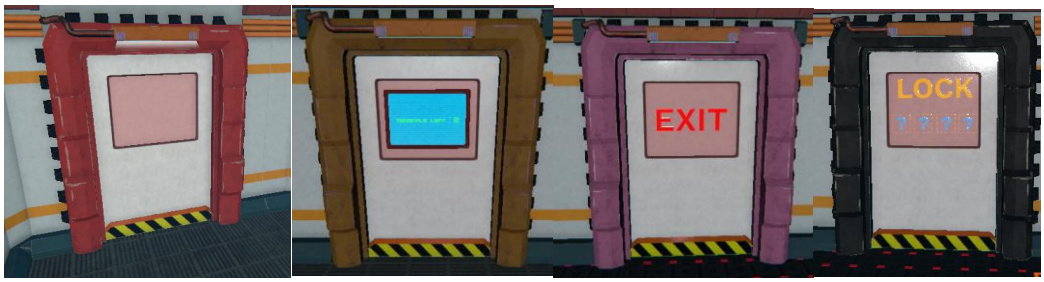


Ilustración 140: diferentes usos de las puertas

5.5.5.2 Elementos a destacar

Además de las funciones básicas de una puerta, también implementa otras funcionalidades las cuales el diseñador puede elegir si activar o no.

- **Iniciar un sonido ambiente**: es decir que, al atravesar una puerta, se puede elegir que se reproduzca in sonido, ya sea para crear tensión u otra emoción en el jugador.
- **Cambiar el estado de la niebla**: esto permite cambiar la atmosfera del juego para viertas zonas de forma dinámica, por ejemplo, para que cuando el jugador suba las escaleras en el Nexo y esté frente a la última puerta se cree un ambiente rojizo.
- **Introducir o quitar un objetivo**: de esta forma al atravesar una puerta con mecánicas a parte como puede ser la mecánica de los terminales, se quite el objetivo de dicha mecánica.

5.5.6 Campo de tiro

Esta es una mecánica para que el jugador pueda practicar su habilidad en el modo de disparo.

5.5.6.1 Descripción general

El campo de tiro, el cual se puede ver en Ilustración 141, es una mecánica en la cual es jugador puede practicar el modo de disparo desde un espacio seguro.

En el campo de tiro se pueden apreciar varios elementos, en primer lugar, a la izquierda del campo de tiro se puede apreciar el marcador de la mejor partida jugada este se mantiene entre partidas gracias a que se gurda en la memoria de la partida. Más adelante se puede ver una barrera transparente, la cual nos bloquea el paso, pero permite que las balas la atraviesen. Finalmente, hay

una diana con el texto “Start” que, tras dispararla, comenzará la sesión del campo de disparo. En esta sesión, habrá varias tandas o rondas de dianas las cuales algunas estarán fijas y otras seguirán un patrón. Al acabar se nos mostrará la puntuación de la partida.

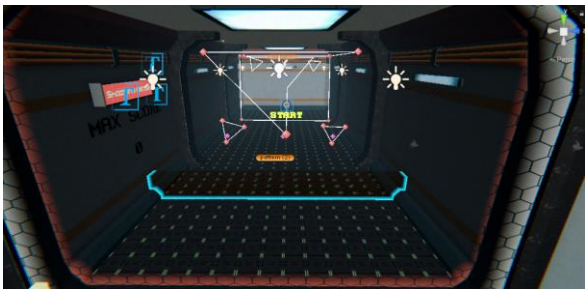


Ilustración 141: campo de tiro

5.5.6.2 Implementación

El campo de tiro está formado por varios elementos que hacen posible esta mecánica, los cuales describiré a continuación:

- **Clase “`ShootingRangeManager`”**: esta clase es la encargada de gestionar todos aquellos elementos de esta mecánica, así como el comienzo de la partida, el comienzo de las fases, la puntuación, el reinicio de la mecánica…
- **Clase “`Target`”**: esta por así decirlo, es la unidad de esta mecánica, esta representa una diana e implementa formas de aparecer, desaparecer o reacción a un disparo.
- **Clase “`Pattern`”**: esta está formada por una serie de “Targets” o dianas, las cuales siguen un recorrido. Este recorrido viene marcado por una serie de nodos que se deben introducir manualmente. Esta tiene métodos tanto para comenzar como para parar y desaparecer.
- **Clase “`Phase`”**: esta clase representa un conjunto de patrones los cuales se deben iniciar cuando la clase “`shootingRangeManager`” lo requiera y que se deben poder finalizar del mismo modo.

5.5.7 Interruptor

Esta mecánica es una mecánica que hace uso de las puertas del apartado 5.5.5 Puerta automática, en esta se debe hacer uso del interruptor de forma que de esta forma el jugador pueda abrir y cerrar puertas.

5.5.7.1 Descripción general

La forma en la que funciona esta mecánica es que hay dos tipos de elementos, los interruptores y las puertas de interruptor.

El interruptor, el cual se puede ver en Ilustración 142, puede tener dos posibles estados, o morado o verde, este se activa usando un disparo del arma, ya que para poder disfrutar de esta mecánica es imprescindible que el jugador esté en el modo de disparos.

Al interactuar con el interruptor, este cambiará de color, y además abrirá todas aquellas puertas del color del interruptor, por lo tanto, si el interruptor está de color morado, se abrirán las puertas moradas y se cerrarán las puertas verdes, y pasaría lo contrario en caso de que estuviera en color verde.

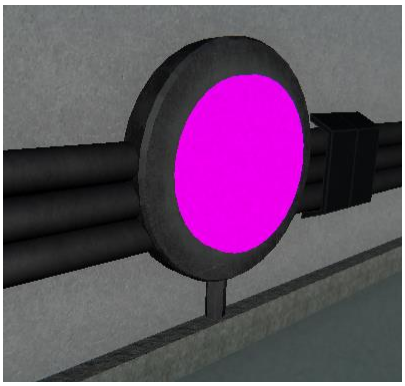


Ilustración 142: interruptor

5.5.8 Carrera

La mecánica de la carrera es la que hace uso de la cuarta marcha del personaje, la cual está bloqueada durante todo el juego menos en esta mecánica. El objetivo de esta mecánica es terminar una carrera antes de que se acabe el tiempo.

5.5.8.1 Descripción general

Como se acaba de mencionar, el objetivo de esta carrera es llegar a la meta antes de que se acabe el tiempo. Para ello, se contará con la mecánica de la cuarta marcha la cual permite al personaje ir mucho más rápido.

La carrera da comienzo tras cruzar el punto de salida, el cual está representada mediante una pared transparente con un filtro de color verde. Tras cruzar esta pared, el personaje, deberá intentar llegar hasta el "trigger" de meta, este está representado con un patrón de cuadros blancos y negros.

En caso de llegar a la meta, antes de que acabe el tiempo, se terminaría la carrera, se mostraría un mensaje con el texto "CONGRATULATIONS" y posteriormente desaparecería todo lo relacionado con la mecánica. En caso de no llegar a tiempo, se reiniciaría la carrera y EDD0 sería transportado hasta su inicio.

Esta mecánica cuenta con varios elementos esenciales:

- **Trigger de comienzo de partida**: este le indica a la mecánica que la carrera ha comenzado y, por lo tanto, el tiempo empieza a correr y se muestra por pantalla.
- **Trigger de salida**: en caso de no querer realizar la carrera, se da la opción de volver por donde se vino de forma que se atraviesa este trigger y la mecánica de carrera queda en reposo.
- **Trigger de modo carrera de EDD0**: Este está colocado junto a la posición de salida y permite que el jugador pueda usar la marcha 4.
- **Trigger de meta**: este trigger da fin a la partida si se llega antes de que acabe el tiempo.

Ilustración 143: mecánica de carrera

5.6 Diseño y Comportamiento de enemigos

5.6.1 Introducción

En este apartado, se abordará detalladamente el diseño, comportamiento e implementación de los enemigos que se encuentran en el juego. Los enemigos desempeñan un papel fundamental en la experiencia de juego, ya que representan los obstáculos y desafíos que el jugador debe enfrentar para avanzar en la historia.

En cuanto al diseño de los enemigos, se ha puesto especial atención en su apariencia visual y características distintivas para que sean reconocibles y se ajusten al estilo del juego. Cada enemigo ha sido cuidadosamente diseñado con sus propios atributos, habilidades y debilidades únicas, lo que crea una variedad de desafíos para el jugador.

5.6.2 Zona de enemigos

5.6.2.1 Algoritmo de Dijkstra

Antes de entrar de lleno en la explicación de los enemigos es fundamental entender cómo funciona el algoritmo de Dijkstra ya que una parte fundamental de los enemigos se basas en la decisión de rutas a través del algoritmo de Dijkstra.

El algoritmo de Dijkstra es una potente herramienta en la teoría de grafos que permite resolver el problema del camino más corto en un grafo ponderado. Su objetivo principal es encontrar la ruta óptima desde un nodo de origen a todos los demás nodos en el grafo.

El funcionamiento del algoritmo de Dijkstra es bastante sencillo [17] pero eficiente. Comienza marcando el nodo de origen con una distancia inicial de 0 y el resto de los nodos con una distancia infinita. A medida que avanza, selecciona el nodo con la distancia más corta y lo visita, actualizando las distancias de los nodos adyacentes en función de las aristas ponderadas.

La implementación de del algoritmo de Dijkstra que se ha hecho en la clase "`EnemyPath`" y se pue de ver en Ilustración 144. Esta se ha implementado con el objetivo de crear rutas aleatorias para los enemigos.

```
1 usage  Carlos Chamizo Cano
private void Dijkstra(int nodeIni)
{
    List<int> selectedNodes = new List<int>();
    selectedNodes.Add(nodeIni);
    List<int> pendNodes = InitPendNodes(nodeIni);
    while (pendNodes.Count > 0)
    {
        int candidate = SelectCandidate(pendNodes.ToArray(), nodeIni);
        selectedNodes.Add(candidate);
        pendNodes.Remove(candidate);
        UpdateDistances(candidate, nodeIni);
    }
}
```

Ilustración 144: implementación del algoritmo de Dijkstra

5.6.2.2 Descripción de la mecánica.

La mecánica de la zona de enemigos presenta un desafío. En una zona específica del mapa, un grupo de enemigos disparadores están patrullando. La ruta que siguen los enemigos está determinada por un grafo, como se puede ver en Ilustración 145, que representa las diversas opciones y caminos disponibles para ellos.

El objetivo del jugador en esta mecánica es eliminar a todos los enemigos presentes en la zona. Al derrotar a los enemigos, se abrirá el acceso a una nueva área del juego.

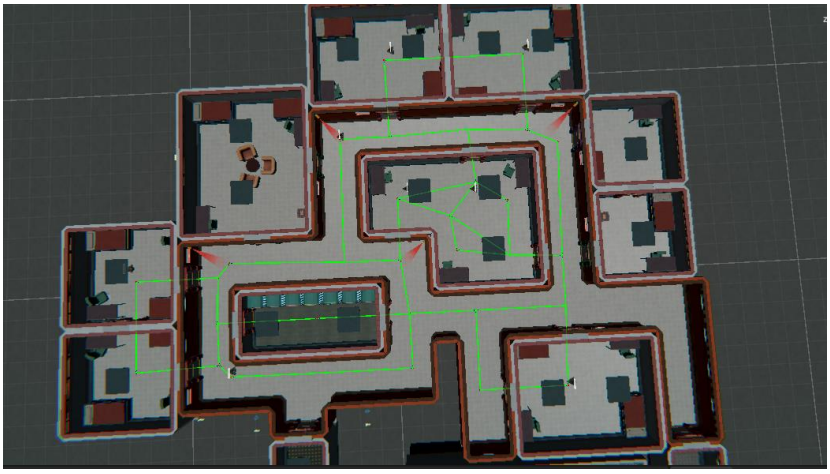


Ilustración 145: grafo de zona de enemigos

5.6.2.3 Clase Enemy Path

La clase "`EnemyPath`" implementa un mecanismo para que los enemigos puedan patrullar por la zona sin necesidad de crear patrones de movimiento predefinidos. La idea es que los enemigos vayan por los diferentes lugares del mapa escogiendo de forma aleatoria el destino.

Para esto se ha tenido que crear un grafo manualmente. Este está compuesto, como se puede ver en Ilustración 146, por un lado, por nodos los cuales son en realidad puntos en el espacio representados por un "Transform" y por otro lado por aristas los cuales se crean en variables "`Vector2`" indicando el índice del nodo origen en el valor de la "X" y el índice del nodo destino en la "Y". De esta forma se tienen tanto los nodos como la arista, y el coste de cada arista se puede calcular como la distancia entre nodos. Por lo que se tienen todos los ingredientes para implementar el algoritmo de Dijkstra.

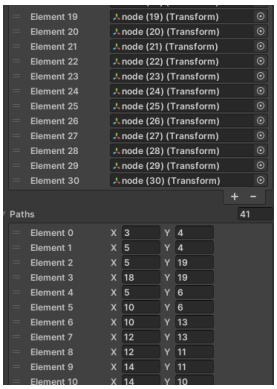


Ilustración 146: nodos y aristas del grafo

Nada más comenzar la ejecución del juego, lo primero que se realiza es el algoritmo de Dijkstra para de esta forma sacar todos los caminos mínimos. Respecto a mi implementación de Dijkstra, no entraré en de talles, pero me gustaría destacar que el vector de distancias está formado por variables del tipo "`vector2`", done en el valor "X" se guarda la distancia mínima al destino y en la variable "Y" se almacena el índice del nodo que se debe escoger para llegar hasta el destino, tal y como se ve en Ilustración 147.

```
1 usage  Carlos Chamizo Cano
private void UpdateDistances(int candidate, int nodeIni)
{
    float distToCandidate = distances[nodeIni][candidate].x;
    for (int i = 0; i < nodes.Count; i++)
    {
        float distActual = distances[nodeIni][i].x;
        float distFromCandidate = distToCandidate + distances[candidate][i].x;
        distances[nodeIni][i].x = Mathf.Min(a: distActual, b: distFromCandidate);
        distances[nodeIni][i].y =
            distActual < distFromCandidate ? distances[nodeIni][i].y : distances[nodeIni][candidate].y;
    }
}
```

Ilustración 147: método Update distances

Una vez creado realizado el algoritmo de Dijkstra sobre el grafo, se necesitan herramientas para que los enemigos puedan hacer uso de ello. Las herramientas implementadas las enumeraré a continuación.

- **Nodo más cercano**: esta función permite inicializar las posiciones de los enemigos, de forma que empiecen desde un punto de partida.
- **Nodo aleatorio**: esta función permite escoger un destino objetivo para la ruta de patrullado de los enemigos.
- **Nodo siguiente**: esta función permite al enemigo seguir con su ruta cuando ha llegado a al siguiente nodo.
- **Nodo más cercano al jugador**: esta función permite establecer como objetivo para los enemigos, ir hasta el nodo que esté más cerca del jugador.

5.6.2.4 Clase EnemyShooterZone

Esta es la clase que controla la mecánica, en ella se tiene la referencia a todos los enemigos que transitan la zona. Principalmente esta clase debe controlar cuantos enemigos quedan vivos y debe ser capaz de reiniciar la mecánica en sí misma.

En cuanto al control del número de enemigos quedan vivos, se debe a que cuando todos los enemigos sean derrotados, debe desbloquear el acceso a otra zona del mapa por lo que es muy importante que lleve un buen control de los enemigos.

La zona de enemigos debe ser capaz de reiniciar la mecánica debido a que, si el jugador muere, lo lógico es que deba enfrentarse al reto otra vez. La zona de enemigos, lo que hace es invocar el método "`ResetEnemy`" el cual le devuelve toda su vida y reinicia su comportamiento.

5.6.3 Clase enemigo y máquina de estados

La clase enemigo es una clase abstracta la cual implementa todos aquellos mecanismos y variables necesarias para implementar los diferentes enemigos. Los métodos que como mínimo deben tener son las siguientes:

- **"`RecieveDamage`"**: es decir implementar las diferentes acciones que debe realizar el enemigo cuando recibe daño.
- **"`Hide`"**: es decir que debe tener un mecanismo para que cuando sea necesario, el enemigo desaparezca, por ejemplo, ante una boss fight en la que se quiere que el enemigo aparezca después de leer un diálogo.
- **"`Spawn`":** es decir que el enemigo debe poder establecer unos valores y un estado inicial.
- **"`GetEnemyDead`":** es decir, una forma de comunicar si el enemigo ha sido derrotado.
- **"`ResetEnemy`":** es decir que se debe de poder implementar una forma de reiniciar el estado del enemigo, así como sus valores.

Por otro lado, para el comportamiento del enemigo disparador y de las cámaras de seguridad, es necesario un mecanismo que gestione el comportamiento diseñado para los enemigos. Para ello el mecanismo implementado es una máquina de estados.

Esta máquina de estados es se implementa de forma bruta dentro de las clases encargadas del comportamiento de los enemigos. El primer elemento necesario es una lista de estados en los que puede estar el enemigo. Posteriormente, es indispensable tener una variable que indique cual es el estado actual. Debido a que como se ve en Ilustración 148, este determina el estado que se va a ejecutar.

```
{
    switch (currentState)
    {
        case States.Patrol:
            Patrol();
            break;
        case States.Searching:
            Searching();
            break;
        case States.DestroyDistraction:
            DestroyDistraction();
            break;
        case States.Alert:
            Alert();
            break;
    }
```

Ilustración 148: implementación máquina de estados

Finalmente quedan las transiciones, estas se hacen desde los propios estados tras ejecutar el código del comportamiento de dicho estado como se ve en Ilustración 149.

```
Frequently called | 1 usage | Carlos Chamizo Cano
private void Searching()
{
    SlowDown();
    lookAtPlayer.LookPlayerSlow();

    if (_timer.Elapsed.TotalSeconds > timerCooldown)
    {
        _timer.Restart();
        ChangeState(States.Patrol, index: 0);
    }

    if (CheckIsInCone())
    {
        ChangeState(States.Alert, index: 2);
    }

    if (_targetDistraction.GetBeingUsed() && InSight(_targetDistraction.transform.position, tag: "Distraction"))
    {
        ChangeState(States.DestroyDistraction, index: 2);
        _timer.Restart();
    }
}
```

Ilustración 149: ejemplo de estado

5.6.4 Enemigos disparadores

5.6.4.1 Descripción general

Los enemigos disparadores, los cuales se ven en Ilustración 150 son una clase de enemigos los cuales patrullan por una zona del mapa. Estos están constantemente buscando al personaje. Si el personaje entra en su rango de detección, estos entrarán en un estado de búsqueda y si el personaje dispara o es avistado directamente por el enemigo, este intentará acabar con el personaje. Además, se puede distraer a los enemigos mediante el uso de una distracción.

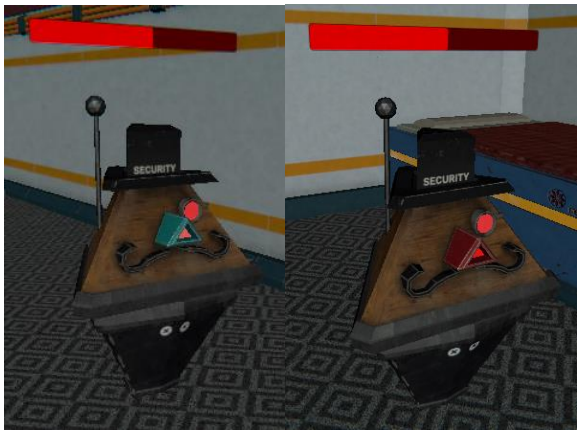


Ilustración 150: enemigos disparadores

Como se ve en Ilustración 150, hay dos tipos de enemigos disparadores en función de las acciones realizadas en el estado de alerta. El primero es el que se ve a la izquierda, el cual una vez entrado en estado de alerta se acercará al jugador y lo seguirá mientras esté en este estado. Por otro lado, está el enemigo disparador de la derecha, estos se quedarán quietos una vez entren en estado de aleta.

5.6.4.2 Descripción de los estados

A continuación, se detallarán las acciones que se realizan en cada uno de los diferentes estados de los enemigos disparadore:

- **Patrullando**: en este estado, el enemigo se encuentra en tránsito hacia su nodo destino. Este siempre apuntará con la cabeza en la dirección al siguiente nodo al que viajará.
- **Buscando**: en este estado, el enemigo reducirá su velocidad para poder observar bien, i lentamente mirará al jugador.
- **Destruir distracción**: en este estado, el enemigo parará de patrullar y se dispondrá a disparar a la distracción del personaje para de esta forma eliminarla.
- **Alerta**: en este estado, el enemigo o bien se acercará al personaje o bien se mantendrá quieto, es entonces cuando le disparará. Mientras permanezca en este estado.

5.6.4.3 Percepciones

A continuación, se detallarán todas aquellas percepciones que el enemigo es capaz de detectar, así como su implementación.

- **En rango de búsqueda**: teniendo la posición del jugador, se puede calcular su distancia al enemigo, por lo tanto, el rango de búsqueda es cuando la distancia entre el jugador y el enemigo esté por debajo de cierto umbral.
- **A la vista**: Conociendo la posición del personaje, se puede lanzar un rayo desde el enemigo hasta el personaje y ver si hay algún impacto entre medias, en caso de no haber, estaría a la vista.
- **Dentro del cono de visión:** el enemigo, lanza un "`spherecastAll`" el cual permite escanear una amplia zona enfrente suya. Este "`ShereCast`" indicará si el personaje está dentro de su rango
- **Tiempo de búsqueda**: este mediante un "`StopWatch`" puede medir el tiempo que lleva buscando. A este tiempo se le puede concretar un límite de búsqueda para que pase a realizar otra acción.
- **Distracción usada**: Debido a que el enemigo tiene acceso a la distracción del personaje, este puede comprobar mediante el método "`GetBeingUsed`" para comprobar si la distracción se ha usado.
- **Distracción destruida**: Esta percepción viene del propio enemigo cuando dispara a la distracción.
- **Jugador dispara:** debido a que los enemigos están suscritos al jugador mediante el patrón "`Observer`" explicado en 5.4.2.3 Acciones del personaje, este recibirá una notificación de que el personaje ha disparado.

5.6.4.4 Esquema

En Ilustración 151 se puede ver el diagrama de la máquina de estados de este enemigo.

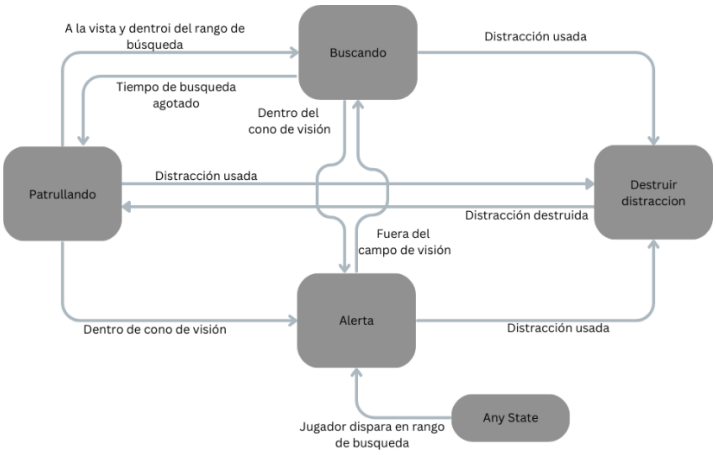


Ilustración 151: Diagrama de la máquina de estados del enemigo disparador

5.6.4.5 Funcionalidades extra

Además del comportamiento el enemigo, estos cuentan con ciertos mecanismos los cuales detallaré a continuación los cuales están destinados a dar retroalimentación al jugador para que este conozca el resultado de sus acciones.

- **Barra de vida**: como se ve en Ilustración 150 los enemigos tienen una barra de vida sobre sus cabezas, la cual indica la vida restante del enemigo.
- **Shader de impacto**: Este es un shader que se aplica sobre los enemigos cada vez que son impactados por una bala, este, aplica una textura totalmente plana y emisiva al enemigo, la cual dura unos milisegundos.
- **Sprites de estados**: estos sprites indican al jugador si el enemigo se encuentra patrullando, es decir, sin Sprite, en estado de búsqueda, es decir, con el Sprite de la interrogación o si está en estado de alerta, es decir con la exclamación.

Ilustración 152: sprites de estados

- **Sonidos**: Todas las acciones de los enemigos van acompañadas de retroalimentación auditiva.
- **Outline**: Durante unos instantes tras el jugador haber disparado al enemigo, este le aparecerá un delineado rojo a su alrededor. Que permitirá que el jugador sepa dónde está el enemigo, incluso si está detrás de una pared.

5.6.5 Cámaras enemigas

5.6.5.1 Descripción general

Las cámaras enemigas, se encargan de vigilar ciertas zonas. Estas se encuentran patrullando, poniendo su foco en una serie de puntos concretos que siguen un patrón. Esta está a la espera de que el personaje entre en el cono de visión de la cámara. Sin embargo, es muy sensible a los cambios, por lo tanto, si detecta que el personaje está corriendo o que tiene las luces encendidas y además

de estos dos factores, está en el campo de visión de la cámara (no necesariamente en el cono de detección) esta intentará destruir al personaje.

Ilustración 153: cámara enemiga

5.6.5.2 Descripción de los estados

A continuación, se detallarán las acciones que se realizan en cada uno de los diferentes estados de las cámaras enemigas:

- **Buscando**: en este estado, la cámara seguirá lentamente al jugador.
- **Patrullando**: en este estado, permanecerá patrullando cuatro puntos en el escenario cambiando el punto de foco entre ellos.
- **Destruir distracción**: en este estado, la cámara se olvida del jugador y se dispone a disparar a la distracción hasta que esta sea destruida.
- **Alerta**: En este estado, el personaje ha sido detectado y la cámara se dispone a destruirlo.

5.6.5.3 Descripción de las percepciones

A continuación, se detallarán todas aquellas percepciones que el enemigo es capaz de detectar, así como su implementación.

- **A la vista**: Conociendo la posición del personaje, se puede lanzar un rayo desde el enemigo hasta el personaje y ver si hay algún impacto entre medias, en caso de no haber, estaría a la vista.
- **Dentro del cono de visión:** el enemigo, lanza un "`spherecastAll`" el cual permite escanear una amplia zona enfrente suya. Este "`ShereCast`" indicará si el personaje está dentro de su rango
- **Tiempo de búsqueda**: este mediante un "`StopWatch`" puede medir el tiempo que lleva buscando. A este tiempo se le puede concretar un límite de búsqueda para que pase a realizar otra acción.
- **Distracción usada**: Debido a que el enemigo tiene acceso a la distracción del personaje, este puede comprobar mediante el método "`GetBeingUsed`" para comprobar si la distracción se ha usado.
- **Distracción destruida**: Esta percepción viene del propio enemigo cuando dispara a la distracción.

- **Luces encendidas**: el enemigo, debido a que tiene acceso a la clase "`PlayerValues`", puede acceder a la variable luces con la cual detectará si las luces están encendidas o no.
- **Corriendo**: es decir que el enemigo, puede acceder a la variable" gear" de "`PlayerValues`" y si esta es mayor que la marcha 2, entonces detectará que está corriendo.

5.6.5.4 Descripción de las percepciones

En Ilustración 154 se puede ver el diagrama de la máquina de estados de la cámara enemiga.

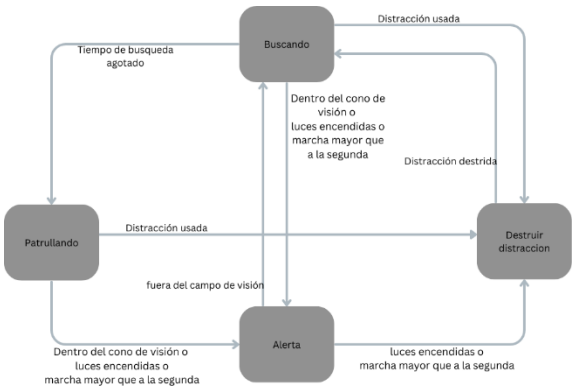


Ilustración 154: diagrama de máquina de estados de cámara enemiga

5.6.5.5 Funcionalidad extra

Además del comportamiento el enemigo, estos cuentan con ciertos mecanismos los cuales detallaré a continuación los cuales están destinados a dar retroalimentación al jugador para que este conozca el resultado de sus acciones.

- **Sprites de estados**: estos sprites indican al jugador si el enemigo se encuentra patrullando, es decir, sin Sprite, en estado de búsqueda, es decir, con el Sprite de la interrogación o si está en estado de alerta, es decir con la exclamación, estos se pueden ver en Ilustración 152.
- **Sonidos**: Todas las acciones de los enemigos van acompañadas de retroalimentación auditiva.
- **Cono y luz de detección**: el enemigo consta de un cono al cual se le ha aplicado un shader para simular una luz volumétrica saliente de la cámara la cual indica el volumen del cono de detección. Además, cuanta con una luz que ilumina la zona hacia donde la cámara apunta. Ambos elementos cambian de color en función del estado, siendo azul el color del estado de patrullando, amarillo el estado de búsqueda y rojo el estado de alerta y de destrucción de la distracción.

5.6.6 Jefes

Además de enemigos hay dos jefes únicos, estos no siguen un comportamiento complejo, sino más bien una secuencia de patrones. Estos enemigos se caracterizan por tener una barra de vida en la interfaz gráfica del usuario.

5.6.6.1 Avispas

Descripción general

Este es el jefe de la cuarta zona, estas son unas avispar las cuales se puede ver en Ilustración 155 habrá que derrotar. Estas se moverán por el escenario disparando activamente al jugador hasta matarlo. Este combate comienza tras leer un diálogo de comienzo. El objetivo de la pelea será el de derrotar a todas las avispas, tras esto, se abrirá una nueva zona en el mapa.

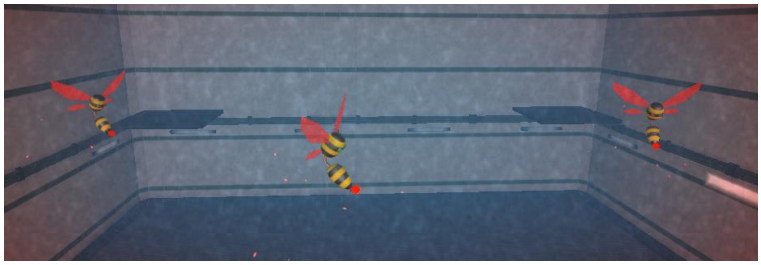


Ilustración 155: jefe avispas

Comportamiento

Las avispas tienen un solo estado, el cual es el estado de alerta. Mientras esté en este estado, realizará dos acciones. La primera es moverse por el mapa, para ello usa el mismo mecanismo que el explicado en la 5.6.2.3 Clase Enemy Path. Las avispas se moverán por los nodos que se pueden ver en Ilustración 156, mientras lo hacen dispararán al personaje de forma constante en intervalos de un tiempo arbitrario. Sin embargo, tienen dos tipos de disparos, el disparo simple y el disparo triple. Este último tiene menos probabilidad de ocurrir.

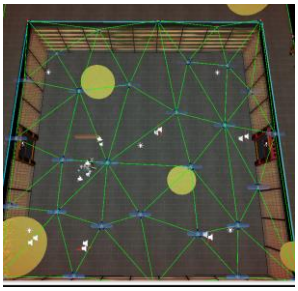


Ilustración 156: grafo de movimiento de avispas

5.6.6.2 Boss final

Descripción general

Este es el jefe final del juego, el cual se puede ver en Ilustración 157, la mecánica de este jefe se basa en un “BulletHell”, es decir que el enemigo dispara una gran cantidad de proyectiles los cuales habrá que esquivar además de intentar disparar al enemigo. Este combate consta de dos fases las cuales se explicarán en el siguiente punto.

Ilustración 157: jefe final del juego

Comportamiento

El combate empieza con un diálogo, tras leerlo, dará comienzo la batalla. Esta batalla consta de dos fases. En la primera fase, el jefe realizará tres posibles ataques, los cuales son:

- **Ataque en espiral**: este ataque dispara una serie de balas a un ritmo constante, mientras los disparadores giran, esto crea un efecto de espiral como se ve en Ilustración 158.

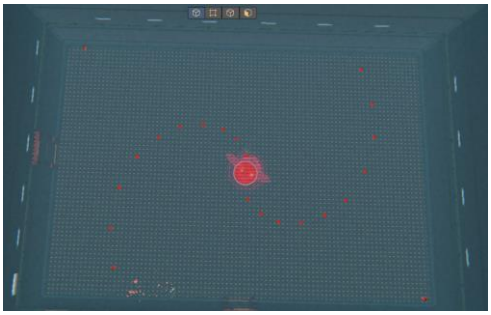


Ilustración 158: ataque en espiral

- **Ataque "Bullet Hell"**: en este ataque se dispararán balas en todas las direcciones, dejando muy poco espacio de movimiento al jugador, como se ve en Ilustración 159.

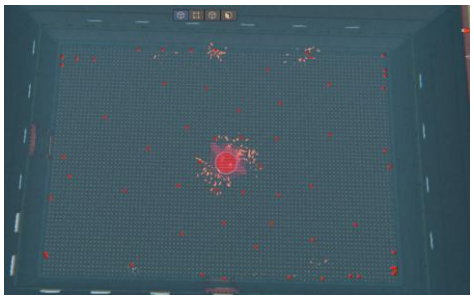


Ilustración 159: ataque Bullet Hell

- Ataque "Arrow": en este ataque se disparan una serie de Balas en forma de flecha hacia el jugador, como se pue de ver en Ilustración 160.

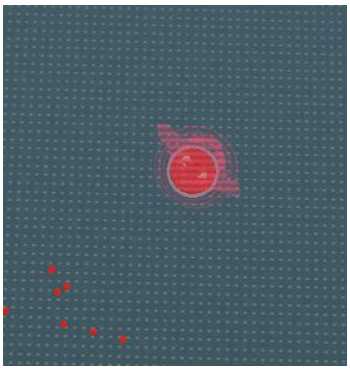


Ilustración 160: ataque arrow

Posteriormente, tras bajarle la mitad de la vida a el jefe, comenzará la segunda fase. En esta segunda fase en primer lugar, se mostrará un diálogo y posteriormente se dará continuación al combate. Los ataques serán los mismos que en la primera fase, con la diferencia de que estos se realizan cada menos tiempo, elevando la dificultad. Además, entre ataque y ataque, está la posibilidad de recibir un ataque vertical por medio de unos Láser como se puede ver en Ilustración 161.

Estos láseres seguirán al jugador de forma lenta, indicando con unas marcas en el suelo que se va a disparar. Tras unos segundos se dispararán los láseres los cuales se mantendrán encendidos durante un corto periodo de tiempo y finalmente se apagarán.

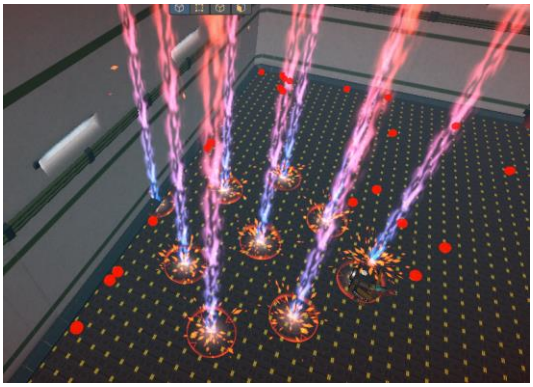


Ilustración 161 ataque de laser.

Finalmente, tras derrotar al jefe se abrirá una puerta que permitirá acceder al final del juego. Cabe destacar que, si se muere en este combate, el jefe se reiniciará teniendo que empezar este otra vez de nuevo.

5.7 Optimización del juego

Debido a que el tamaño del escenario era demasiado grande, esto causaba que a la hora de jugarlo el rendimiento ser viera muy afectado debido a que se necesitaba cargar todo el mapa, así como los comportamientos de todos los elementos de cada zona. Es por ello por lo que se idearon una serie de técnicas que ayudaron a que el juego aumentara considerablemente su rendimiento y haciendo que este se pudiera jugar en cualquier tipo de ordenado o portátil actual. A continuación, explicaré los problemas y las soluciones planteadas para cada problema.

5.7.1 Borrado de zonas

Debido a la gran extensión del escenario, era inviable tener todo el escenario cargado durante la partida, ya que, a parte de la cantidad de recursos necesarios para renderizar la escena, se utilizan recursos de los comportamientos presentes en cada una de las zonas. Es por ello, que se tomó la decisión de dividir el escenario en 5 zonas. De forma que solo hubiera activa la zona en la que se encuentra el jugador.

Esto es posible ya que el jugador una vez que entra en una zona, es imposible que vuelva a la zona de nexo y escoja otra ruta, primero debe terminar la zona.

Esta técnica es funcional, pero sin embargo cuantas más zonas hay desbloqueada, funciona peor en la selección de zona ya que deben estar todas disponibles al poder el jugador querer acceder a cualquiera de ellas. Sin embargo, al seleccionar acceder a cualquiera de las zonas el problema se soluciona ya que se pueden deshabilitar el resto de las zonas.

Para reducir en cierta medida el problema anteriormente planteado, se ha hecho uso del "oclussion culling" de UNITY, el cual se explicará a continuación.

5.7.2 Oclusión de la cámara

Esta es una herramienta que ofrece UNITY por defecto pero que debe ser activada. En ella, lo que se hace es solo renderizar los objetos que se ven en el "frustrum" de la cámara, como se puede vere en Ilustración 162, de esta forma, reduce en gran medida la carga de renderizado.

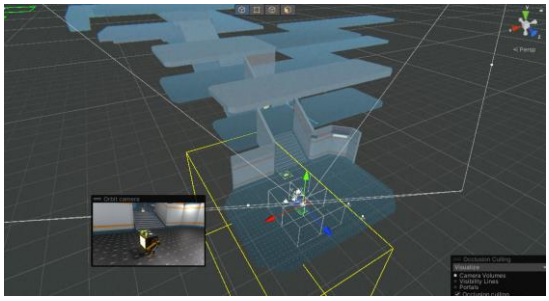


Ilustración 162: oclussion culling

5.7.3 Iluminación

La iluminación del escenario también era un problema, ya que lo lógico hubiera sido generar unos "lightmaps" de forma que no se tuviera que computar la luz de forma dinámica. Sin embargo, esto debido a la gran extensión del escenario era inviable ya que mi ordenador no tiene los recursos suficientes para hacerlo, llegando a bloquearse completamente en ocasiones durante la generación de "lightmaps".

Es por ello por lo que se tuvo que buscar una solución ingeniosa a este problema. La solución que se me ocurrió fue hacer un sistema de encendido de luces automático, por lo que tan solo se encenderían como se ve en Ilustración 163, las luces más próximas al jugador. Esto además encaja con la estética del juego ya que, al ser un refugio vacío, no tendría mucho sentido tener todas las luces encendidas. Gracias a esto el rendimiento no se vio afectado en absoluto.

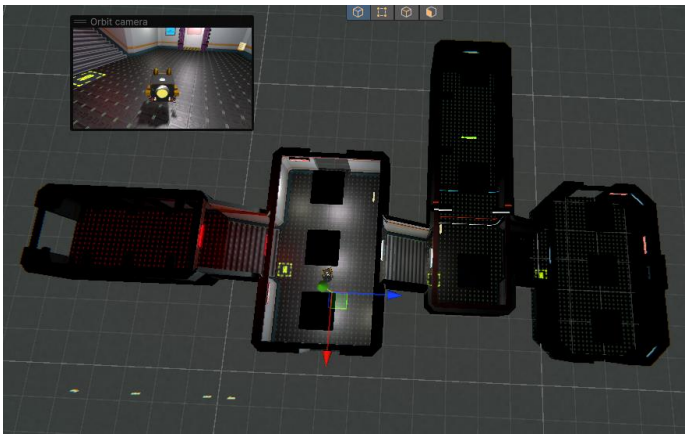


Ilustración 163: luces en nexo

5.7.4 Pool de balas

Finalmente, el último problema de recursos que era necesario solucionar era el de las balas usadas tanto por el personaje como por los enemigos, ya que estas se acumulaban gastando muchos recursos innecesarios.

Para ello se ideó un mecanismo de disparo único para todo el juego, es decir que todos los elementos, ya sea, los enemigos, el personaje, el jefe final…, utilizan el mismo tipo de objeto. Este es el disparador y se compone de tres elementos:

- **El disparador**: este se encarga de pedirle al pool que le entregue una bala para ser disparada. La bala, tras impactar en cualquier objeto de la escena, perderá su uso y volverá al pool de balas disponibles para volver a ser usadas.
- **Un pool de balas**: esta es una estructura que implementa un conjunto donde almacenar las balas que ya han sido utilizadas. De esta forma cuando el disparador necesite una bala, se la pedirá al pool. En caso de que no haya balas disponibles, se generarán nuevas balas las cuales posteriormente volverán al pool.
- **Una plantilla de bala**: esto se debe a que no todos los enemigos o personaje, usan el mismo tipo de bala, ya que esta puede variar tanto en tamaño como en color.

Es importante conocer que cada disparador tiene su pool, por lo que para evitar conflictos no se mezclan las balas.

Gracias a esta implementación el problema de la cantidad de balas se solucionó mejorando el rendimiento del juego.

5.7.5 Assemblies

Los Assemblies, son referencias a directorios donde se encuentran los scripts de proyecto. Estos se encargan de almacenar los datos que han sido modificados en dicho directorio para que, de esta forma a la hora de compilar los scripts, solo se copilan aquellos que están dentro del directorio de dicho Assemby [18].

Por defecto, UNITY implementa por defecto un Assemby general, por lo que cada vez que se realizan cambios en un script se debe recompilar todos los scripts. Esto en un proyecto como este es inviable ya que, al tener alrededor de unos 150 scripts en total, cada vez que se modificara un

script el tiempo de compilado sería exagerado. Es por ello por lo que se ha organizado cuidadosamente la carpeta de scripts para de esta forma optimizar el tiempo de recopilación.

5.8 Sonido

El sonido juega un papel fundamental en el juego, ya que desempeña un papel crucial en la creación de una experiencia inmersiva y enriquecedora para los jugadores. El diseño de sonido cuidadosamente elaborado contribuye a la atmósfera general del juego, sumergiendo a los jugadores en un mundo sonoro que complementa y realza la acción visual. Los efectos de sonido, como explosiones, disparos, pasos y ambientaciones, agregan profundidad y realismo a las interacciones del jugador con el entorno del juego. Además, la música de fondo juega un papel importante en la creación de la atmósfera del juego.

A continuación, se explicarán los aspectos más relevantes respecto a la implementación del sonido.

- **Audio mixer y Audio Sources**: el elemento más destacado respecto al sonido es el uso de un audio mixer. Este es muy importante ya que gracias a él se pueden gestionar de forma general el sonido de cada aspecto del juego, pudiendo dividir estos en varios grupos y gestionar su subconjunto como una unidad.
    - **Máster**: permite gestionar el sonido general del juego.
    - **Música ambiente**: permite gestionar el sonido de todos los Audio Source marcados por el sub controlador música ambiente.
    - **Efectos de sonido**: permite gestionar el sonido de todos los Audio Source marcados por el sub controlador efectos de sonido.
    - **Sonidos de la interfaz**: permite gestionar el sonido de todos los Audio Source marcados por el sub controlador sonidos de interfaz.

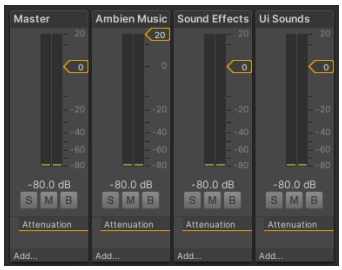


Ilustración 164: Audio mixer

El audio Mixer es muy útil sobre todo para los ajustes donde se puede gestionar el volumen de cada subgrupo del audio mixer como se pue de ver en Ilustración 62.

También un aspecto relacionado a destacar es que para que desde el menú de pausa se pusiera a cambiar el volumen de forma lineal, se aplicó la fórmula que se ve en sonido donde hace el logaritmo en base 10 del valor normalizado objetivo y se le multiplica por 20.

```
audioMixer.SetFloat( name: "MasterVolume",  value: Mathf.Log10(sliderValue) * 20);
```

Ilustración 165: fórmula para linealizar el cambio del volumen del sonido

- **Zonas de reverberación**: las zonas de reverberación permiten realizar cambios en el ambiente. Estos aplican un filtro de reverberación, el cual tiene varios modos predefinidos en función del espacio, y contribuyen a mejorar la experiencia de juego.
- **Sonido 3D**: algunos elementos del juego, como puede ser las cápsulas, tienen incorporado sonido 3D, esto permite mayor inmersión ya que este variará en función de la posición del jugador respecto a la fuente de sonido.

5.9 Organización del proyecto en UNITY

EL proyecto en primer lugar, se tomó la decisión de alojarlo en GitHub [19] de forma que, aunque estuviera trabajando en mi ordenador principal no dependiera de este, sino que pudiera tener la misma información el portátil por ejemplo para trabajar desde otras ubicaciones.

Por otro lado, en cuanto a la organización del proyecto de UNITY, este como se puede ver en Ilustración 166. De esta manera, el tamaño del proyecto no fue un problema, ya que todos los elementos estaban cuidadosamente organizados.

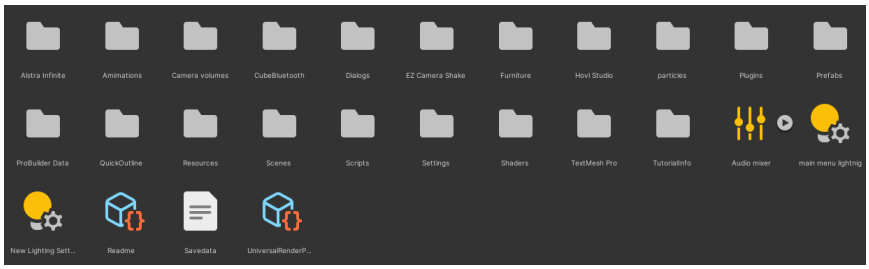


Ilustración 166: organización del proyecto de UNITY

5.10 Arte y conceptos

En la realización de este proyecto se ha realizado también un amplio trabajo de arte, donde se ha texturizado y modelado la mayor parte de los elementos del juego. A continuación, mostraré de forma breve el trabajo realizado en este campo.

Todas las referencias artísticas usadas se pueden ver en la tabla de Pinterest creada para este proyecto [20].

5.10.1 Concepto, modelado y animación del personaje

5.10.1.1 Concepto

En cuanto al concepto del personaje, se empezó con una hoja de siluetas para explorar las diferentes posibilidades. Posteriormente y con una idea más clara, me puse a hacer un boceto del personaje que me gustara hasta que llegué a una idea que me convenció. Por ello, se desarrolló esta idea y se hizo un beauty, el turn around y finalmente las pruebas de color. También se realizó un boceto de la idea del poster del juego, pero, sin embargo, este no se llegó a terminar por problemas de tiempo. Todas las ilustraciones digitales están hechas con el programa Procreate.

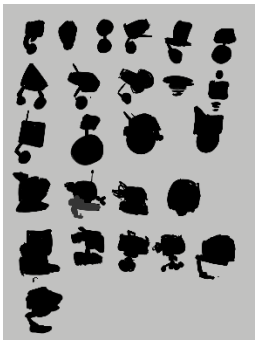


Ilustración 167: exploración de siluetas

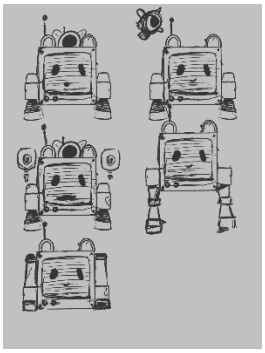


Ilustración 168: boceto de EDD0

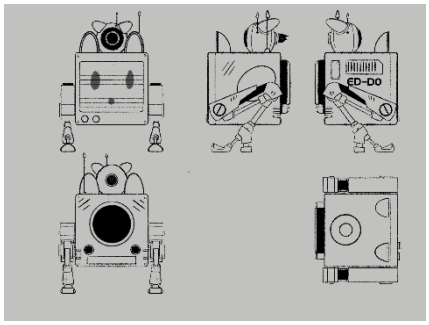


Ilustración 169: turn around de EDD0

Ilustración 170: Beauty de EDD0

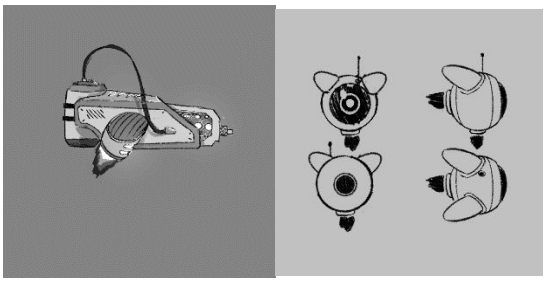


Ilustración 171: concept del arma y del dron de EDD0

Ilustración 172: pruebas a color de EDD0

Ilustración 173: boceto de portada del juego

Tabla 13: concept EDD0

5.10.1.2 Modelado, animaciones y texturizado

Una vez se tenía una idea muy clara de la forma del personaje, se procedió a modelarlo, sin embargo, antes de ponerse con esta larga tarea, hice un trabajo de investigación acerca del animal o robot bípedo más parecido a EDD0, para comprobar si era posible que pudiera moverse. La respuesta era si, como se puede ver en la animación sobre la vista lateral que realicé [21], ya que vi una referencia muy buena la cual era la de los AT-ST de "Star Wars" [22].

Usando el turn around del personaje creado en la etapa de concept, se modeló el personaje en Blender. Posteriormente, se tuvieron ajustar las UVs del modelo ya que por defecto venían muy mal optimizadas. Una vez se hubo completado esto, tuve que ponerme a aprender a utilizar SubstancePpainter, la herramienta que se utilizó para texturizar ya que esta es la que mejores resultados da, pero debido a que no es nada intuitiva de usar estuve varios días atascado. En SubstancePainter, hice una textura genérica y como los colores del robot son 2 para generar nuevas variantes del modelo tan solo tenía que cambiar esos 2 colores.

Una vez texturizado, volviendo a Blender, investigue la mejor forma de riggear un modelo mecánico como mi robot. El rigging y el skining es diferente a lo que se haría tradicionalmente. En vez de asignar huesos a zonas del modelo, cada hueso corresponde con un bloque solido del modelo, de esta forma el modelo no se deforma a excepción de la antena ya que sí era necesario que se moviera con deformación. Además, tanto la antena como las piernas tienen animación inversa para facilitar el proceso de hacer animaciones, es decir que, controlando el último hueso, automáticamente se generan el resto de las rotaciones. Además, para facilitar el proceso de animaciones, puse restricciones de rotación y traslación a las piezas de tal forma que no pudiera equivocarme y romper accidentalmente el modelo y las propias animaciones.

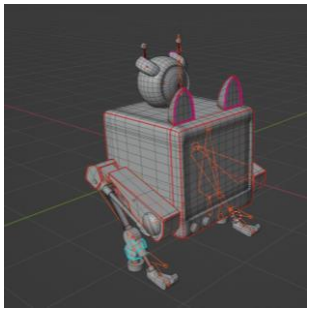


Ilustración 174 Modelado y rigging de EDD0:

Ilustración 175: texturizado de EDD0

Ilustración 176: render en substance painter

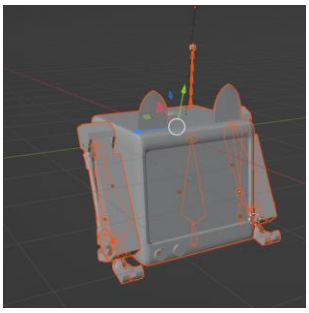


Ilustración 177: Animación de sentado

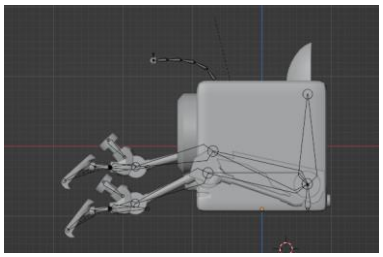


Ilustración 178: animación de marcha 4

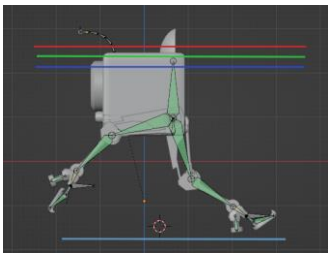


Ilustración 179: animación de correr

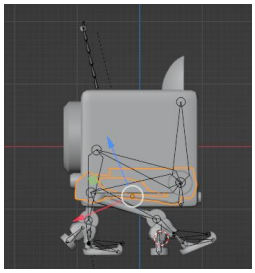


Ilustración 180: animación de caminado

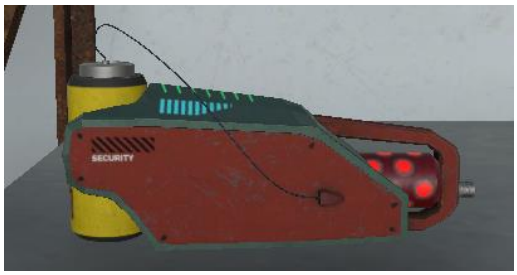


Ilustración 181: arma de EDD0

Tabla 14: concept modelado, texturizado y animaciones de EDD0

5.10.2 Concepto, modelado y animación de los personajes no jugables

Además de EDD0, se han diseñado y modelado otros personajes, sin embargo, estos no se han texturizado ya que la idea inicial era usar shaders con efecto de hologramas para ellos.

5.10.3 Alice

Ilustración 182: boceto Alice

Ilustración 183: ilustración Alice

Ilustración 184: ilustración Alice Malvada

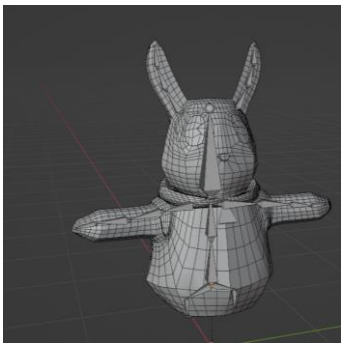


Ilustración 185: Modelado y rigging de Alice

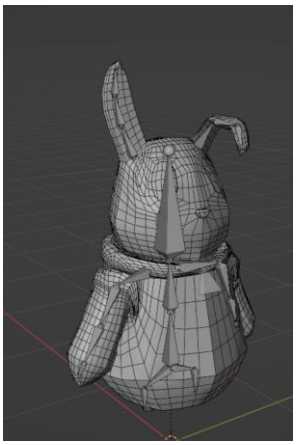


Ilustración 186: animación de Iddle de Alice

ç

Ilustración 187:resultado final Alice

Tabla 15: concept y modelado de Alice

5.10.4 Concepto y modelado del escenario

Finalmente, el diseño y modelado del escenario también supuso un verdadero reto debido a las múltiples opciones. Debido a que había que decidir si hacer un escenario en un solo modelo o y texturizarlo, o hacer diferentes módulos del escenario texturizados y montarlos en UNITY a coste de mayor número de polígonos. Finalmente se decidió tomar el camino de la segunda opción debido a que hacer un escenario en un solo modelo era una idea que, por lo menos desde mi inexperiencia veía inviable.

Lo primero que se hizo fue un mapa general con todas las zonas, para de esta forma sacar el diseño de los niveles, así como las diferentes ambientaciones que se iban a necesitar. Posteriormente me puse a modelar y texturizar los módulos del escenario. A continuación, se mostrarán algunos de los modelos creados para el juego:

Ilustración 188: boceto de mapa

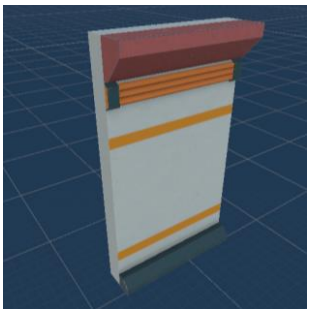


Ilustración 189: pared genérica

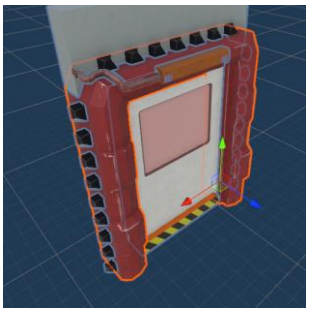


Ilustración 190: puerta

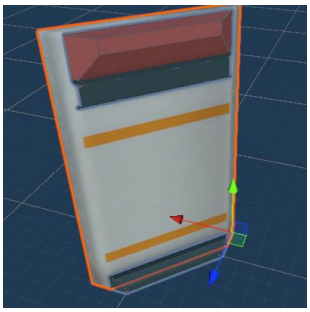


Ilustración 191: esquina larga

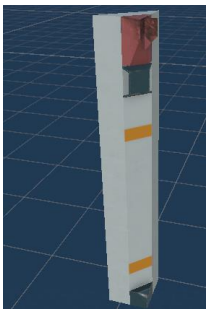


Ilustración 192: esquina interior

Ilustración 193: esquina exterior

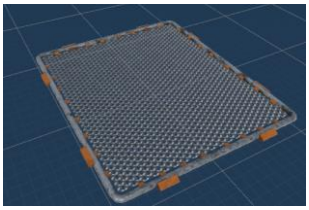


Ilustración 194: suelo de malla de hierro

Ilustración 195: estantería con cajas

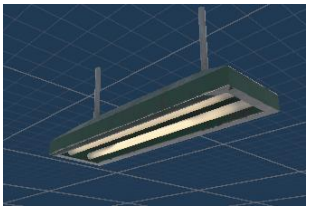


Ilustración 196: lampara larga

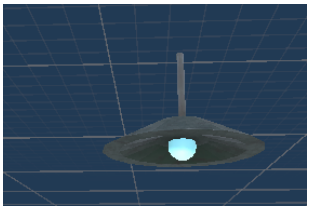


Ilustración 197: lámpara redonda

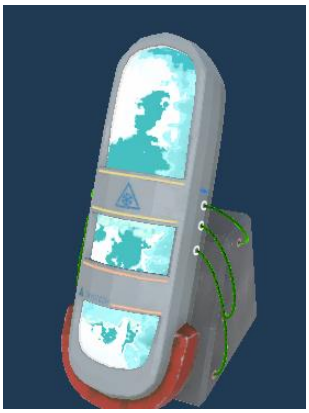


Ilustración 198: capsula humano

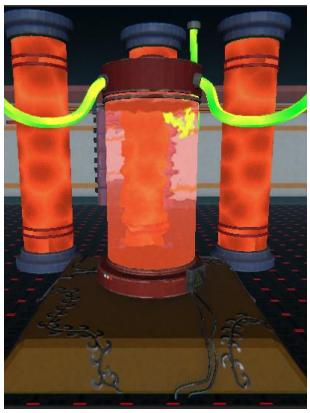


Ilustración 199: capsula miniboss

Tabla 16: modelos del escenario

6. Evaluación

Tras el desarrollo del juego, se ofreció la versión BETA de este a diferentes usuarios. Estas tras jugar mostraron puntos ciertos puntos en común. Los cuales se resumirán a continuación.

Respecto a la experiencia de juego general fue buena, sin embargo, el ritmo de juego les parecía lento además de que los minijuegos son en ocasiones repetitivos o demasiado simples.

También le hubiera gustado que se hubieran explotado ciertas mecánicas del juego las cuales creen que tienen mucho potencial pero que no se les da demasiado importancia. Por otro lado, también echan en falta jefes más originales con los que usar estas mecánicas.

Respecto al arte y el diseño de los niveles, así como del escenario son opiniones positivas, haciendo que los niveles no sean nada repetitivos y con una dificultad bien balanceada.

En resumen, las sensaciones han sido bastantes positivas, aunque caben destacar varios problemas los cuales son, el ritmo de juego o el falte de variedad en la aplicación de las mecánicas.

7. Conclusiones

Durante los últimos 6 meses se han dedicado muchas horas a este juego, suponiendo un gran parte de mi tiempo, en paralelo junto con el otro trabajo de fin de grado de ingeniería de computadores. En este trabajo he aprendido mucho sobre los diferentes aspectos del desarrollo de un videojuego. He profundizado en primer lugar en UNITY, del cual considero que tenía una base sólida, pero me ha servido para ampliar mis conocimientos, tocando aspectos que nunca había tocado como puede ser la creación de Shaders o la optimización de un juego.

Por otro lado, se han tratado otras disciplinas como el arte o el sonido, respecto al arte ya tenía conocimientos, sin embargo, me ha servido para profundizar en tareas que tiempo atrás había evitado como puede ser el texturizado. Respecto al sonido, he aprendido a usar técnicas que implementa UNITY y de las cuales conocía pero que jamás me había usado en un escenario real.

Por otro lado, se ha tratado el diseño de niveles, el cual, a pesar de considerarme una persona inexperta en este tema, creo que ha quedado un resultado más que decente en cuanto dificultad y sensaciones que el escenario transmite.

En cuanto al Gameplay y sensación del personaje principal es la parte de la que más orgulloso me siento ya que el resultado está muy pulido y realmente se siente cómodo, el cual era uno de los objetivos principales de este trabajo de fin de grado.

A pesar de que el desarrollo ha sido largo, ha merecido la pena solo por el resultado ya que ver todo el trabajo realizado en un solo ejecutable me hace sentirme muy orgulloso de mi mismo, así como del trabajo realizado.

7.1 Retrospectiva de objetivos

- **Hacer uso de la Interfaz de Cubo de Rubik desarrollado en el trabajo de fin de grado de Ingeniería de Computadores de forma que se cumplan los siguientes requisitos:**
    - **El juego debe poder jugarse en su totalidad con el cubo de Rubik:** tal y como se puede ver en 5.3.2.2 Inputs del Cubo el juego se puede jugar en su totalidad con el cubo de Rubik, por lo que este objetivo se cumple.
    - **El juego debe explicar al usuario de qué forma usar el cubo de Rubik para juagar:** como se puede ver en 5. 1 Documento de diseño del juego, en el apartado de controles, se muestra siempre por pantalla los posibles inputs del juego, así como un tutorial de los cubos de Rubik en el menú principal por lo que este objetivo se cumple.
- **Creación de un videojuego desde cero pasando por todas las fases de desarrollo. Este videojuego debe cumplir los siguientes requisitos:**
    - **El juego debe poder ser jugable para una persona que no tenga un Smart Cube.:** como se explica en el punto 5.3.2.3 Inputs del teclado, el juego se puede jugar en su totalidad mediante ratón y teclado sin ningún tipo de e problema por lo que este objetivo se cumple.
    - **El juego debe tener un trabajo de Concept y modelado:** como se ve en el apartado 5.10 Arte y conceptos hay un gran trabajo de Arte y concept presentes en el juego. Por lo que este objetivo también quedaría superado.
    - **El control del personaje principal debe ser cómodo y satisfactorio.:** como se desarrolla en el apartado 5.4.2 Implementación del personaje jugable la mayor parte del esfuerzo se ha dedicado a que el personaje se sienta fluido y cómodo. Por lo que este objetivo queda superado.
    - **El juego debe tener un sistema de guardo de juego:** en el punto 5.2 Sistema de guardado y carga de partida se explica la implementación seguida a la hora de crear un sistema de guardado, de forma que se pueda seguir el progreso del juego en diferentes instantes de tiempo. Por lo que este objetivo queda superado.
    - **Debe ser fluido en cualquier ordenador moderno:** como se explica en el apartado 5.7 Optimización del juego se han implementado mecanismos para la optimización y mejora del rendimiento este objetivo queda completado, sin embargo, se tiene una pequeña muestra sobre este dato al haber sido probado en pocos ordenadores.
    - **Debe tener enemigos gestionados por un comportamiento:** esto como se puede ver en 5.6 Diseño y Comportamiento de enemigos se cumple debido a la

implementación de varios enemigos de los cuales dos de ellos son gobernados por una máquina de estados.

- **Debe tener una historia, así como un sistema de diálogos.:** esto se cumple como se puede ver en 5. 1 Documento de diseño del juego en el apartado de narrativa, como en el apartado 5.5.3.2. Diálogos donde se muestra la implementación de los diálogos. Por lo que este objetivo está superado.
- **El proyecto debe estar bien organizado:** como se puede ver en 5.7.5 Assemblies y en el apartado 5.9 Organización del proyecto en UNITY se ha organizado el proyecto de forma cautelosa, por lo que este objetivo queda superado.

En definitiva, se han superado todos los objetivos establecidos al principio del proyecto e incluso se ha realizado más trabajo del que en un principio abarcaba este proyecto.

7.2 Trabajos Futuros

De cara al futuro, tengo pensado hacer varias cosas. En primer lugar, me gustaría realizar más pruebas sobre el juego con voluntarios, de forma que se puedan detectar y arreglar problemas que no se hubieran tenido en cuenta.

En segundo lugar, me gustaría ampliar el juego haciendo uso de los elementos ya existentes en el juego, para crear un modo "arcade" del tipo de juegos como el "Call of Duty" modo zombies donde el objetivo sea sobrevivir lo máximo que se pueda, Para esto tendría que hacer un diseño menos lineal que el que se ha diseñado para el juego.

Por otro lado, de cara a un futuro cercano, me gustaría preparar el estreno del juego, haciendo una pequeña campaña de marketing, la cual incluiría un teaser, un poster del juego, así como una carátula y un tráiler de este.

Finalmente me gustaría documentarlo todo en una página web para de esta forma añadirlo a mi porfolio de proyectos.

7.3 Conclusiones personales

Mi experiencia desarrollando este juego ha sido agridulce. Por un lado, ha sido muy buena debido a que es un proyecto el cual he escogido yo y es algo que me apasiona muchísimo y algo que siempre había querido hacer es un juego totalmente mío donde pudiera plasmar mis ideas tal y como las tengo en la cabeza, pero sin embargo, ha sido una carga de trabajo brutal debido a que el desarrollo de un videojuego de estas dimensiones se ha hecho difícil en ciertas ocasiones, por lo que si tuviera que hacer un proyecto así en un futuro, sin ninguna duda buscaría un equipo.

Referencias

[1] Itch.io, «003,» [En línea]. Available: https://danieljimenezmorales.itch.io/003. [Último acceso: 14 julio 2023].

[2] Real academia de la lengua española, «Videojuego,» [En línea]. Available: https://dle.rae.es/videojuego.

[3] Ahoy, «A Brief History of Video Games,» [En línea]. Available: https://www.youtube.com/watch?v=GoyGlyrYb9c.

[4] C. Trilnick, «IDIS, Tennis for two,» [En línea]. Available: https://proyectoidis.org/tennis-for-two/.

[5] Pong game.org, «Pong game,» [En línea]. Available: https://www.ponggame.org/.

[6] J. A. Román, «CLASIFICACIÓN de GÉNEROS de VIDEOJUEGOS,» 4 junio 2020. [En línea]. Available: https://www.gameoverla.com/clasificacion-de-generos-de-videojuegos.html.

[7] P. Luban, «Designing and Integrating Puzzles in Action-Adventure Games,» [En línea]. Available: https://www.gamedeveloper.com/design/designing-and-integrating-puzzles-in-action-adventure-games.

[8] Wikipedia, «Cubo de Rubik,» [En línea]. Available: https://es.wikipedia.org/wiki/Cubo_de_Rubik.

[9] M. L. Michelone, «A 43 años del nacimiento del Cubo de Rubik,» [En línea]. Available: https://www.unocero.com/ciencia/43-anos-del-nacimiento-del-cubo-rubik/#:~:text=Fue%20David%20Singmaster%20quien%20invent%C3%B3,se%20denomin%C3%B3%20%E2%80%9Cnotaci%C3%B3n%20Singmaster%E2%80%9D.. [Último acceso: 12 julio 2023].

[10] Wikipedia , « Espacios liminales,» [En línea]. Available: https://es.wikipedia.org/wiki/Espacio_liminal.

[11] A. Capdevila, «EL PATRÓN OBSERVADOR EN C#,» [En línea]. Available: https://albertcapdevila.net/patron-observador-csharp/. [Último acceso: 13 julio 2023].

[12] C. C. Cano, «Itch.io,» [En línea]. Available: https://notsooart.itch.io/penguin-age. [Último acceso: 13 julio 2023].

[13] J. Flick, «Orbit Camera,» 26 enero 2020. [En línea]. Available: https://catlikecoding.com/unity/tutorials/movement/orbit-camera/. [Último acceso: 13 julio 2023].

[14] Hovl Studio, «RPG VFX PACK,» [En línea]. Available: https://assetstore.unity.com/packages/vfx/particles/spells/rpg-vfx-bundle-133704. [Último acceso: 13 julio 2023].

[15] Wikipedia, «Ley de Hook,» [En línea]. Available: https://es.wikipedia.org/wiki/Ley_de_elasticidad_de_Hooke. [Último acceso: 13 julio 2023].

[16] Coolmathgames, «World's Hardest Game 2,» [En línea]. Available: https://www.coolmathgames.com/0-worlds-hardest-game-2. [Último acceso: 13 julio 2023].

[17] E. C. Navone, «Algoritmo de la ruta más corta de Dijkstra - Introducción gráfica y detallada,» [En línea]. Available: https://www.freecodecamp.org/espanol/news/algoritmo-de-la-ruta-mas-corta-de-dijkstra-introduccion-grafica/. [Último acceso: 14 julio 2023].

[18] Unity, «Assembly definitions,» [En línea]. Available: https://docs.unity3d.com/Manual/ScriptCompilationAssemblyDefinitionFiles.html. [Último acceso: 14 julio 2023].

[19] C. C. Cano, «Repositorio del juego,» [En línea]. Available: https://github.com/chamix0/Cute-Robot-Game-TFG. [Último acceso: 14 julio 2023].

[20] C. C. Cano, «cute robot game tfg,» [En línea]. Available: https://pin.it/2HfMTB4. [Último acceso: 14 julio 2023].

[21] C. C. Cano, «prueba de animación de ED D0,» [En línea]. Available: https://youtu.be/L84sA5XZcAI. [Último acceso: 14 julio 2023].

[22] B. B. [En línea]. Available: https://www.youtube.com/watch?v=r9E9l0pRME8. [Último acceso: 14 julio 2023].

[23] Facultat d'Informàtica de Barcelona, «Historia de los videojuegos.,» [En línea]. Available: https://www.fib.upc.edu/retro-informatica/credits.html.

[24] J. Velasco, «Historia de la tecnología: Spacewar!, el videojuego que nació en el MIT,» hipertextual, [En línea]. Available: https://hipertextual.com/2011/07/spacewar-el-videojuego-que-nacio-en-el-mit.

[25] videojuegos fandom, «Videojuego de disparos en primera persona,» [En línea]. Available: https://videojuegos.fandom.com/es/wiki/Videojuego_de_disparos_en_primera_persona#:~:text=Disparos%20en%20primera%20persona%2C%20en,primer%20FPS%20fue%20Maze%20War..

[26] Gaming history, «Arcade Video game published 44 years ago by Atari, Inc.,» [En línea]. Available: https://www.arcade-history.com/?n=asteroids&page=detail&id=126.

[27] C. Kohler, «The Console Wars,» 2008.