Proyecto Fin de Grado

Ingeniería de las Tecnologías Industriales

Diseño y Desarrollo de un Videojuego de Plataformas basado en Unity y Herramientas Open Source

Autor: Diego Gómez Colliga

Tutor: Juan Antonio Sánchez Segura

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Grado
Ingeniería de las tecnologías industriales

Diseño y Desarrollo de un Videojuego de Plataformas basado en Unity y Herramientas Open Source

Autor:

Diego Gómez Colliga

Tutor:

Juan Antonio Sánchez Segura

Dpto. de Ingeniería Electrónica

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de Carrera: Diseño y desarrollo de un videojuego de plataformas basado en Unity y herramientas open Source

Autor: Diego Gomez Colliga

Tutor: Juan Antonio Sánchez Segura

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mis amigos y mi familia
A mis maestros

Agradecimientos

Este proyecto no hubiera sido posible sin el apoyo de todos mis amigos que se han implicado probando este proyecto. Sobre todo, agradecer a mi amigo David Varo Gomez quien sin obtener ninguna remuneración a cambio y de forma desinteresada, ha realizado toda la música del videojuego, la cual conlleva un gran trabajo. Sin su apoyo este proyecto nunca se hubiera podido llevar a cabo.

También quería agradecer a mis padres y mi hermana por haberme animado a terminar este proyecto y sobre todo esta carrera tan complicada, gracias a su apoyo nunca me vine abajo y pude afrontar cada problema con más fuerza.

Resumen

.

Este proyecto trata de plasmar todo lo aprendido sobre informática en la carrera, al área de los videojuegos. El trabajo en sí es un reto de dificultad de programación, pero también de creatividad.

El objetivo de este proyecto es crear un videojuego desde cero para una empresa ficticia, con todo lo que conlleva esto.

Comenzaré situando al lector en el contexto actual de la industria de los videojuegos y su huella en la humanidad. A continuación, se elegirá que tipo de videojuego se quiere realizar y por qué motivo. Una vez elegido el motor con el que se programará el videojuego se explicará paso a paso como se ha realizado este y para finalizar se hará además un desglose económico del proyecto como si fuese un proyecto real.

Abstract

This project tries to carry out everything I have learned in my degree about programming to the area of video games, this job is a challenge of programming difficulty, but also of creativity.

The objective of this project is to create a video game from the beginning to the end for a non-real company, with all that this entails.

I will Begin by making the reader understand the current context of the video game industry and its footprint on humanity. Next, I will choose what type of video game I want to make, once I have chosen the engine where I will program de video game, I will explain step by step how I had made it, an economic breakdown of the project will also be made as if it were a real project.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

Hay muchas formas de ver los videojuegos y su diversión. Estos pueden ser usados tanto como para ocio como para entrenamiento de diferentes aptitudes, por ejemplo, un simulador de vuelo. En mi caso centraré mi trabajo en los videojuegos de uso recreativo. Los videojuegos empezaron a desarrollarse antes de que existiera el propio Internet o los móviles y ahora en la actualidad la industria de los videojuegos mueve más de 1500 millones de euros solo en España.

Uno de los primeros Videojuegos de gran éxito fue el Space Invaders, creado por el japones Tomohiro Nishikado en 1978. Se hizo tan popular en esa época que el gobierno de su país, tuvo que declarar la escasez de monedas de 100 yenes, ya que en esa época si uno quería jugar solo podía hacerlo yendo a los recreativos, una gran sala repleta de máquinas recreativas donde para jugar una partida se debía introducir una moneda.

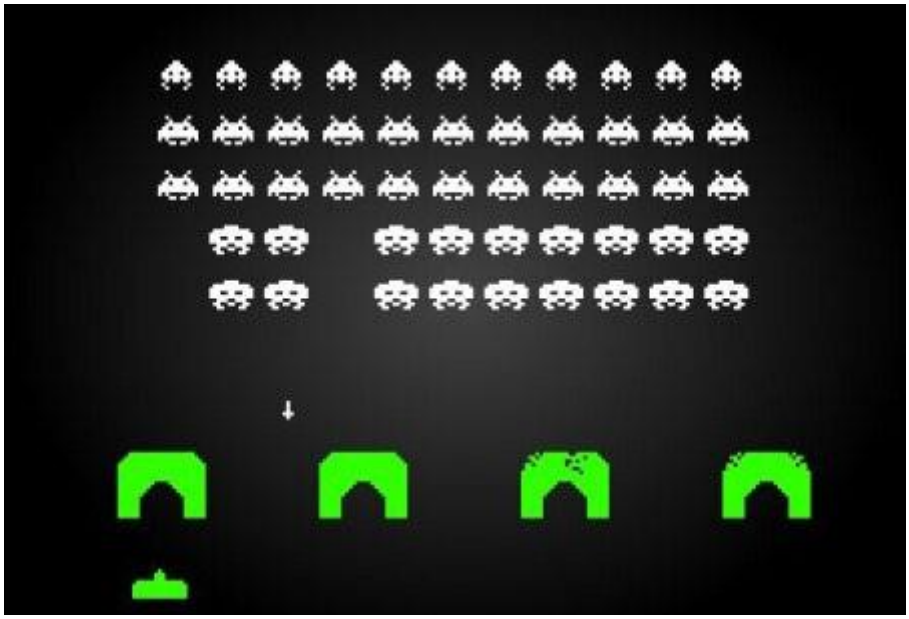


Figura 1-1. Fotograma del juego Space Invaders

La primera videoconsola de uso doméstico fue la ATARI 2600, que salió en 1977 y para la cual uno podía comprar el cartucho de Space Invaders para jugar en tu propia casa. A partir de aquí los videojuegos han ido evolucionando de una manera atónita.

En 1986 salió a la venta Metroid para Gameboy, un videojuego que marcaría un antes y un después en el desarrollo de estos. Makoto Kano el creador de este videojuego quiso innovar y creo un videojuego para Nintendo de plataforma 2D, tipo Mario Bros, pero con un sistema de juego no lineal, es decir, el mismo juego obliga el usuario a explorar todo el mapa para averiguar por donde pasar afín de alcanzar el siguiente nivel. El mundo está conectado de principio a fin, aunque las rutas están limitadas por algún tipo de obstáculo que solo se puede suprimir cuando se consigue la herramienta o la técnica necesaria. Esto cambio la forma de percibir los videojuegos.

Figura 1-2. Fotograma del juego Metroid

En 1986 llegó Castlevania un videojuego de plataforma en 2D para la consola NES muy simple pero que también revolucionó el mercado del videojuego ya que este contenía mecánicas de juegos de rol, donde se podía cambiar de arma, de hechizos, o elegir su propio camino. Cosa que nunca se había realizado en un juego 2D

Figura 1-3. Fotograma del juego Castlevania

En la actualidad la creación de un videojuego conlleva mucho más trabajo, las empresas de desarrollo de videojuegos trabajan durante años con equipos de hasta 2000 trabajadores (como es el caso del ultimo videojuego se Spider-Man) para poder llevar a cabo el trabajo planteado. Esto es así ya que los videojuegos de la actualidad tienen muchos elementos a llevar a cabo como la animación, la música, las mecánicas, la inteligencia artificial, etc.

El desarrollo de videojuegos siempre está en continuo cambio, las grandes empresas hacen videojuegos AAA, son videojuegos para los cuales se utiliza un alto presupuesto y se necesitan muchas personas trabajando en ellos. Aunque también existen pequeñas empresas desarrolladoras de videojuegos con presupuestos más bajos y menos personas implicadas en estas. Estos videojuegos se llaman juegos indie o independientes. Suelen ser juegos de plataforma 2D, aunque no tiene por qué. En mi caso al carecer de presupuesto y de un grupo amplio de desarrolladores, para mi trabajo me centraré en un videojuego indie 2D de plataforma de estilo MetroidVania, es un estilo que une lo más importante de los dos videojuegos explicados anteriormente. Mi objetivo es plantear un videojuego de plataforma donde el jugador irá ganando varias habilidades o recogiendo objetos que le ayudarán en la aventura para seguir avanzando por el mapa. Un ejemplo de este tipo de juego actual es Ori and the Will of the wisps para Xbox one puesto en venta en 2020.

Figura 1-4. Fotograma del juego Ori and the Will of the wisps

En conclusión, para este proyecto desarrollaré un juego indie 2D de plataforma de tipo Metroidvania, ya carezco de un gran presupuesto y al ser un trabajo individual tendré que ocuparme personalmente de cada apartado del desarrollo del videojuego, como son la animación, la música, las mecánicas, etc. Para poder llevar a cabo este trabajo utilizaré los conocimientos adquiridos en las asignaturas de Informática y Diseño Asistido por ordenador.

Crazy Robot, el título de este videojuego, nos llevará a un futuro distópico en el cual controlaremos un Robot que adquiere conciencia propia, tendremos que luchar y escoger el camino correcto para poder salir de las instalaciones donde lo construyeron.

Uno de los puntos más importantes de un videojuego es la correcta interacción de este con el usuario por eso tras elegir una serie de usuarios a medida, les someteremos nuestro videojuego para verificar que el juego va avanzando en su desarrollo y comprobar que funciona correctamente.

Este documento seguirá la siguiente estructura. En primer lugar, presentaré una comparativa y análisis de los motores de videojuegos más representativos (capítulo 2). A continuación, ofreceré una definición de las características más importantes del videojuego, como por ejemplo los enemigos y los objetivos del juego (capítulos del 3 al 10). Después de todo el proceso de planificación y desarrollo (capítulos del 11 al 14), terminaré con las funcionalidades y características que han quedado fuera del proyecto (capítulo 15), finalizando con un enlace de descarga al juego y la bibliografía y las fuentes de información consultadas.

2. MOTORES DE VIDEOJUEGOS

Un motor de videojuegos es un conjunto de herramientas que te permite crear tu propio videojuego, dichas herramientas permiten varias acciones como mostrar gráficos por pantalla, reproducir sonido, detectar teclado, manejar las físicas, etc.

Cuando no existían estos motores de videojuegos cada desarrollador tenía que programar su propio código que cumplía la función de motor, aunque este código se guardaba y se utilizaba para futuros juegos que en apariencia eran diferentes pero el código seguía siendo muy parecido, en conclusión, los motores gráficos nos facilitan el trabajo a la hora de desarrollar un videojuego.

Existen muchos motores de videojuegos para mi trabajo analizaré los motores ~~de~~ más importantes que tengo a mi disposición, compararé las ventajas y desventajas de unity, unreal engine y gamemaker studio 2. Tres de los principales motores de videojuegos gratis, aunque estos motores sean gratis, esto no significa que los tengamos que tachar de mediocres ya que muchos juegos indies que han vendido millones de copias han sido creado con algunos de estos motores, por ejemplo, Cuphead, un videojuego para todas las plataformas.

Figura 2-1. Fotograma de Cuphead, Juego Indie creado con Unity, creado solo por dos hermanos que tuvieron que rehipotecar su casa para poder acabar el juego en 2017

2.1. Unity

Unity es un motor que va enfocado principalmente a programadores, podremos trabajar con este motor con C# o JavaScript. De hecho, es un motor bastante completo que sirve para hacer juegos tanto 3D como 2D.

Este motor de videojuegos te permite exportar los videojuegos que se han creado a casi todas las plataformas, Windows, Android, Linux, PlayStation, etc. También es compatible con la mayoría de aplicaciones de diseño 3D y 2D ya que este no contiene características de construcción o modelado, por lo tanto, para construir animaciones y personajes de mi videojuego he decidido utilizar Blendr, aunque también haré uso gratis de la Asset Store de Unity para utilizar modelos y animaciones ya creadas. Siendo Unity un motor muy completo, su interfaz es bastante sencilla de utilizar ya que implementa por ella misma las físicas en 2D y los sprites.

Figura 2-2. Logo De Unity

Actualmente Unity es gratis con todas sus características disponibles, aunque si el juego obtiene más de 100000$ al año tendremos que comprar la versión profesional que cuesta 75$ al mes. Para Crazy Robot tendré en cuenta que no obtendrá más de 100000$ de ingreso por lo tanto la versión gratis me servirá.

Algunos de los juegos más importantes creados por este motor de videojuegos han sido super Mario run, Temple Run, Rust, Pokemon Go.

2.2. Unreal Engine 5

Este motor es uno de los más avanzados y más populares, su facilidad para utilizarlo y su alto rendimiento gráfico como su sistema de partículas o de iluminación hacen de este motor uno de los más atractivos del mercado. De hecho, es el que usa Epic Games, los creadores de Fortnite, por lo tanto, en la actualidad es todavía más popular.

Es un motor originalmente utilizado para los shooter en primera persona. Además, es un motor muy sencillo de utilizar para todo el mundo ya que implementa los blueprint, un sistema para poder llevar a cabo el videojuego sin tener que escribir líneas de código, no obstante, este motor también se puede trabajar escribiendo código en C++ que se complementa con los blueprints.

Como ya he dicho anteriormente está más orientado a shooters, aunque tiene una excelente interfaz para crear juegos en 2D, sin embargo, al ser mucho mas completo y potente esta orientado a equipo de desarrolladores grandes

Figura 2-3. Logo de Unreal Engine

Este motor de videojuegos es totalmente gratis, aunque como unity a partir de una cierta ganancia que en este caso se cifra en 3000$, se deberá pagar un 5% de las ganancias a la empresa.

Los videojuegos más importantes creados con unreal engine son Dragon Ball FighterZ, Gear of War 4, Hellblade.

2.3. Gamemaker studio 2

Este es un motor amistoso de usar, con una interfaz muy simple para todo el mundo y enfocada sobre todo para videojuegos 2D, aunque también se pueden crear juegos simples en 3D.

Este motor permite exportar tus juegos a muchas plataformas como Windows, Mac, Xbox one, etc. Es un motor sencillo ya que no es necesario saber programar para poder usarlo, es muy parecido a unreal engine, al poder arrastrar y soltar las acciones para crear tus juegos, este método se conoce como D&D (Drag and Drop). Aunque también se podría complementar estas acciones escribiendo líneas de código en GML (Game Maker Language) el lenguaje de programación integrado por defecto en el motor. Para añadir efectos visuales, solo basta con codificar en GLSL, de esta manera se podrá ir añadiendo shaders que están basados en la tecnología OpenGL.

Figura 2-4. Logo de GameMaker Studio 2

La licencia gratuita de este programa te permite probar tus propios juegos de forma limitada en el propio motor, para poder exportar tus videojuegos a PC o Mac es necesario pagar 39$ al año, aunque también se puede comprar la versión de desarrollador por 99$. Para poder exportarlos a otra plataforma habrá que pagar por plataforma.

Este motor no es tan conocido como los dos anteriores, pero se está haciendo cada vez más popular a medida que pasa el tiempo ya que se están poniendo de moda de nuevo los videojuegos en 2D. Por lo tanto, los juegos creados con este software no son tan conocidos, algunos de ellos son: Nuclear throne, Blackhole, Downwell.

2.4. Eligiendo motor

A continuación, he estudiado las ventajas y desventajas de cada motor que he expuesto anteriormente. Como la idea de este proyecto es hacer un videojuego indie en 2D he descartado Unreal Engine ya que está orientado a grupos de desarrolladores más grandes y no necesito una calidad grafica tan alta y precisa, además su aprendizaje es más complicado que los otros ya que utiliza los blueprints que no he trabajado antes. Aunque sus físicas son excelentes, es gratis al principio y se puede modelar personajes en el mismo motor, he decidido descartar este motor ya que creo que sus desventajas son mayores.

Gamemaker studio 2, también lo he descartado, aunque sea idóneo para crear un videojuego en 2D no se podrá exportar al PC gratis como lo hace Unity, que es un programa mucho más completo, donde poder aplicar todos mis conocimientos en programación C# y el cual tiene mucha más información y documentación ya que es el más antiguo de todos estos.

En conclusión, para desarrollar mi proyecto utilizare Unity versión 2019.3.12f1 principalmente para programar el videojuego y Blendr ocasionalmente para la creación de personajes.

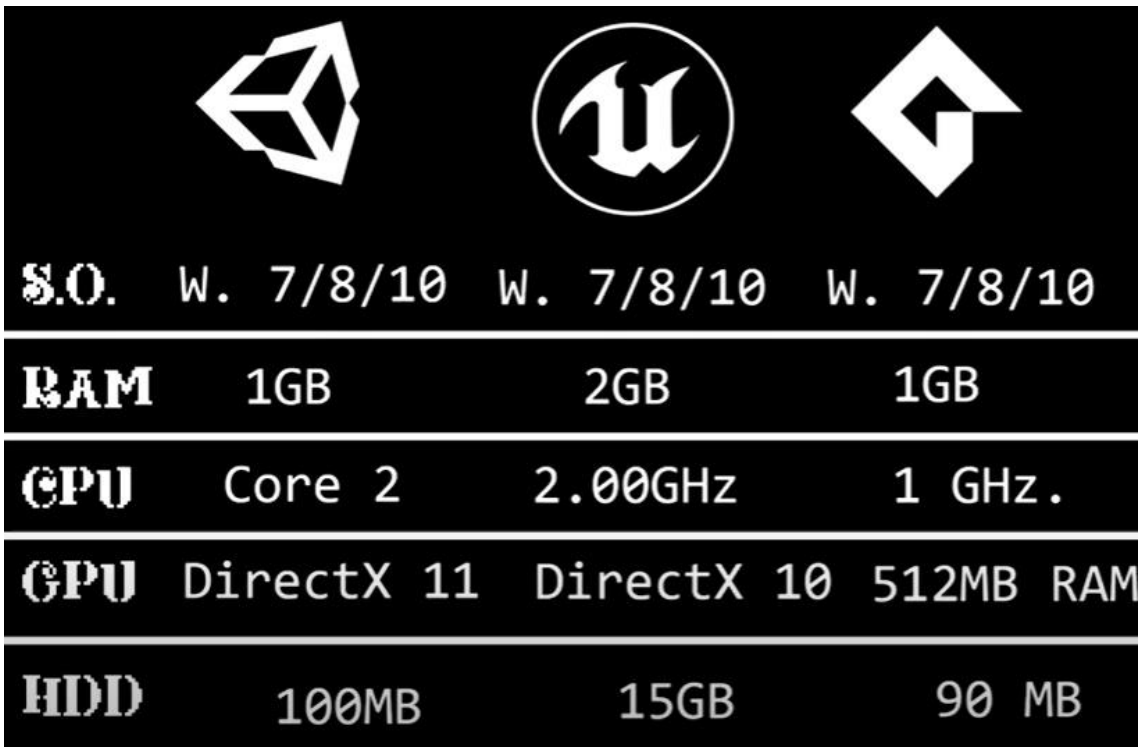


Figura 2-5. Comparación de Requisitos mínimos para cada motor

3. ESTILO

Como ya he explicado en la introducción, desarrollare un juego indie 2D de plataforma del tipo MetroidVania. El estilo de juego determina en mayor parte la interacción del usuario con el juego, por tanto, en este apartado complementaré la información ya aportada sobre el estilo de videojuego y definiré todos sus elementos clave.

3.1. Genero

El género MetroidVania al que pertenece Crazy Robot es un subgénero de los videojuegos de acción y aventura, pero basado en un juego de plataforma no lineal, como ya he comentado, este género surgió de una mezcla de los juegos Metroid y Castlevania.

Es un género que incluye un gran mundo conectado entre sí, sin embargo, habrá partes del mundo que serán inaccesibles hasta que el jugador haya explorado toda la otra zona y además haya encontrado, las herramientas, armas o habilidades que hagan que se abra esta zona del juego. Con estas mejoras, el personaje irá adquiriendo mas fuerza y nivel para derrotar a los enemigos y localizar atajos y secretos (muy comunes en este tipo de juegos). A menudo el jugador, para encontrar esas herramientas deberá volver hacía partes ya exploradas.

En conclusión, para desarrollar este tipo de videojuegos se pondrá énfasis en el diseño de niveles para fomentar la exploración y experimentación de tal manera que los jugadores se sientan inmersos en la historia. Los desarrolladores indies, es decir grupos de desarrolladores muy pequeños y con poco presupuesto como es mi caso, son los principales desarrolladores de este género ya que se criaron jugando a este tipo de juegos y no son proyectos de alto nivel.

3.2. Cámara

La vista lateral o Side Scroller View es la más usada para los juegos de plataforma 2D como es nuestro caso, se le puede dar efecto de profundidad haciendo que las cosas lejanas se muevan más lentas que las cercanas como sucede en Mario Bros

La cámara ira siguiendo continuamente al protagonista desde un plano lateral, por lo tanto, el personaje solo se desplazará hacia la izquierda o derecha, o en caso de que salte hacia arriba y abajo. Este tipo de cámara fue muy utilizado en la era dorada de los arcades y la tercera generación de consolas. Ha habido muchos juegos populares con este tipo de cámara, pero sin lugar a dudas, el más conocido de estos es Mario Bros.

3.3. Gráficos

En este tipo de videojuegos se antepone la importancia de un mapa bien diseñado y unas mecánicas divertidas y atractivas antes que una calidad gráfica excelente. Además, al ser un juego de plataforma 2D como los que se usaban antes, se intentará imitar los gráficos pixelados de los juegos de los 90 para así darle a Crazy Robot un 'toque retro'

Aun así, se intentará cuidar al máximo todas las animaciones, así como los personajes utilizados en este juego. Teniendo en cuenta que no cuento con personal especializado en diseño gráfico y animación usaré la Asset store de Unity, donde consigues descargar gratis multitud de personajes y animaciones, aunque el personaje principal lo modelare yo mismo con la ayuda de Blendr, un software gratis de modelado de personajes ~~gratis~~.

3.4. Predecesores

Como ya he dicho anteriormente este tipo de juegos es una mezcla entre Metroid y Castlevania, aunque desde entonces han aparecido muchos MetroidVania. Por eso a continuación, expondré algunos de los mejores de los cuales me he inspirado para seguir mi trabajo.

3.4.1. Ori and the blind forest

Tenemos a continuación una saga de dos videojuegos, la cual es considerada una de las sagas más bonitas y conmovedoras de todos los MetroidVania. Lo desarrolló la compañía austriaca Moon Studios en Unity, se estrenó el 11 de marzo de 2015 a un precio de 40$ solo para XBOX one, en la actualidad está disponible también para PC y Nintendo switch.

Figura 3-1. Portada de Ori and the blind forest

Es un MetroidVania especial ya que este equipo de desarrolladores además de hacer un diseño de mapa y unas mecánicas perfectas, hicieron que la historia y la ambientación del videojuego tuviera más importancia de lo habitual consiguiendo meter al jugador de lleno en la historia. En 2015, este juego ganó el premio al juego con la mejor dirección artística.

Figura 3-2. Fotograma durante la exploración de mapa

Figura 3-3. Fotograma Ori contra jefe

3.4.2. Hollow Knight

Hollow Knight es otro MetroidVania estupendo creado y desarrollado por Team Cherry, lanzado inicialmente para Windows en febrero de 2017. En la actualidad está disponible para todas las plataformas. El proyecto fue financiado por una corporación de beneficio público estadounidense con 57000$

Figura 3-4. Portada de Hollow Knight

Este videojuego predomina sobre los otros MetroidVania ya que el mapeado de las cuevas que se encuentran en este juego es sublime, tiene una maravillosa banda sonora que realza la ambientación misteriosa que sugiere este juego. Ganó el premio al mejor juego de plataforma de 2017

Figura 3-5. Fotograma explorando el mapa

Figura 3-6. Fotograma luchando

3.4.3. Dead Cells

En este caso tenemos un juego tipo Metroidvania que se mezcla con un género de exploración de mazmorras, creando su propio género nuevo al que llaman roguevania, desarrollado por el estudio francés Motion Twin. Fue lanzado para todas las plataformas en 2018 a un precio de salida de 30$

Figura 3-7. Portada del juego Dead Cells

Este juego es diferente a los otros ya que, aunque todo lo relacionado con el género MetroidVania sigue latente en él, Dead Cells tiene pemadeath, si el jugador muere no revivirá de nuevo, tendrá que empezar el juego de nuevo excepto por algunas herramientas especiales que tendrás para siempre. Lo realmente curioso de este juego es que cada vez que el personaje revive el mapa se vuelve a construir de forma aleatoria, por lo tanto, nunca se experimenta la sensación de repetirlo una y otra vez, siempre se está explorando algo nuevo.

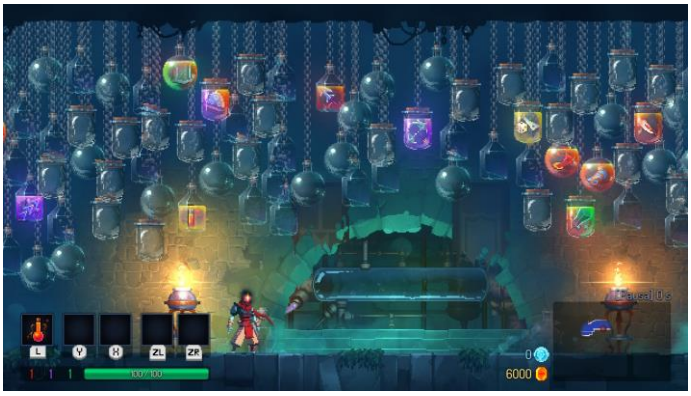


Figura 3-8. Zona de reaparición después de morir

Figura 3-9. Fotograma combatiendo

4. PLATAFORMA

Crazy Robot está dirigido para los jugadores en PC Windows 64 bits. En mi caso lo jugaré con Windows 10 pro, también se puede jugar con Windows 7 o próximas versiones. Mi decisión de elegir el PC como plataforma viene justificada por los puntos siguientes:

- La licencia de Unity es gratis si queremos exportar el juego a PC.
- Los ordenadores personales con Windows, actualmente están al alcance de prácticamente todo el mundo.
- El rendimiento es mucho mayor en el ordenador.
- Los periféricos como el ratón, nos ayudarán en el desarrollo y control.
- Existe la posibilidad de obtener ingresos si pongo el juego a la venta en Steam.

Se analizará más tarde la posibilidad de exportar el videojuego a Xbox one si Unity lo permite de manera gratuita.

5. PROPOSITO Y AUDIENCIA

El propósito de este videojuego es divertir al usuario, trata de sumergir al jugador en la exploración del mapa para poder seguir avanzando. El usuario obtendrá una recompensa al pasar ciertas pruebas difíciles, lo cual servirá de estímulo pare el jugador.

5.1. PEGI

Las siglas PEGI significan Pan European Game Information. Es el sistema que se utiliza para clasificar por edades los videojuegos en Europa. Se utiliza principalmente para informar a los padres sobre el contenido del juego.

Este sistema se puso en vigor en 2003 sustituyendo una serie de procedimientos de clasificación reuniéndolos en este solo, este sistema es respaldado por las principales marcas de videojuegos.

Tanto en el anverso como en el reverso de los videojuegos llevarán etiqueta y descriptores, la etiqueta es la que define a que categoría de edad pertenece el juego. Actualmente existen 6 etiquetas que limitan a las edades de 3,7,12,16,18 y para todos los públicos. En cuanto a los descriptores se pueden distinguir hasta 8 e indican el contenido del videojuego por lo que se utilizan como base para establecer la etiqueta.

5.2. PEGI 12

Los juegos con esta etiqueta contendrán un lenguaje más adulto o escenas de violencia en un contexto de fantasía. Se contempla la desnudez parcial y violencia menos explícita contra personas o animales.

Crazy Robot es un juego en 2D de plataforma, aunque se añadirán ciertas habilidades y armas al protagonista, con las cuales, a medida que avance la historia, ira eliminando a los enemigos que se opongan en su camino.

En conclusión, la audiencia a la que va dirigida este juego son personas mayores de 12 años, las etiquetas y descriptores que deberían aparecer en mi videojuego una vez acabado serían los siguientes:

Figura 5-1. Etiqueta para PEGI 12

Figura 5-2. Descriptor de violencia

6. HISTORIA

A continuación, se presenta la historia que ambientará mi videojuego, a partir de esta historia se irán creando los enemigos y los escenarios del videojuego.

6.1. Prólogo

En el año 2311 toda la población vivía en paz y tranquilidad, no existía ni la pobreza ni las guerras en el mundo. Todas las personas tenían un androide f.11 en su hogar para ayudarles con las tareas diarias, era lo normal. Hasta que un día soleado en plena tarde de una bonita primavera del año 2311, la alarma de androitech (la fábrica en la cual se fabricaban todos los androides del mundo) saltó.

El doctor Kate, encargado de la investigación y el avance de la I.A. de los androides, fue a averiguar porque había saltado la alarma, cuando llegó, no podría creer lo que veían sus ojos, el nuevo androide f.12 había salido de su módulo de fabricación y salía corriendo hacia el ascensor de la planta más alta de la fábrica. El doctor no conseguía entender lo que estaba ocurriendo ¿porque corría el nuevo androide?,¿habría desarrollado una inteligencia emocional nunca vista antes? No sabía cómo reaccionar y se quedó allí, al final del pasillo, como si lo hubiesen petrificado hasta que observo al nuevo androide montándose en el ascensor, el profesor le miro con detenimiento, estaba allí al final del pasillo casi ni podía verlo, la fábrica era enorme, entorno los ojos para verlo mejor y no se creyó lo que pasaba cuando vio al f.12 levantar la mano para despedirse.

6.2. Línea argumental

Lo que no sabía el nuevo androide era que el ascensor solo bajaría dos plantas, ante él y la salida hacía a su libertad le quedarían 4 plantas repletas de obstáculos.

El usuario en este juego controlará al Robot que ha adquirido sentimientos y desea su libertad, deberemos ser capaces de sobrepasar los obstáculos que hay en nuestro camino si queremos llegar a nuestro destino. Al final de cada planta habrá un mini jefe esperando, pero una vez vencidos nos darán nuevas habilidades

Al terminar todas las fases podremos salir y se acabará el juego, no sin antes enfrentarnos con el jefe final que será el doctor Kate, el cual en principio nos explicará que todo ha sido un mal entendido e insistirá para que volvamos con él a la planta de arriba para investigarnos, deberemos negarnos ya que el doctor Kate solo mira por si mismo, si volvemos arriba nos des despiezará en mil pedazos. Si queremos la libertad deberemos enfrentarnos a su androide de protección especial.

7. MECANICAS DEL JUEGO

En el siguiente apartado describiré las características del juego y su funcionamiento, en esta sección solo se mostrará el resultado final, explicaré con más detalle todo el proceso de diseño e implementación en el capítulo 13

7.1. Objetivos

El objetivo del usuario en este videojuego es simple, ir avanzando por el mapa hasta encontrar la salida de la fábrica, aunque el objetivo parezca fácil y sencillo a primera vista, será difícil de llevar a cabo. Al final de cada planta habrá un jefe que como ya he mencionado en apartados anteriores, deberemos derrotar si queremos seguir avanzando.

A medida que vamos progresando por el mapa también se nos presentarán ante nosotros varias puertas que estarán cerradas a primera vista, deberemos explorar bien todo el mapeado para encontrar la manera de seguir.

Siempre podremos movernos por todo el mapa por lo que si creemos que nos hemos dejado algo siempre podremos volver atrás e investigar, a no ser que hayamos derrotado al jefe final y hayamos salido de la fábrica, en tal caso el juego se habrá acabado.

7.2. Recompensas del jugador

Como ya he dicho deberemos de enfrentarnos a unos jefes en cada planta, una vez los derrotemos estos nos darán ciertas habilidades u objetos, el jugador también será recompensando por su exploración, deberá encontrar ciertas herramientas muy escondidas por el mapa.

7.3. Economía

La economía del juego recae sobre la cantidad de batería o vida de nuestro robot y la munición.

Deberemos mantener la batería de nuestro robot en un valor positivo sino el robot se apagará y estaremos en la obligación de volver al último punto de guardado del mapa. Estos puntos están repartidos equitativamente por todo el mapeado. Algunos enemigos también soltarán una pequeña cantidad de batería al eliminarlos.

La munición será necesaria para poder disparar con nuestra arma integrada y protegernos de los enemigos. Si bien es cierto que no es un requisito imprescindible

para seguir avanzando, ya que nuestro robot también tendrá algunos ataques a melee que no necesitan de munición, sin embargo, son ataques mucho más débiles, por lo que nos costará más avanzar en el juego.

7.4. Criterios de éxito

Los criterios de éxito están estrechamente conectados con el alcance de los objetivos (7.1. Objetivos). Para completar el juego el usuario deberá encontrarse con varios jefes que tendrá que eliminar sin morir en el intento para así adquirir las herramientas necesarias afín de seguir avanzando.

7.5. Criterios de fracaso

Estos se sitúan en el momento en que simplemente al perder toda la batería del robot este se apagará, todos los enemigos eliminados volverán a reaparecer y el usuario deberá empezar de nuevo desde el último punto de guardado.

7.6. Controles básicos

Para poder controlar a nuestro robot dentro del juego utilizaremos las flechas del teclado, abajo para agacharnos y la derecha o izquierda para movernos hacia los lados, además para saltar usaremos la barra espaciadora.

Para atacar a los enemigos usaremos la tecla x para disparar, si no tenemos munición utilizaremos la tecla alt izquierda para propinar golpes a corta distancia. Se disparará o golpeará en la dirección en la que este mirando nuestro personaje.

7.7. Habilidades

Como ya he mencionado tendremos dos habilidades principales la primera será en ataque a melee con una guadaña, este ataque tendrá cierto tiempo de reutilización. La otra habilidad será el disparo el cual hará más daño y dependerá como ya dije de la munición que tengamos en el momento.

Cada habilidad tendrá unos efectos visuales y animaciones que explicaré en el apartado 9. A continuación mostraré una tabla con las características detalladas de cada habilidad.

Características de las habilidades del personaje principal					
Icono	Nombre	Descripción	Tiempo de uso	Daño	Rango
	Guadaña	El personaje hace aparecer una guadaña con la que daña a los enemigos.	1s	En un principio hará 25 de daño	Rango cercano al personaje
	Bala	El personaje disparará una bala que dañará al enemigo	—	En un principio hará 50 de daño	Rango infinito hasta que choque con algún objeto

Tabla 7-1. Características del personaje

7.8. Estadísticas

Para el personaje, empezaré con unos atributos ya definidos, estos atributos serán ataque, vitalidad y carga máxima. Aunque dichos atributos irán cambiando a medida que el personaje vaya avanzando de nivel.

A continuación, muestro una tabla de los valores base de cada atributo y lo efectos al subir de nivel en cada uno.

Tabla de efectos de características		
Característica	Efecto por cada nivel subido	Valor Base
Ataque	Suma 5 puntos de ataque al ataque base	20 puntos para la guadaña
		50 puntos para el arma de fuego
Vitalidad	Suma 10 puntos de vida a la vida base	100 puntos de vida
Carga Máxima	Suma 5 objetos más que puedes llevar como máximo	10 objetos máximos

Table 7-2. Efectos de características

7.9. Experiencia y nivel

Cada enemigo otorgará cierta cantidad de experiencia (dependiendo de su dificultad). Inicialmente el personaje tendrá nivel 1. Un nuevo nivel otorgará como he dicho antes un aumento en las estadísticas del personaje. Cada nivel necesitará más experiencia que el anterior. Para el cálculo de experiencia necesaria en mi caso he usado una formula cuadrática que es la siguiente:

$$y=1.5x^2+2x+2$$

La incógnita es el nivel actual. El resultado es la experiencia que necesitamos para alcanzar el siguiente nivel. A continuación, muestro el grafico de la experiencia para los primeros 20 niveles

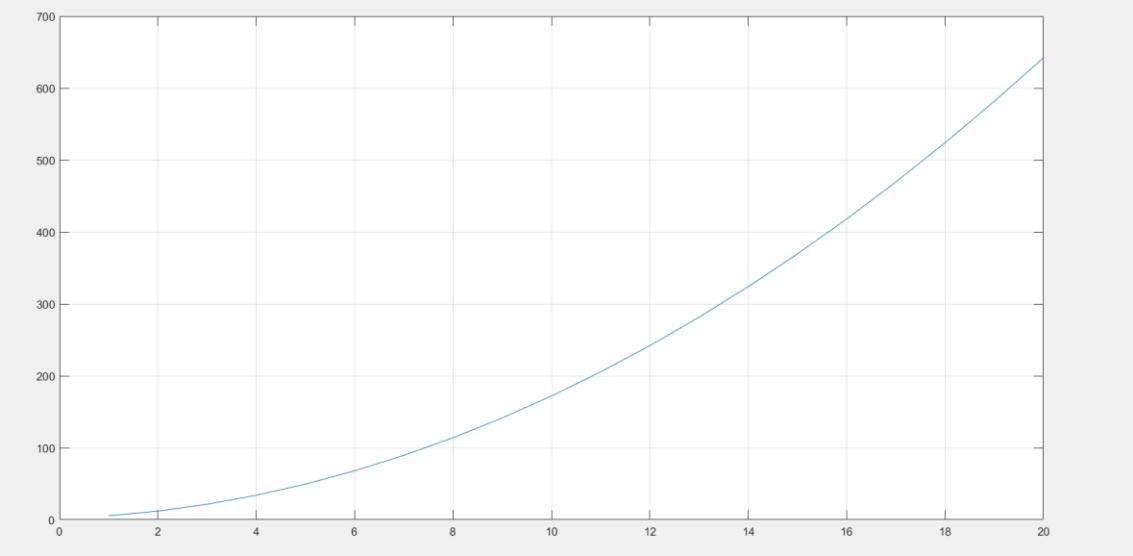


Figura 7-1. Experiencia necesaria por nivel

7.10. Objetos

Los objetos que se pueden encontrar en el juego son contenedores para regenerar la vida, monedas para comprar objetos y munición para las armas.

Las opciones de vida restaurarán un 40% de la vida máxima del personaje, los objetos en forma de bala nos otorgarán 2 balas en total si no tenemos el máximo y habrá tres tipos de monedas, la primera nos otorgará 100 unidades de esta, la segunda 200 y la última y más difícil de encontrar nos concederá 500.

Los objetos se encontrarán flotando a la vez que rotando por el mapeado y al entrar en contacto con alguno de ellos, se usarán en caso de la poción o se guardarán inmediatamente, como es el caso de la moneda y de la munición.

Figura 7-2. Tipo de monedas ordenadas de menor a mayor

Figura 7-3. Vial de vida

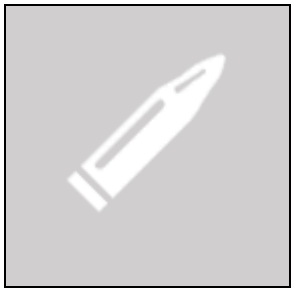


Figura 7-4. Bala para recoger

7.11. Enemigos

Los enemigos son una de las partes más importantes de un videojuego. He incluido 3 tipos de enemigos sin olvidar el jefe final del primer nivel, cada uno con diferentes variaciones

El jefe final es un enemigo único lo que significa que una vez eliminado no volverá a aparecer, en cambio los otros tipos de enemigos volverán a aparecer cada vez que muramos. El jefe final es más difícil de exterminar y posee mecánicas más variadas que los otros tipos de enemigos.

Habrá un enemigo que ataque a distancia y otro que ataque cuerpo a cuerpo, cada uno como ya he dicho con ciertas variaciones.

7.11.1 Enemigos comunes

A continuación, se detallan los 2 tipos de enemigos comunes y algunas particularidades sobre su comportamiento:

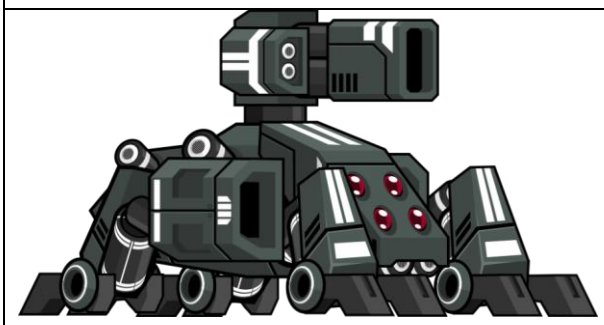
Enemigos Comunes	
Imagen	Características
	**Nombre:** mite **Vida:** 100 **Ataque:** 20 **Comportamiento:** Este enemigo no atacará simplemente, sino que irá andando por las plataformas estrechas o por pasillos para impedir el paso o complicar aún más el manejo de las plataformas.
	**Nombre:** gunner **Vida:** 150 **Ataque cuerpo a cuerpo:**20 **Ataque de bala:**30 **Comportamiento:** Este enemigo atacará a distancia con su arma e irá andando o estará quieto según el comportamiento que le queramos poner.

Tabla 7-3. Enemigos comunes y sus características

7.11.2 Enemigos único

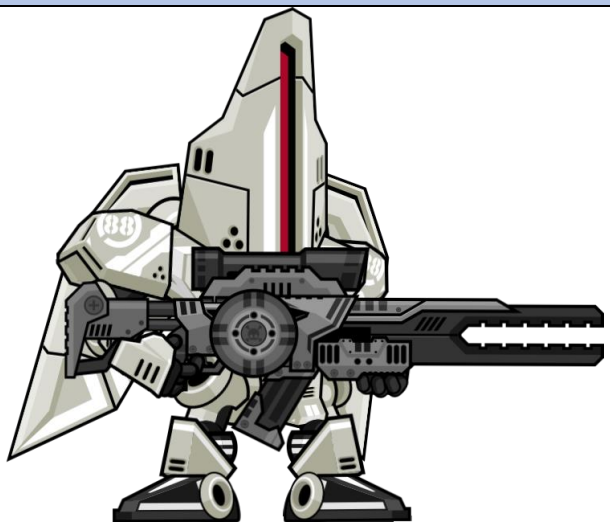
Enemigo Único(jefe)	
Imagen	Características
	**Nombre:** Destroyer 011 **Vida:**500 **Ataque cuerpo a cuerpo:**20 **Ataque de bala:**30 **Comportamiento:** Este enemigo atacará a distancia con su arma y estará quieto, pero se ira teletransportando de un lugar a otro de la escena del jefe.

Tabla 7-4. Enemigo único y características

7.11.3 Mecánicas de movimiento de enemigos

Para los enemigos comunes se ha programado 3 tipos de movimientos los cuales definiré a continuación:

-El primero es estático, lo que significa que el enemigo permanecerá en su posición sin moverse.

-El segundo es Walker(andador)lo cual quiere decir que el enemigo irá andando por una plataforma o donde quiera que esté y cuando se encuentre con un precipicio o una pared este dará la vuelta hasta que ocurra lo mismo.

-El tercero y último es un movimiento de patrulla donde se definen dos puntos, uno a su izquierda y otro a su derecha y el enemigo irá andando de un punto a otro, se definirá también si queremos que espere una vez llegado al punto una cierta cantidad de tiempo para que luego vuelva al otro punto

En el siguiente diagrama de máquinas de estado se puede ver la lógica del código para el movimiento de cada enemigo.

En el anexo de código se podrá encontrar el script usado para esta lógica.

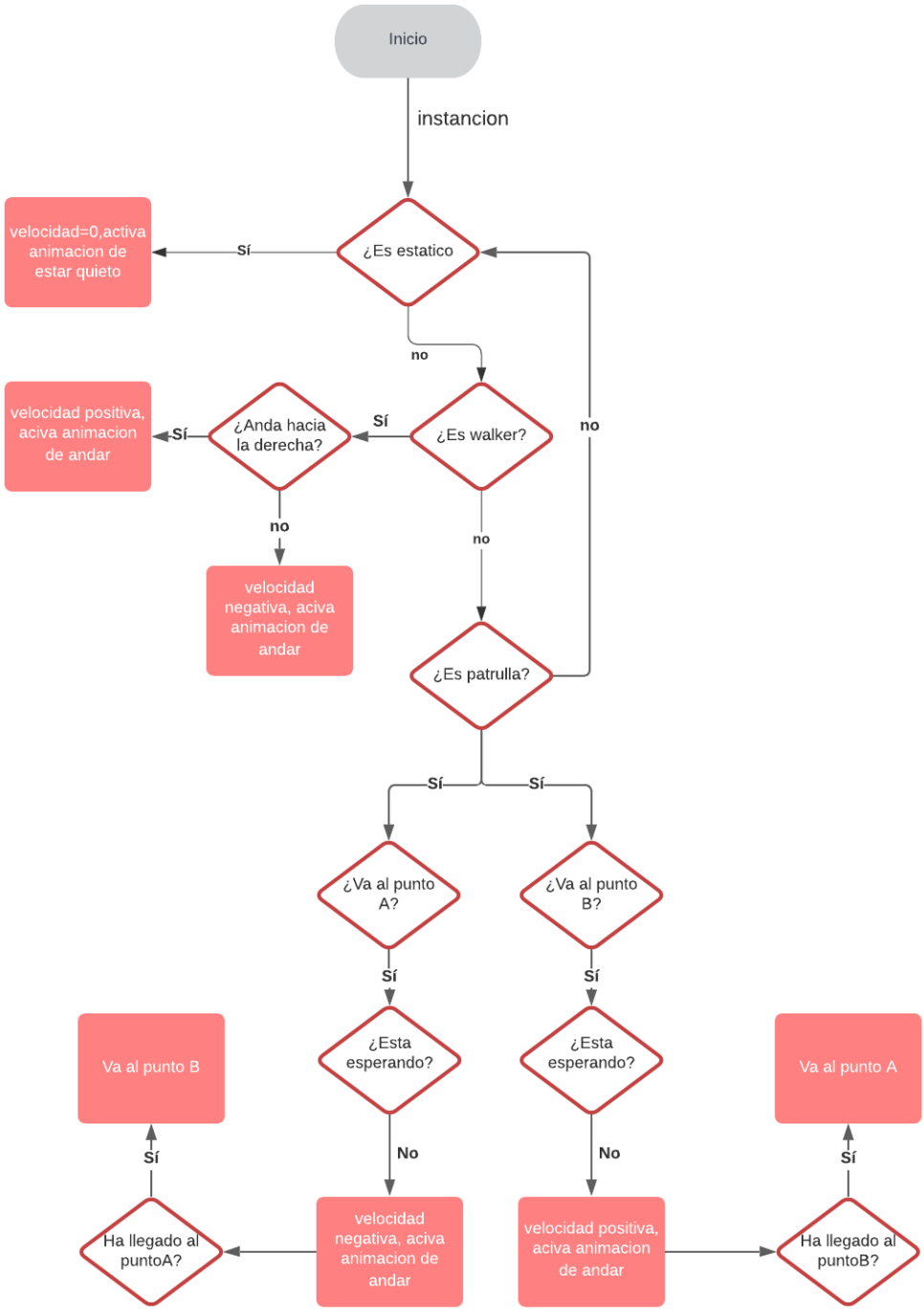


Figura 7-5. Diagrama de movimiento enemigo

7.11.4 Mecánicas de ataque de enemigos

La mecánica de ataque que tendremos que programar será la del tirador en mi caso ‘gunner’.

Como para el movimiento he construido varias posibilidades de ataque, la primera es llamada ‘freqshooter’, con esta opción el enemigo dispara cada cierto tiempo sea cual sea su movimiento. El segundo tipo de ataque es llamado ‘watcher’, lo que supone esta opción es que dispara en cuanto el jugador esté en su rango, si se queda en su rango seguirá disparando.

Se muestra un resumen de la lógica con el siguiente diagrama

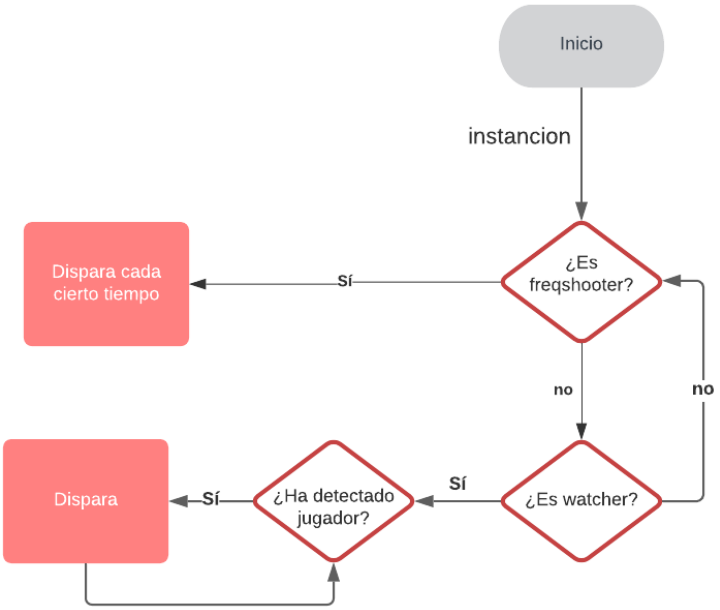


Figura 7-6. Diagrama de ataque enemigo

Como con el anterior en anexo se puede encontrar el código usado para el ataque del enemigo.

7.12. Funcionamiento

El juego trata de completar todos los niveles a lo largo de la historia. Cada nivel tendrá un enemigo único o jefe diferente. Los posibles objetivos pueden ser: eliminar al jefe final, eliminar enemigos o encontrar objetos entre otros.

Cada nivel tendrá una zona de guardado donde al entrar se conservará en un archivo binario dentro de los archivos del juego automáticamente el nivel adquirido y por supuesto los objetos que tenga el jugador en posesión en ese momento. Los mapas podrán volver a recorrerse las veces que uno quiera, aunque el jefe haya sido derrotado, si el personaje muere este será devuelto al principio o a la última zona de guardado. Igualmente, si el usuario cierra la aplicación sin guardar antes, se perderán todos los datos no guardados.

7.13. Flujo de la experiencia de juego

El juego está construido de tal forma que los jefes se ajustarán a un rango de nivel concreto por lo que, si se llega a un jefe sin haber investigado, cogidos algunos objetos o sin haber eliminado a ningún enemigo, a este le será imposible seguir avanzando. Como he mencionado antes este juego tiene la opción de poder volver atrás por lo que se podrá rehacer niveles para conseguir más estadísticas y así llegar más fuerte al jefe

De esta manera el juego supondrá un reto para los nuevos jugadores y los expertos se divertirán intentando completar el juego al 100% encontrado todos los coleccionables.

8. INTERFAZ GRAFICA

Cada vez que iniciemos la aplicación del juego saltará el menú principal de este que es el mostrado a continuación. Desde el menú podremos iniciar el juego desde donde lo guardamos la última vez, acceder a las opciones de este o salir de la aplicación.

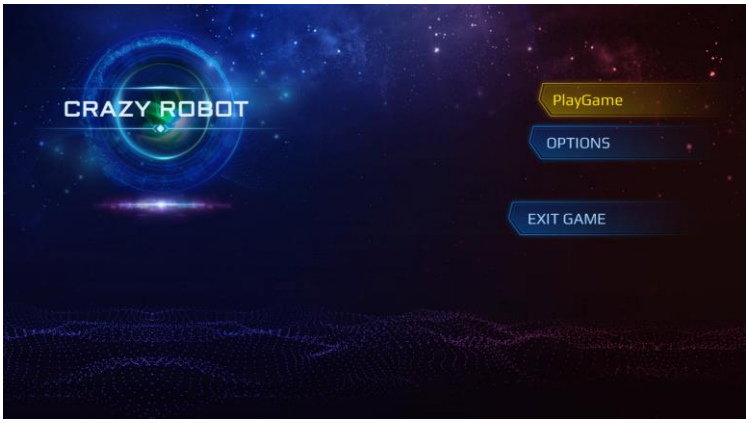


Figura 8-1. Imagen Menú principal

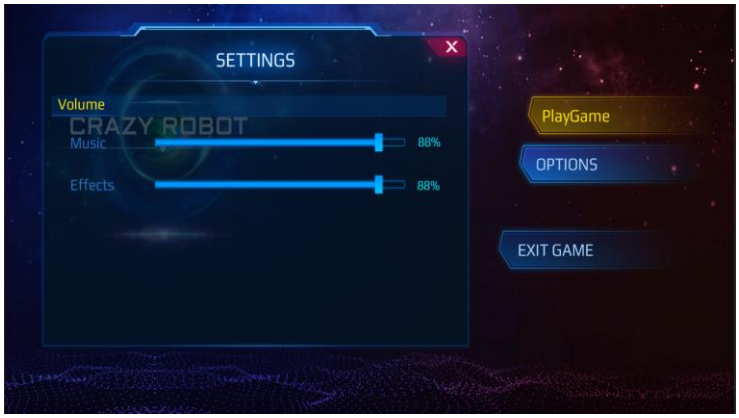


Figura 8-2. Imagen Menú principal con ajustes

Los elementos restantes pertenecientes a la interfaz gráfica de usuario dentro del juego en si son los referentes al personaje (barra de vida, barra de experiencia, objetos…), el menú de pausa, los ajustes y la barra de vida del jefe cuando aparece

8.1 Interfaz principal

La interfaz gráfica del personaje será lo suficientemente pequeña como para no resultar invasiva para el jugador, pero lo suficientemente grande como para que se pueda consultar de manera correcta la vida de nuestro personaje la experiencia que tenemos actualmente y el nivel

Se ha tratado de hacer una interfaz limpia y que sea tan sencilla que no necesite explicación

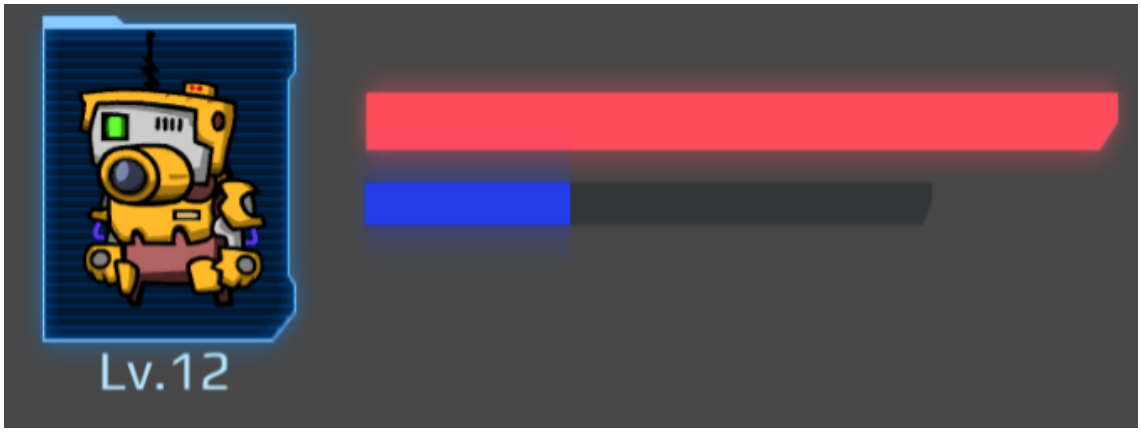


Figura 8-3. Interfaz gráfica de nuestro personaje

La interfaz de nuestro personaje se encontrará en la esquina superior izquierda y mostrará:

- Un retrato del personaje controlado.
- El nivel de nuestro personaje en el inferior del retrato (12) en este caso de ejemplo
- La barra de vida en rojo y la barra de experiencia en azul

Figura 8-4. Interfaz gráfica de objetos

La interfaz gráfica de objetos estará situada en la esquina superior derecha y mostrará:

- En el extremo izquierdo la cantidad de monedas que tenemos en el momento
- A la derecha la cantidad de balas de la que disponemos entre las balas máximas que podemos tener.

8.2 Menús y navegación

El juego contará con varios menús, todos estos menús detienen el transcurso del juego, los menús son los siguientes:

- Menú de pausa para pausar el juego y desde el cual se accede a los demás menús.
- Menú de opciones en el que se puede modificar el volumen de la música y de los efectos, las teclas definidas por el usuario y la calidad grafica
- Menú para abandonar la partida y volver al menú principal.

La tecla definida para acceder al menú de pausa es ‘ESC’. Para acceder a los demás menús bastará con pulsar los diferentes botones que se muestran a continuación. El botón de ‘Restart’ servirá para empezar de nuevo desde el último punto de guardado, implica perder todo lo no guardado hasta ese momento

Figura 8-5. Interfaz gráfica de ajustes en el juego

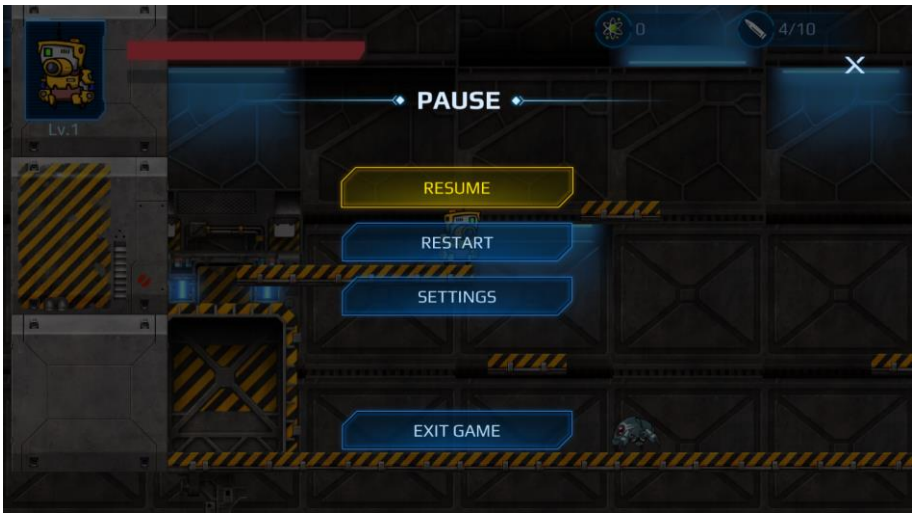


Figura 8-6. Interfaz gráfica del menú de pausa

Figura 8-7. Interfaz gráfica para salir del juego

8.3 Pantalla de Game Over

Al perder toda la vida saltará una pantalla con el típico Game Over de los videojuegos y con un botón de Restart para poder empezar inmediatamente desde el último punto de guardado.

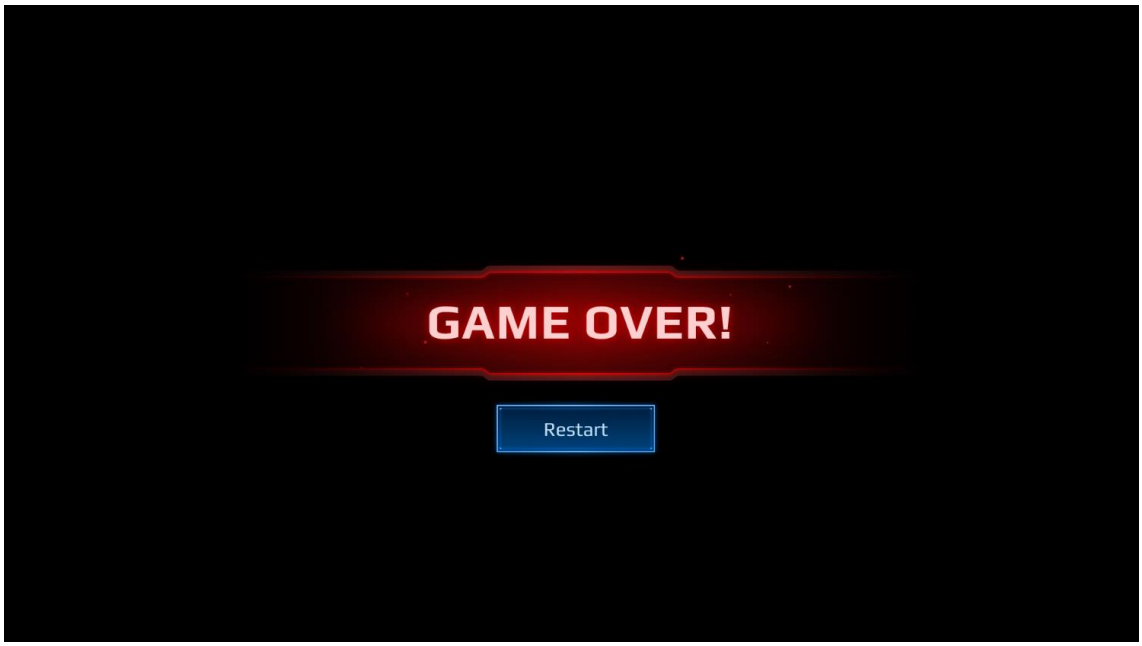


Figura 8-8. Interfaz gráfica de la pantalla al perder

9.EFECTOS VISUALES

A continuación, representaré los diferentes efectos visuales mas importantes con los que cuenta el juego.

9.1 Animaciones

A continuación, iré mostrando los sprites usados para crear las animaciones de los personajes

Los sprites en Unity son imágenes realizadas mediante un mapa de bits con un hardware gráfico especializado sin cálculos adicionales de la CPU, a menudo son pequeños y transparentes. Suelen ser utilizados para dibujar los diferentes frames de una animación.

9.1.1 Animaciones del jugador

Figura 9-1. Animación para cuando está parado

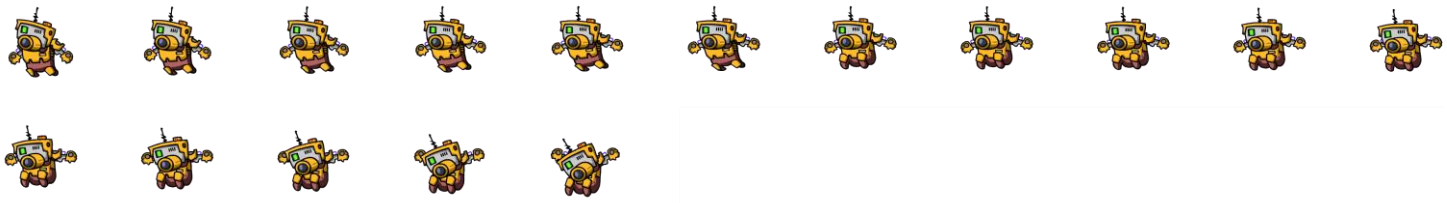


Figura 9-2. Animación de salto

Figura 9-3. Animación cuando está corriendo

Figura 9-4. Animación de ataque

Figura 9-5. Animación de ataque

Figura 9-6. Animación realizada al agacharse

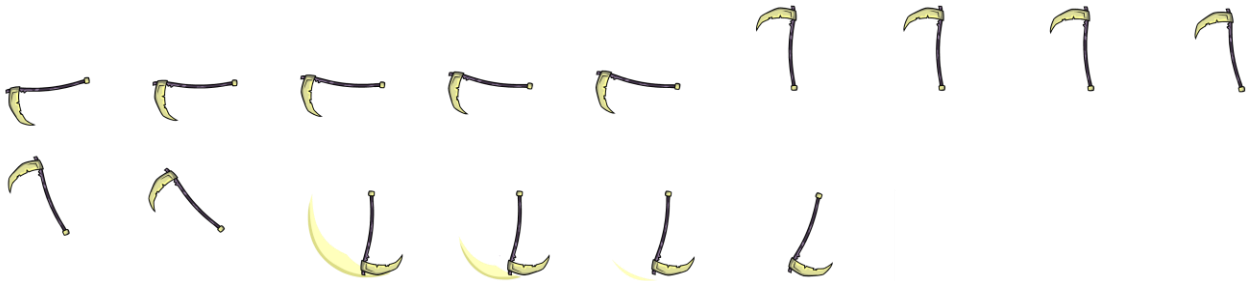


Figura 9-7. Animación de guadaña

9.1.2 Animaciones de los enemigos

Figura 9-8. Animación de 'mite' parada

Figura 9-9. Animación de 'mite' andando

Figura 9-10. Animación de 'mite' muriendo

Figura 9-11. Animación de 'gunner' parado

Figura 9-12. Animación de 'gunner' andando

Figura 9-13. Animación de 'gunner' muriendo

Figura 9-14. Animación de 'gunner' disparando

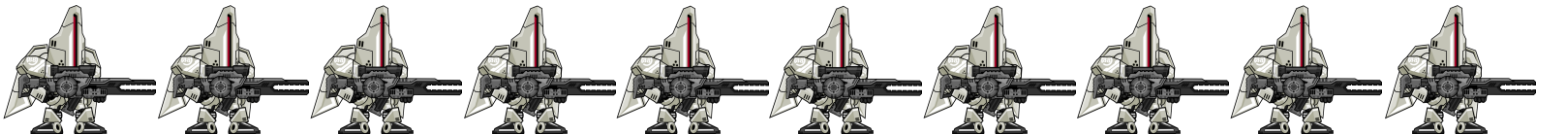


Figura 9-15. Animación de 'destroyer' parado

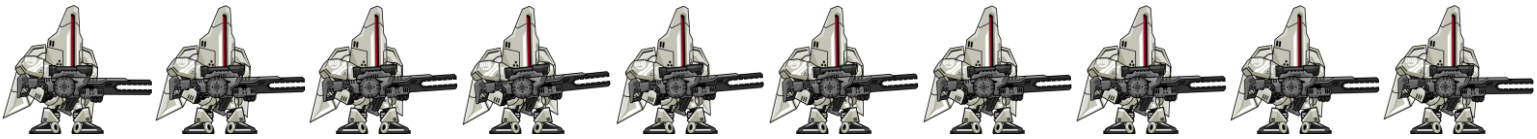


Figura 9-16. Animación de 'destroyer' disparando

Figura 9-17. Animación de 'destroyer' muriendo

9.2 Efectos de proyectiles

En este caso hemos realizado dos prefabs, uno para el proyectil de nuestro jugador y otro para el proyectil de los enemigos.

Dentro de un script se instanciará el efecto del choque de la bala cuando impacte contra un collider. El collider es un componente que se le puede añadir a cualquier objeto de nuestro juego, para detectar colisiones contra este o con otro objeto contra otro collider como es el caso de nuestro proyectil.

Los prefabs son plantillas para poder instanciar los objetos en la escena, las ediciones realizadas dentro de un prefab se guardarán para todos, aunque también se pueden modificar individualmente si fuese necesario.

Figura 9-18. Imagen de bala enemiga

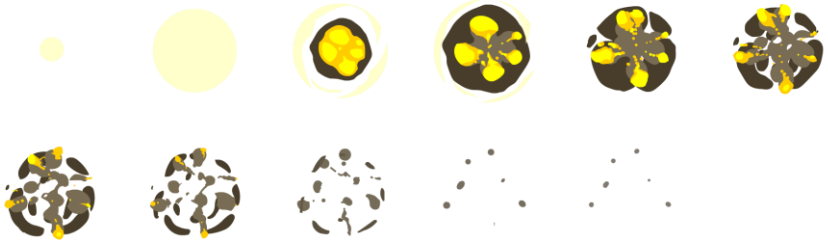


Figura 9-19. Efecto animado de explosión de las balas

Todos los proyectiles se desplazarán solo en el eje x, positivo si dispara hacia la derecha y negativo si dispara hacia la izquierda. Cabe destacar de nuevo que Unity trae integrado un motor de físicas reales por lo que para que no caiga la bala deberemos desactivar la gravedad del objeto que la contiene.

9.3 Efectos de daño en enemigos y personaje principal

Para esto hemos realizado un Script en el que cambiaremos el material de un enemigo al color blanco y con una corrutina esperaremos un periodo de tiempo bastante pequeño para volver a agregar a este su material original. De esta manera simularemos una especie de parpadeo brillante al golpear un enemigo o al ser golpeados como se puede ver en las imágenes siguientes:

Figura 9-20. Capturas de efecto de enemigo siendo golpeado

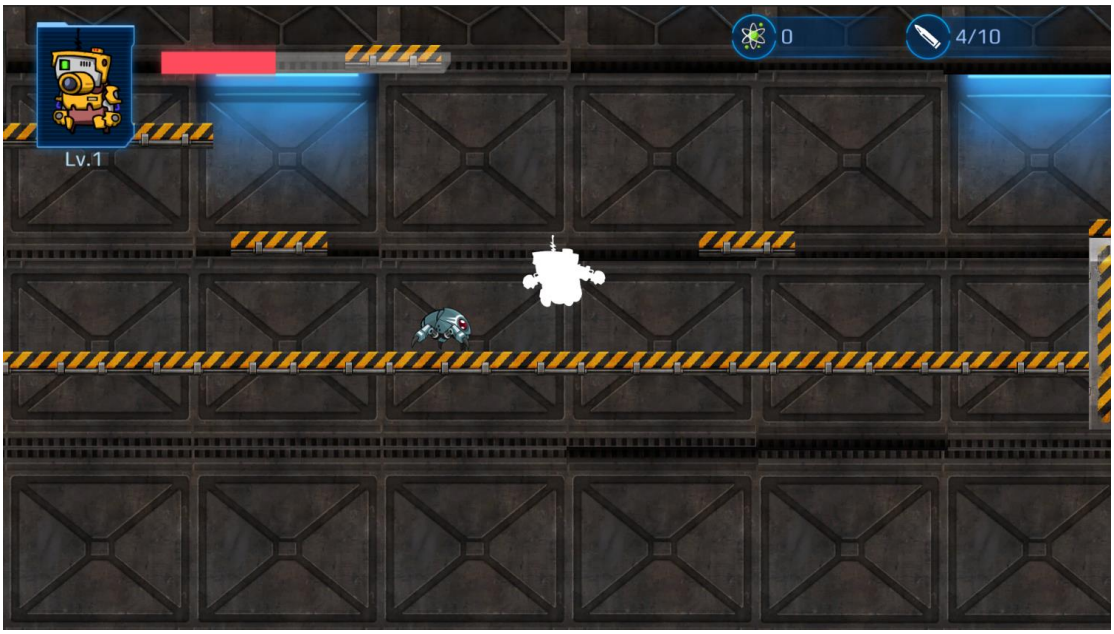


Figura 9-21. Captura de efecto de jugador siendo golpeado

El jugador y los enemigos recibirán también un impulso para atrás y para arriba al ser golpeados dependiendo del tipo de enemigo y del tipo de ataque que reciba.

9.4 Efectos de relleno y vaciado de barras

Las barras de salud tanto la del jefe como la del personaje se irán llenando o vaciando a través de una interpolación lineal, lo que hará que parezca que rellenan o vacían poco a poco.

9.5 Texto flotante

Para mejorar la estética y retroalimentación del usuario con nuestro videojuego he incluido unos textos flotantes cada vez que reciba un daño el enemigo.

Aunque como he mencionado antes el ataque ya está definido con un valor, en realidad, el daño recibido por el enemigo no será exactamente este valor. El daño recibido por el enemigo vendrá dado por tres estadísticas principales que serán el ataque base de nuestro jugador, el ataque de nuestra arma y la defensa de cada enemigo en cuestión.

Como vemos en la siguiente línea de código el daño recibido es un valor aleatorio entre un rango de valores del ataque total, que es la suma de las 2 estadísticas de ataque y la de defensa. Si el daño recibido es el máximo de los valores aleatorios, se duplicará el ataque y con el texto flotante aparecerá ataque critico.

```
totalAttack = attack + weaponAttack + (100 / (100 + enemyDefense));
float finalAttackPower = Mathf.Round(Random.Range(totalAttack - 10, totalAttack +
5));
```

Figura 9-22. Captura de efecto de texto flotante con un ataque critico

9.6 Subida de nivel

Al subir de nivel nos aparecerá en la pantalla un texto confirmando que hemos alcanzado un nuevo nivel

Figura 9-23. Captura del efecto al subir de nivel

9.7 Guardar partida

Al llegar a la plataforma de guardado, se nos guardará como mencioné dije antes todo lo que conlleva la subida de nivel y las monedas. Cuando aparezca el mensaje avisando de que el juego se ha guardado correctamente, nuestro personaje quedará parado durante unos segundos durante los cuales aparecerá un corto efecto de partículas que desaparecerá en cuanto podamos movernos, para que el jugador tenga en cuenta que si muere más adelante volverá a reaparecer en este sitio.

Figura 9-24. Captura del efecto al subir de nivel

10.EFECTOS DE SONIDO

Para este proyecto hemos contado con la ayuda de un ingeniero de sonido con el que hemos realizado 3 pistas de sonido, una para el jefe, otra pista de sonido para todo el juego en sí y una última para el menú principal. Por lo tanto, la música de este juego es única y exclusiva de este juego.

También gracias a la Asset store (tienda de unity) hemos podido descargar gratuitamente algunos efectos de sonido que hemos incluido en el juego.

Para implementar la música en nuestro juego he usado la herramienta Audio Source en Unity, dividido en dos pistas de sonidos, la música y los efectos para así poder controlar el volumen de estas desde el menú principal y desde el menú de pausa.

Una vez hecho esto y con la ayuda de un par de scripts he podido controlar el momento en el que sonaría la música y los efectos. A continuación, esbozo la lista de efectos de sonido que he incluido en el juego:

- Efecto de sonido de salto.
- Efecto de sonido al disparar tanto para enemigos como para el jugador
- Efecto de sonido al impactar una bala o la guadaña del jugador
- Efecto de sonido del jugador al ser golpeado
- Efecto de sonido al subir de nivel
- Efecto de sonido al morir el personaje y al morir un enemigo
- Efecto de sonido al saltar la pantalla de Game Over.
- Efecto de sonido al guardar la partida
- Diferentes efectos de sonidos al pulsar los botones del menú de pausa o del menú principal
- Diferentes efectos de sonido al recoger los contenedores de vida, la munición o las monedas.

11. DEFINICIÓN DEL AMBITO DEL PROYECTO

Como autor del proyecto he tenido que tomar el rol de gestor del proyecto y desarrollador. Esto implica que, aparte de haber tenido que desarrollar el videojuego, también deberé realizar tares de gestión, planificación, control y desarrollo.

Para los siguientes apartados se considerará el proyecto como si fuese un encargo de una empresa de desarrollo de videojuegos, por lo que se deberán estimar costes y recursos de manera real, esto será vital para elegir el ciclo de vida de la gestión de este ambicioso proyecto. Empezaremos a estudiar el alcance del proyecto para así obtener una planificación, gestión y ejecución exitosa

11.1 Alcance del proyecto

No se dispone de cliente real por lo tanto redactaré la definición del proyecto para así obtener los requisitos que debe cumplir el videojuego, es decir, después de analizar la definición se obtendrán los requisitos del proyecto.

11.1.1 Definición del proyecto

Es un videojuego 2D de plataforma tipo MetroidVania, como se ha explicado antes, definiéndose como un subgénero de la mezcla de aventura y acción. El objetivo será entretener al jugador. El usuario deberá completar varios retos avanzando por las plataformas con total libertad hasta acabar el juego.

El juego estará compuesto por varios mapas de plataformas con un jefe al final. Cada mapa aparte de la misión principal de llegar al enemigo único para poder seguir avanzando, también tendrá varios coleccionables repartidos por el mapa. Estos coleccionables formarán parte de objetivos secundarios del juego. Cuando se elimina cel jefe final de una fase se obtendrá una nueva habilidad o un objeto único que abrirá el paso hacia el próximo nivel. Los mapas se podrán volver a recorrer las veces que el jugador quiera y tendrán una duración aproximada de 20 minutos para que no se vuelvan tediosos.

El videojuego contendrá varios enemigos, cada uno de estos con varias mecánicas y características (daño, movimiento, velocidad de ataque, vida...). Los jefes serán un tipo de enemigo único al disponer de un mayor abanico de movimientos, poseerán mas vida y más ataque que los demás.

Los bosses (termino otorgado comúnmente a los jefes en el argot de los videojuegos, que proviene del inglés) solo aparecerán una vez en el juego. Por lo tanto, una vez eliminados el jugador no tendrá que volver a enfrentarse a ellos. Los enemigos tendrán varias mecánicas de movimiento como he explicado anteriormente y algunos no atacarán. Sin embargo, los que si ataquen con proyectiles deberían poder ser esquivados por el jugador, para no hacer imposible el avance del jugador. Al acabar con algunos de los enemigos estos otorgarán al jugador experiencia y de forma aleatoria, pociones, munición o monedas, en este caso los jefes otorgarán más experiencia de lo normal al tener una dificultad mayor.

Cuando la experiencia del jugador consiga o supere la experiencia requerida para subir de nivel, se alcanzará un nuevo nivel y como ya se ha visto antes, aumentará la experiencia para poder subir de nivel y conforme se vaya avanzando de mapa, los enemigos se irán haciendo también más difícil de exterminar y proporcionarán más experiencia al jugador. De esta manera, el juego siempre será un reto para el jugador y nunca se hará aburrido o fácil. Al subir de nivel aumentarán las diferentes características del jugador, el ataque la vida y la carga máxima.

El usuario desplazará a su avatar con las flechas derecha o izquierda según la dirección en la que queramos ir. La flecha hacia abajo servirá para que nuestro personaje se agache y así pasar por sitio estrechos. Con la barra espaciadora nuestro personaje saltará de tal manera que si dejamos presionada la barra nuestro personaje alcanzará más altura, hasta un cierto máximo definido. Nuestro personaje también podrá correr pulsando ctrl izquierdo. Si se quiere controlar con un mando de Xbox conectado al ordenador se podrá dirigir el movimiento con un joystick, el salto con el botón ‘A’ y el personaje correrá pulsando el botón ‘RB’.

El sistema de combate será en tiempo real y para ello usaremos alt izquierdo para atacar cuerpo a cuerpo con nuestra guadaña o ‘B’ en el caso del mando y para disparar ‘x’ con el pc y ‘RT’ en el caso del mando.

Se usará una cámara lateral que irá siguiendo al personaje en todo momento, se ha configurado la cámara para que lo siga con fluidez. La cámara tendrá un cierto tiempo de retraso con lo que dejará que el jugador se mueva primero y con esto conseguiremos un mejor efecto de seguimiento.

Cada vez que se realice una acción de movimiento esta tendrá la animación correspondiente tanto como para nuestro personaje como para todos los enemigos incluidos los enemigos únicos. Como explicamos anteriormente el videojuego también contará con efectos musicales y la música de fondo será única y exclusiva realizada por nosotros mismo.

El guardado de partida se hará cuando penetremos en la zona de guardado, una zona neutral sin enemigos donde al entrar obtendremos un mensaje confirmando que se ha guardado las correspondientes características de nuestro personaje

La interfaz de usuario ocupará la mínima porción posible de la pantalla y ofrecerá información en tiempo real de los recursos más importantes del personaje (nivel, puntos de vida, experiencia, cantidad de monedas, cantidad munición). Desde la interfaz de usuario se tendrá acceso también al menú de pausa donde se podrá controlar el volumen tanto de la música ambiental como de los efectos de sonido, también podremos pulsar el botón Restart para empezar de nuevo desde el ultimo punto de guardado, ajustar los detalles gráficos o salir del juego. Al subir de nivel saltará en la pantalla un mensaje informando que hemos subido de nivel y al morir no saltará la pantalla de Game Over con la opción para empezar de nuevo desde el último punto de guardado.

La información de la interfaz deberá ser clara y concisa para poderse entender sin ninguna explicación por lo que la barra de vida y experiencia tendrán colores vistosos.

Cuando los enemigos reciban daño como se mencionó antes, para mejorar la retroalimentación del usuario se indicará en un texto la cantidad de daño recibida por este que es un valor aleatorio y que dependerá del ataque básico, el ataque del arma y la defensa. Se usarán efectos visuales y sonoros para avisar al usuario de enemigos y del impacto de los ataques.

Se requieren transiciones entre escenas fluidas visualmente.

La ambientación del juego será futurista y de ciencia ficción, centrada en los robots, el juego será un reto para jugadores novatos y profundo para jugadores expertos.

11.1.2 Principales requisitos del proyecto y tiempo estimado por requisito

A continuación, proporcionaré una lista de los principales requisitos extraídos de las definiciones anteriores, enumerados por orden de prioridad. En la tabla también se adjunta una estimación inicial del tiempo necesario para satisfacer cada requisito

Requisitos principales	Estimación (Horas)
1. Usando las flechas del teclado, la barra espaciadora y las teclas alt y x, el usuario efectuará ataques y desplazará al personaje. El personaje tendrá animaciones y efectos fluidos y claros	73
2. Los enemigos se moverán y atacarán el personaje según el estilo de movimiento y ataque que queramos definir y serán dañados por el personaje principal. Los enemigos también tendrán sus propios efectos y animaciones claras.	82
3. Se programará la cámara que seguirá al personaje automáticamente con cierta fluidez por todo el mapa delimitando su rango.	6
4. Nuestro robot ganará experiencia de los enemigos derrotados, subirá de nivel, aumentando sus características	13
5. Se tendrá una interfaz de usuario clara y simple que reflejará los objetos que tenemos en el momento y la cantidad de vida y experiencia. También nos permitirá configurar el juego desde ciertos submenús o salir/entrar del juego.	24
6. Se contará aparte de la gama de enemigos ya mencionados, varios enemigos únicos que otorgarán más experiencia al jugador, ya que serán más fuertes y tendrán mecánicas más interesantes que el resto.	43
7. El uso de efectos visuales para aportar más información al usuario, como efecto al disparar, perdida de vida, daño causado, partida guardada, subida de nivel. Los efectos de texto de daño y explosión de bala deben ser vistosos y las transiciones entre escenas deberán ser fluida.	20
8. El juego será dividido en varios mapas llenos de plataformas que pueden ser móviles o no y obstáculos que tendremos que superar para poder continuar. La construcción de los niveles debe ser variada y presentando con cierta dificultad para llamar la atención del usuario. Se deben de poder volver a recorrer las veces que uno	84

quiera.	
9. El juego deberá tener varias zonas de guardado donde al entrar se guarden todas las características de nuestro personaje.	12
10. La música y los efectos deben ser variados para cada acción en la pantalla.	26

Tabla 11-1. Requisitos principales y estimación de horas

11.2 Estructura de desglose de requisitos

A continuación, expondré un esquema tipo RBS (Requierement Breakdown Structure), este está constituido por los requisitos en orden de importancia. Expone una visión clara del grado en que la solución está definida y así permite elegir el modelo de ciclo de vida de la gestión del proyecto por su grado de complejidad. Este esquema constituye la parte superior del WBS (Work Breakdown Structure) del cual hablaremos en la parte de planificación. Los principales requisitos corresponden al nivel superior, en este caso en color rojo, en azul claro tenemos los requisitos no funcionales y en amarillo claro los funcionales. La ventaja de estos esquemas es que para el cliente son resultan muy intuitivos

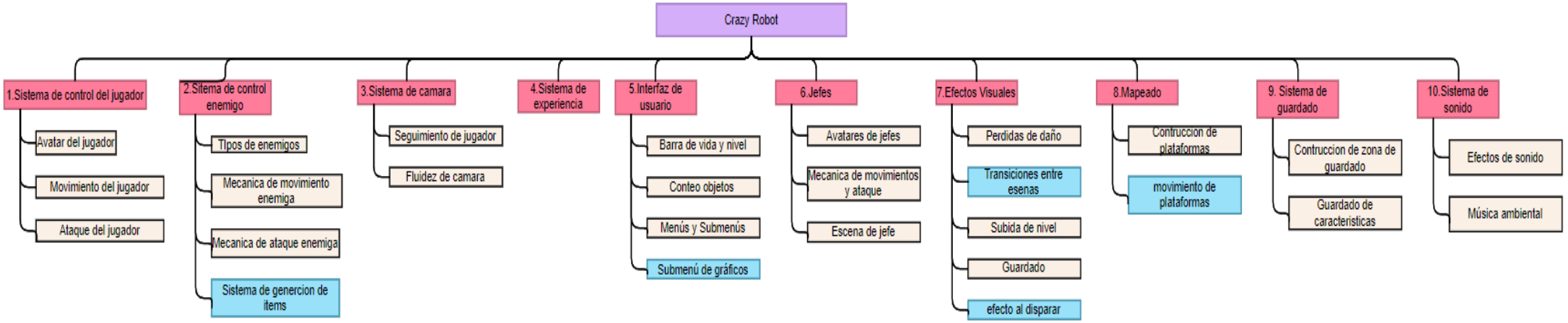


Figura 11-1. Diagrama RBS

11.3 Ciclo de vida de la gestión del proyecto

Para la planificación del proyecto y ejecución de las futuras tareas de desarrollo que se definirán en la etapa de planificación se ha optado por el uso de una metodología ágil (Scrum). Usaré la metodología Scrum ya que no tengo mucha experiencia en este tipo de proyectos y el RBS aun siendo completo no está muy detallado.

Esta metodología es una de las más usadas por las industrias para el desarrollo del software y consiste en un conjunto de prácticas y roles que permiten el trabajo de entregas incrementales de un producto.

Como vemos en el esquema el proyecto tiene todos los requisitos definidos, pero teniendo en cuenta mi experiencia y la envergadura de este proyecto los requisitos pueden cambiar modificando o eliminando alguna funcionalidad menos importante. Como ya he mencionado, al no haber trabajado nunca en un proyecto así el aprendizaje será por descubrimiento, una característica de la metodología elegida. Necesito una metodología para afrontar un gran proyecto con incrementos de funcionalidad, intentando tener un producto funcional desde el primer momento

La metodología Scrum trae consigo reuniones periódicas muy importantes para la planificación que no podré realizar al tratarse de un proyecto en solitario. Aunque no dispongo de grupo de proyectos se han consultado periódicamente unos aficionados a los videojuegos consiguiendo así un grupo de testers (usuarios con experiencia en el mundo de videojuegos que prueban estos mismos para comunicar a los desarrolladores errores en el software) los cuales periódicamente descubrían anomalías que han ido siendo corregidas.

11.4 Declaración general del proyecto

Para este apartado expondremos el documento de declaración general del proyecto. Un medio para pasar a la fase de planificación, deberá ser aprobado por el cliente, el gestor y el equipo de desarrollo si lo hubiese.

<table>
<tr><td>PROJECT OVERVIEW STATEMENT</td><td>Project name
Crazy Robot</td><td>Project No.
001</td><td>Project Manager
Diego Gomez Colliga</td></tr>
<tr><td colspan="4">Problem/Opportunity
Es un hecho que la industria del videojuego está en expansión, una industria que genera miles de millones de dólares al año, por lo tanto, tenemos la oportunidad de un negocio rentable</td></tr>
<tr><td colspan="4">Goal
El objetivo de este proyecto es obtener al final de la fase de ejecución un videojuego comercializable el cual satisfaga los requisitos de calidad y funcionalidad actuales de los jugadores.</td></tr>
<tr><td colspan="4">Objectives
Obtener al menos un videojuego que contenga las siguientes funcionalidades:
• Control fluido del personaje
• Combate satisfactorio para el usuario
• Sistema de experiencia
• Varios enemigos y jefes
• Mapeado bien efectuado
• Efectos de sonido y música</td></tr>
<tr><td colspan="4">Success criteria
• Que un 35% de los clientes potenciales compren el juego
• Que el 75% de los usuarios que prueben el juego lo valoren positivamente</td></tr>
<tr><td colspan="4">Assumptions, Risks, Obstacles
• Disponer del tiempo necesario para llevar a cabo el proyecto
• Posibilidad de que el equipo informático utilizado para el proyecto deje de funcionar en algún momento
• Disponer de conexión a internet para buscar información.</td></tr>
<tr><td>Prepared by
Diego Gomez Colliga

Autor TFG</td><td>Date
30/11/20</td><td>Prepared by
Juan Antonio Sánchez Segura

Tutor TFG</td><td>Date
30/11/20</td></tr>
</table>

Tabla 11-2. Documento POS

12. PLANIFICACIÓN DEL PROYECTO

La planificación del proyecto es uno de los aspectos más importantes del proyecto ya que nos ayudará a la hora de comprender lo que debemos hacer y seremos mucho más eficientes lo que nos ahorrará mucho tiempo.

Para la planificación del proyecto descompondré el RBS en tareas de tal manera que podremos estimar el tiempo y el esfuerzo. Estas tareas no podrán superar el tiempo de una iteración, el cual definiremos más adelante, estimaré también valores importantes como la duración los costes y los recursos necesarios para llevar a cabo el proyecto.

12.1 Desglose del trabajo

En este apartado desglosaremos el WBS (Work Breakdown Structure) el cual hemos llevado a Excel para crear una tabla de todas las tareas, añadiendo el tiempo estimado de duración y la asignación de prioridad. Para que sea más fácil para el lector se adjuntará la tabla de las tareas con el tiempo estimado en vez del esquema WBS. Tal y como se puede ver en el diagrama las tareas rojas serán las superiores del RBS y las amarillas las inferiores a las mismas.

Crazy Robot	
TAREAS	**TIEMPO ESTIMADO(Horas)**
1.Sistema de control del jugador	
1.1 Avatar del jugador	
1.1.1 Diseño	3
1.1.2 Implementación	6
1.2 Movimiento del jugador	
1.2.1 Diseño	12
1.2.2 Implementación	18
1.2.3 Pruebas	3
1.3 Ataque del jugador	
1.3.1 Diseño	12
1.3.2 Implementación	17
1.3.3 Pruebas	2
2.Sistema de control enemigo	
2.1 Tipos de enemigos	
2.1.1 Diseño de todos los enemigos	2
2.1.2 Implementación	5
2.1.3 Pruebas de animaciones	3
2.2 Mecánica de movimiento enemiga	

2.2.1 Diseño de 3 tipos de movimientos	5
2.2.2 Implementación	20
2.2.3 Pruebas	3
2.3 Mecánica de ataque enemiga	
2.3.1 Diseño de dos tipos de ataques	4
2.3.2 Diseño de detección de enemigos	3
2.3.3 Implementación	14
2.3.4 Pruebas	2
2.4 Sistema de generación de ítems(objetos)	
2.4.1 Diseño	4
2.4.2 Implementación	8
2.4.3 Pruebas	1
2.5 Pruebas	
2.5.1 Pruebas propias	5
2.5.2 Pruebas de usuarios	3
3.Sistema de cámara	
3.1 Seguimiento del jugador	
3.1.1 Diseño	1
3.1.2 Implementación	1
3.1.3 Pruebas	0.5
3.2 Fluidez de cámara	
3.2.1 Diseño	1
3.2.2 Implementación	2
3.2.3 Pruebas	0.5
4.Sistema de experiencia	
4.1 Diseño	2
4.2 Implementación	10
4.3 Pruebas	1
5.Interfaz de usuario	
5.1 Barra de vida y nivel	
5.1.1 Diseño	1
5.1.2 Implementación	4
5.1.3 Pruebas	0,5
5.2 Conteo de objetos	
5.2.1 Diseño	1
5.2.2 Implementación	2
5.2.3 Pruebas	0,5
5.3 Menú y submenús	
5.3.1 Diseño	1
5.3.2 Implementación	2
5.3.3 Pruebas	0,5
5.4 Submenús de gráficos	
5.4.1 Diseño	1
5.4.2 Implementación	2
5.4.3 Pruebas	0,5

5.5 Pruebas	
5.5.1 Pruebas propias	3
5.5.2 Pruebas de usuarios	5
6.Jefes	
6.1 Avatares de jefes	
6.1.1 Diseño de todos los jefes	1
6.1.2 Implementación	2
6.1.3 Pruebas de animaciones	1
6.2 Mecánica de movimiento y ataque	
6.2.1 Diseño	1
6.2.2 Implementación	16
6.2.3 Pruebas	2
6.3 Escena de jefes	
6.3.1 Diseño	1
6.3.2 Implementación	8
6.3.3 Pruebas	1
6.4 Pruebas	
6.4.1 Pruebas propias	7
6.4.2 Pruebas de usuarios	3
7.Efectos visuales	
7.1 Perdidas de daño	
7.1.1 Diseño	0.5
7.1.2 Implementación	3
7.1.3 Pruebas	1
7.2 Transición entre escenas	
7.2.1 Diseño	0,5
7.2.2 Implementación	1
7.2.3 Pruebas	0,5
7.3 Subida de nivel	
7.3.1 Diseño	0,5
7.3.2 Implementación	1
7.3.3 Pruebas	1
7.4 Guardado	
7.4.1 Diseño	2
7.4.2 Implementación	2
7.4.3 Pruebas	0,5
7.5 Efecto al disparar	
7.5.1 Diseño	1
7.5.2 Implementación	1
7.5.3 Pruebas	0,5
7.6 Pruebas	
7.6.1 Pruebas propias	1
7.6.2 Pruebas de usuarios	3
8.Mapeado	
8.1 Construcción de plataformas	

8.1.1 Diseño	16
8.1.2 Implementación	41
8.1.3 Pruebas	5
8.2 Movimiento de plataformas	
8.2.1 Diseño	4
8.2.2 Implementación	5
8.2.3 Pruebas	3
8.3 Pruebas	
8.3.1 Pruebas propias	3
8.3.2 Pruebas de usuarios	7
9.Sistema de guardado	
9.1 Construcción de zona de guardado	
9.1.1 Diseño	0,5
9.1.2 Implementación	2
9.1.3 Pruebas	0,5
9.2 Guardado de características	
9.2.1 Diseño	1
9.2.2 Implementación	6
9.2.3 Pruebas	2
10.Sistema de sonido	
10.1 Efectos de sonido	
10.1.1 Diseño	4
10.1.2 Implementación	2
10.1.3 Pruebas	5
10.2 Música ambiental	
10.2.1 Diseño	8
10.2.2 Implementación	2
10.2.3 Pruebas	5

Tabla 12-1. Nuevos grupos de tareas y su duración

Estimación Resumida por grupo de tareas	
Grupo de Tareas	Tiempo Estimado (Horas)
1	73
2	82
3	6
4	13
5	24
6	43
7	20
8	84
9	12
10	26
TOTAL:	383

Tabla 12-2. Estimación resumida por grupo de tareas

Como vemos el total de horas estimadas para nuestro proyecto es de 383, como ya he remarcado antes, este tipo de proyectos suele contar con al menos un grupo de 10 de personas para así dividir las tareas y disminuir el tiempo de trabajo de cada uno buscando la eficiencia. En este caso el proyecto es realizado por una única persona por lo que excluiré la tarea 8 y la 6. En su lugar haré un mapeado simple de demostración con un solo jefe. El tiempo estimado finalmente es de 256 horas. Es importante remarcar que en este apartado no se contempla el tiempo dedicado a este documento

12.2 Estimación de duración

Para la elaboración de este proyecto imitaremos una jornada laboral de 8 horas contando con los días de descanso del fin de semana con lo que obtenemos una duración del proyecto de 3 meses. Dividiremos los 3 meses en 6 iteraciones de 15 días cada una. Las tareas comprendidas en cada iteración se mostrarán a continuación. La fecha de finalización será el 22 de febrero dejando el tiempo restante para terminar este documento el cual se estima que durará 90 horas.

TABLA DE ITERACIONES			
Iteraciones	**Tareas comprendidas**	**Fecha de inicio**	**Duración estimada(horas)**
1	1.1, 1.2	30/11/20	42
2	1.3, 2.1	14/12/20	41
3	2.2, 2.3	28/12/20	51
4	2.4, 2.5, 3, 4	11/01/21	40
5	5, 7	25/01/21	44
6	9,10	08/02/21	38
			TOTAL:256

Tabla 12-3. Iteraciones

12.3 Estimación de recursos

Los recursos de un proyecto suelen ser lo bienes o activos que posee una empresa para poder llevar a cabo el proyecto. El recurso más importante siempre son las personas, pero también dispondrán de equipos tecnológicos y herramientas que aumenten el rendimiento de trabajo.

12.3.1 Recursos humanos

Como el producto será solo realizada por una persona, esta persona deberá realizar todos los roles del grupo de desarrollo mínimo necesario para llevar a cabo este trabajo los cuales son los siguientes:

- Gestor de Proyectos
- Diseñador Gráfico con experiencia en Photoshop
- Modelador 2D con experiencia en Blender
- Animador 2D con experiencia en Blender
- Programador con experiencia en Unity 2D
- Diseñador de videojuegos
- Ingeniero de sonido

Para calcular nuestro sueldo actuaremos como si hiciéramos solo el trabajo de programador de videojuego, en la actualidad el sueldo medio de un programador de videojuegos en España es de 1740 euros brutos al mes y si dividimos entre las horas trabajadas de media en un mes obtenemos un sueldo medio de 9.88 euros la hora,

Nuestro proyecto está estimado en 256 horas por lo que nuestro sueldo sería de **2530,90 €**.

12.3.2 Recursos Técnicos

En este apartado daré a conocer la infraestructura informática y software necesario para terminar este trabajo en el caso de que fuese real. Si el proyecto fuese real se estima que se terminaría en un plazo de dos meses.

EQUIPO INFORMATICO	Precio de adquisición	Cuota de uso (por hora)	Coste
Portátil MSI GL65 9SEK	1.199,99 €	0,04€	11,68 €
IPad Pro	999,99 €	0,038 €	9,74 €
		TOTAL:	**21,42 €**

Tabla 12-4. Tabla de iteraciones

Para calcular el coste se dividirá el precio de adquisición entre su vida útil en horas y se multiplicará por el total de horas de nuestro proyecto. Suponemos que el portátil tiene una vida media de 3 años al igual que el IPad y el mando de 2 años.

GASTOS DE SERVICIOS (2 MESES)		
Concepto	**Precio al mes**	**Coste**
Internet O2 fibra óptica	42,99 €	85,98 €
Luz	70,00 €	140 €
Agua	15,64 €	31,28 €
Oficina en Sevilla	250 €	500 €
	TOTAL:	**757,26 €**

Tabla 12-5. Gastos fijos de servicio

Estos son los gastos fijos de servicios que la empresa debe subsanar para todos los precios se ha usado la media estimada en España.

Para un proceso real se deberá de utilizar mucho más software dedicado a tareas concretas como un software de creación de sonido, a continuación, contemplare solo el software utilizado en mi caso.

GASTOS SOFTWARE		
Concepto	**Razón**	**Precio (2 meses)**
Unity 2021 1.10f1	Motor de videojuegos	-
Adobe Photoshop CC	Creación de algunos sprites 2D	24,18 €
Blender 2.91	Modelado y animación	-
Microsoft Office	Creación de informes y tablas	14,00 €
	TOTAL:	**38,18 €**

Tabla 12-6. Gastos en Licencias de Software

12.4 Estimación de costes

Para este apartado en cuestión usaremos todos los precios expuestos anteriormente para realizar un presupuesto estimado. En la próxima tabla se enseñará un resumen de gastos en el caso de tratarse de un proyecto real. El precio es la suma de todos los gastos más un 25% de beneficios a los cuales se les restara un 15% de impuestos de sociedades para emprendedores.

RESUMEN ESTIMACION DE COSTE Y PRECIO FINAL	
Concepto	**Precio**
Mano de obra	2530,90 €
Equipo informático	21,42 €
Servicios	757,26 €
Software	38,18 €
TOTAL, GASTOS:	3347.76 €
PRECIO ESTIMADO (+25% de beneficios)	**4184,7 €**
BENEFICIOS ANTES DE IMPUESTOS	836,94 €
BENEFICIOS DESPÚES DE IMPUESTOS	**711,39 €**

Tabla 12-7. Resumen costes y precio final

Para completar del todo este apartado se realizará la tabla de costes reales como se muestra, esta vez con la duración estimada de 3 meses

COSTES REALES	
Mano de obra (desarrollo)	1771,63 €
Mano de obra (documentación)	622,44 €
Licencia Software	57,27€
TOTAL:	**2451,34 €**

Tabla 12-8. Costes reales del proyecto

Como se puede observar los costes de servicio de la empresa para el caso real desaparecen, la mano de obra es más barata ya que no se deberá pagar deducciones, la licencia se tendrá que pagar durante 3 meses por lo que aumentará ligeramente su precio. El equipo informático usado ya era propiedad del alumno por lo que tampoco aparece en los costes.

La mayor variación en este caso es la suma de la mano de obra por la realización de este documento que estimándolo en 90 horas y multiplicando por el precio por hora de un desarrollador anteriormente calculado ascendería a un total de 622,44 euros, de nuevo sin tener en cuenta los impuestos que no se aplican en el alumno. Por lo que junto a la mano de obra de desarrollo esta sumaría un total de mano de obra de **2394,07 €**.

12.5 Diagrama de Gantt

Ya he mencionado anteriormente que este videojuego se realizare mediante 6 iteraciones con una duración de 2 semanas para cada una. Ahora veremos el diagrama de Gantt de cada iteración. Se ha podido realizar llevando la tabla de las tareas a Excel y calculando el diagrama con esta misma herramienta. Se verá que el progreso es lineal ya que solo consta de una persona. El diagrama se mostrará en horas y a la izquierda estará el número de la tarea en cuestión.

Figura 12-1. Diagrama de Gantt iteración 1

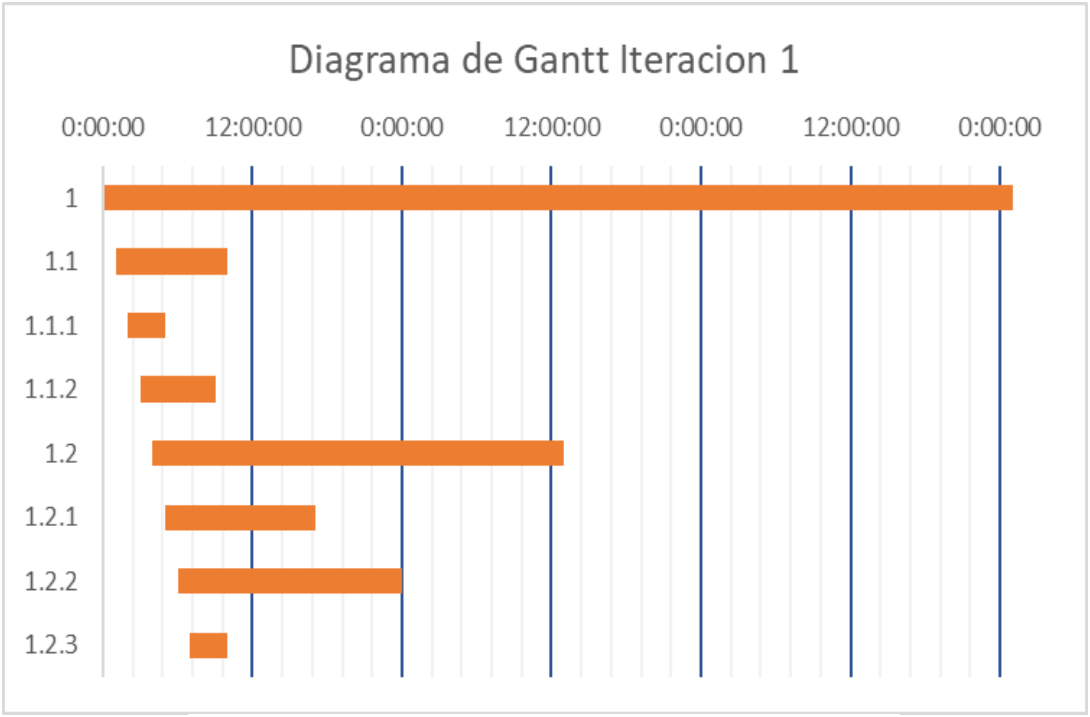
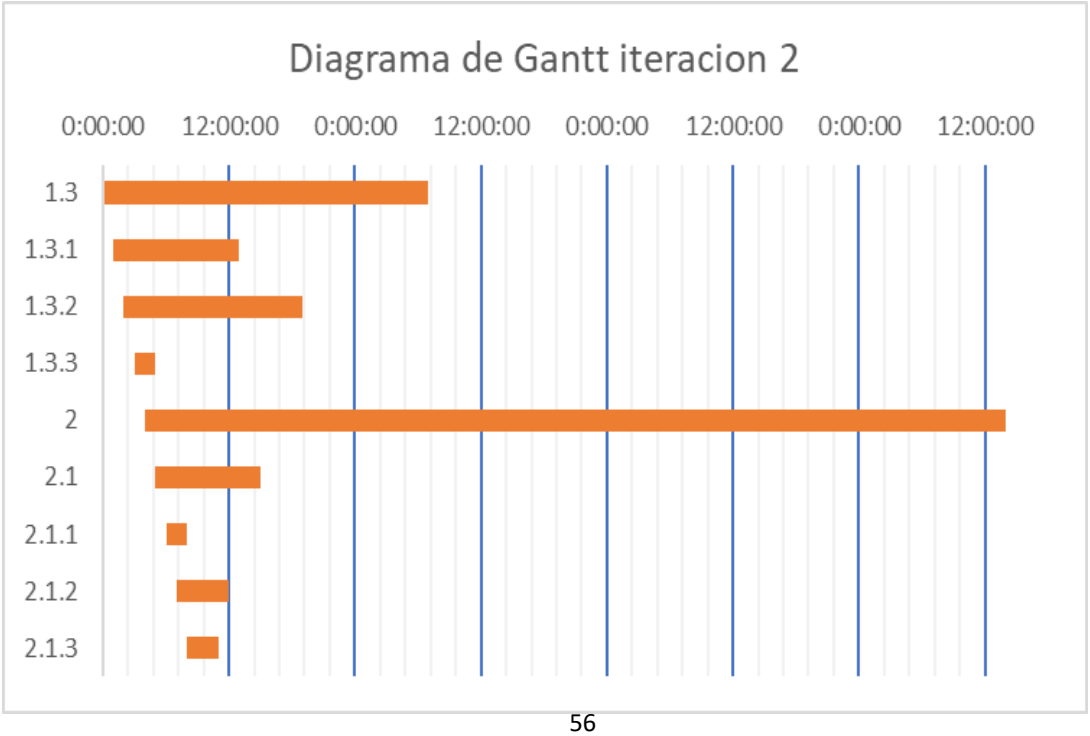


Figura 12-2. Diagrama de Gantt iteración 2



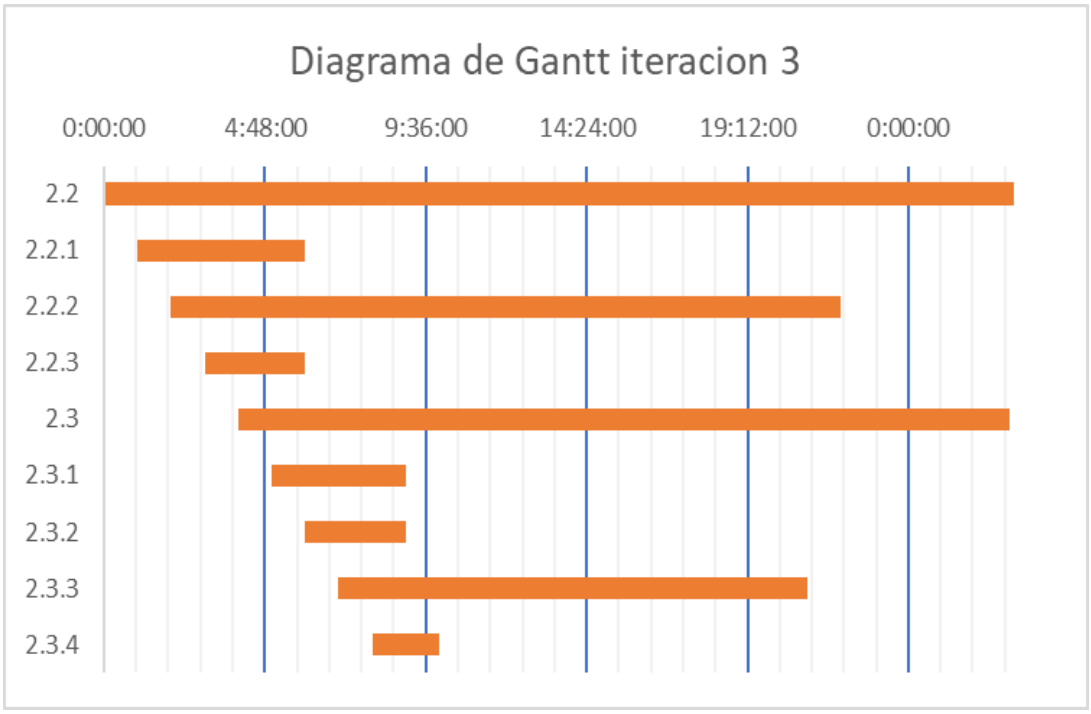


Figura 12-3. Diagrama de Gantt iteración 3

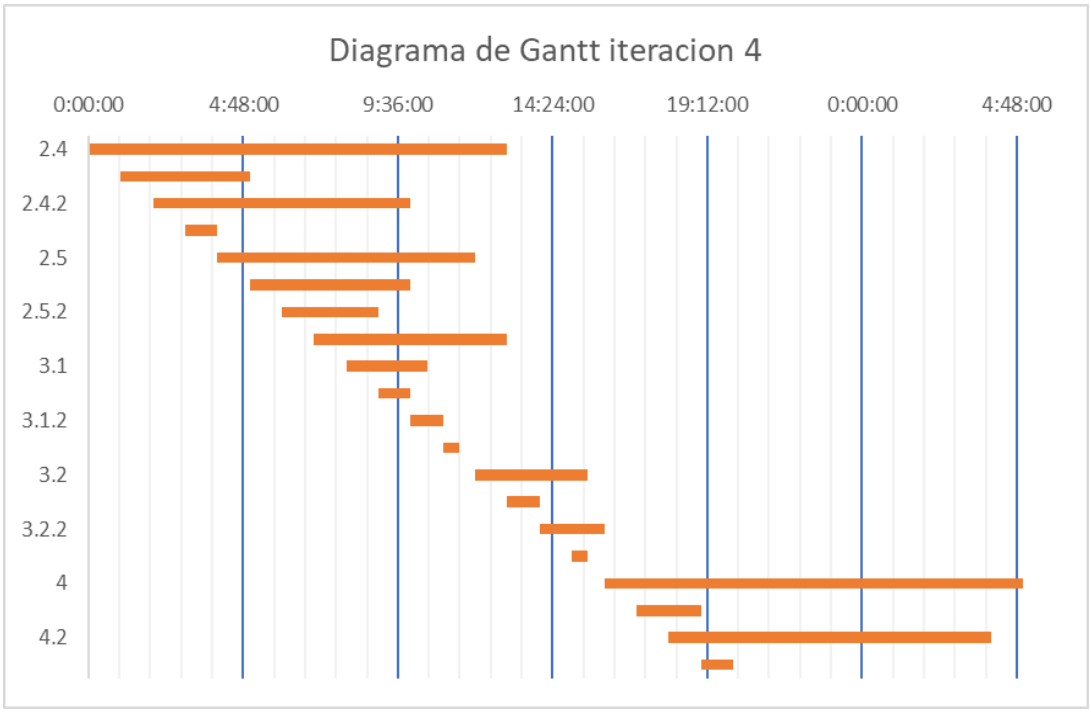


Figura 12-4. Diagrama de Gantt iteración 4

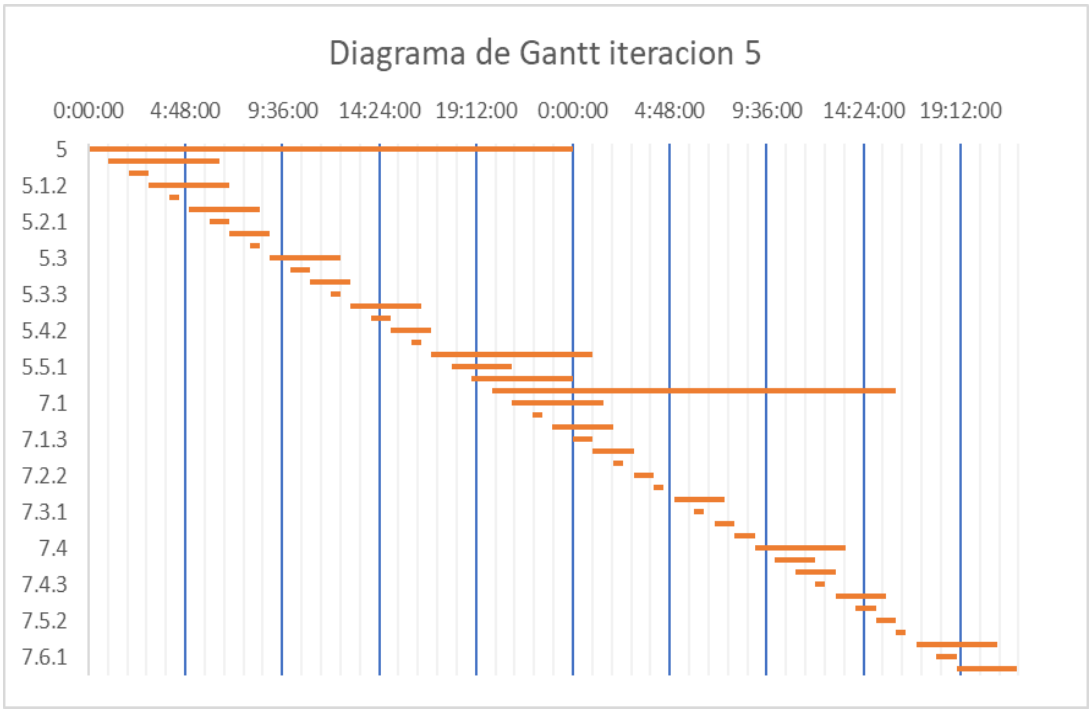


Figura 12-5. Diagrama de Gantt iteración 5

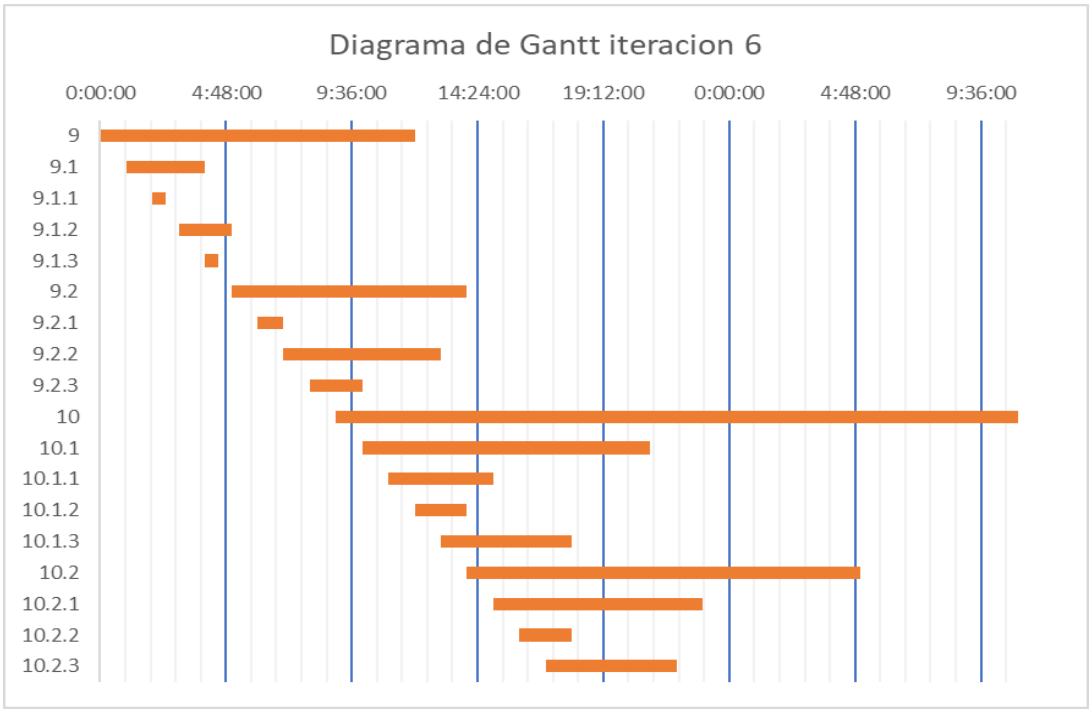


Figura 12-6. Diagrama de Gantt iteración 6

13. EJECUCIÓN DEL PROYECTO

Una vez desarrollada toda la fase de planificación me dispondré a completar todas las tareas mencionadas en cada iteración hasta finalizar el proyecto. Este apartado tratará de presentar un resumen de los pasos dados para llevar a cabo cada tarea y de los problemas que han ido surgiendo a lo largo de este, siguiendo el esquema mencionado de diseño, implementación y prueba.

13.1 Iteración 1 (del 30/11/20 al 14/12/20)

Esta iteración se desarrollará en 2 subgrupos del grupo de tarea 1 que engloba el control general de nuestro jugador. Veremos a continuación un resumen de como se ha trabajado en cada tarea.

13.1.1 Grupo de tareas 1: Sistema de control del jugador

13.1.1.1 Subgrupo de Tareas 1.1: Avatar del jugador

13.1.1.1.1 Tarea 1.1.1: Diseño

Como ya he comentado, el juego sucederá dentro de un ambiente de ciencia ficción en un futuro distópico por lo que nuestro personaje será un robot. Teniendo en cuenta que solo hay un trabajador para este proyecto, se ha optado por la opción de descargar el personaje con todas sus animaciones de la Unity Store.

Figura 13-1. Imagen del modelo

13.1.1.1.2 Tarea 1.1.1: Implementación

Para implementar el personaje en 2D pegamos el Sprite de este personaje en cualquiera de sus posiciones y le definimos todas las animaciones que puede realizar. Como ya mencioné, el motor contiene físicas que solo se implementarán cuando añadamos el objeto de rigidbody2d donde podremos dar valor a la magnitud de la gravedad, de la masa, de la posición… También le añadiremos el objeto de Collider2D en forma de cuadrado alrededor del personaje el cual detectará las colisiones del personaje con otros objetos que tengan este componente.

13.1.1.2 Subgrupo de Tareas 1.2: Movimiento del jugador

13.1.1.2.1 Tarea 1.2.1: Diseño

El movimiento del jugador se realizará mediante las teclas del teclado, el diseño de las animaciones como las de andar, correr, saltar, agacharse y atacar fueron descargadas anteriormente.

13.1.1.2.2 Tarea 1.2.2: Implementación

Para poder mover al personaje deberemos confirmar que este tiene el componente ridgibody2D y el collider2D. Una vez tenga estos componentes crearemos un par de scripts en C# y en mi caso me he ayudado de Visual Studio, aunque se puede usar cualquier entorno de desarrollo.

El primer Script será ‘CharacterController’ donde pondremos todas las variables que queramos controlar en general con respecto a todos los personajes tanto enemigos como nuestro robot. Después realizaremos el script ‘Playermovement’ donde programaremos toda la lógica del movimiento y de las animaciones.

Al pulsar sobre una tecla le añadiremos una velocidad sobre su Ridgibody2D con lo cual tendrá la posibilidad de moverse. Haremos lo mismo para todos los demás movimientos, aunque en el caso del salto hemos tenido en cuenta que si se mantiene la barra espaciadora pulsada el personaje saltará durante más tiempo. Cada vez que se realice una acción llamaremos a la animación pertinente.

El salto ha sido bastante complicado ya que he tenido que añadir a los pies del personaje un objeto invisible a la vista del usuario pero que contiene un collider el cual, cuando detecta colisión con alguna plataforma significa que este estará en el suelo y podrá saltar, si en cambio no detecta ninguna colisión no podrá saltar, así evitaremos que, si se presiona la barra espaciadora sin soltarla, nuestro personaje vuele sin control

13.1.1.2.3 Tarea 1.2.2: Pruebas

Realizamos varias pruebas viendo como actuaba las animaciones y como se movía el personaje. Se ajustó varias veces la velocidad de movimiento y de las animaciones para que todo fuera fluido. Al chocar con el extremo de algunas plataformas durante el salto me di cuenta que nuestro personaje se quedaba parado en ese sitio con lo cual para remediar al problema he tenido que añadir un material sin rozamiento a nuestro personaje para que pudiera deslizarse en vez de quedarse atrapado.

13.2 Iteración 2 (del 14/12/20 al 28/12/20)

En esta segunda iteración estudiaremos el último subgrupo del grupo de tareas 1 que se refiere al ataque del jugador y el primer subgrupo del grupo 2 de tareas referente a los tipos de enemigos que añadiremos en el juego.

13.2.1 Grupo de tareas 1: Sistema de control del jugador

13.2.1.1 Subgrupo de Tareas 1.3: Ataque del jugador

13.2.1.1.1 Tarea1.3.1: Diseño

Para el diseño de los ataques como recordarán lo mencionado anteriormente, no podremos crear los personajes objetos o animaciones por falta de tiempo así que en la Asset Store de Unity descargamos las armas que implementaremos a nuestro personaje como la guadaña o las balas ya descritas anteriormente.

13.2.1.1.2 Tarea1.3.2: Implementación

Para implementar los ataques de nuestro personaje en el mismo script de 'Playermovement' añadiremos el control de la animación de ataque cuando pulsemos ctrl izquierdo.

Modificaremos la bala y la convertiremos en un prefab para así poder instanciarla por pantalla en cualquier momento mediante nuestro script, pulsando la tecla x de nuestro teclado.

Las dos armas tendrán el componente collider y en el script llamaremos la función Ontriggerenter2D que detecta colisiones de collider con otros collider comparando un tag o nombre que se le ponga, siendo el desarrollador quien tome la libertad de elegir el nombre a su antojo. En nuestro proyecto tanto las balas como la guadaña tendrán el tag de 'weapon'.

La guadaña no se instanciará porque no se convierte en prefab por lo que simplemente será un objeto hijo del personaje no visible que se revelará cuando se llame la animación de ataque para desaparecer más tarde otra vez. En cuanto a la bala se destruirá cuando detecte una colisión.

13.2.1.1.3 Tarea1.3.3: Pruebas

En este apartado nuestro objetivo era que la guadaña apareciera justo en el momento oportuno al igual que la bala y que así los movimientos se coordinasen. Aquí el problema consistía en que el collider de la guadaña se activaba justo al pulsar el botón de ataque y en mi caso quería que se activase justo cuando la guadaña estuviera en la posición de pegar por lo que tuve que crear un evento dentro de la animación de la guadaña para llamar a una función que activará el collider2D justo en el momento preciso.

13.2.2 Grupo de tareas 2: Sistema de control enemigo

13.2.2.1 Subgrupo de Tareas 2.1: Tipos de enemigos

13.2.2.1.1 Tarea 2.1.1: Diseño

Como en todos los diseños anteriores he tenido que investigar en la tienda de Unity los tipos de enemigos que quería añadir. Mi propósito era localizar unos enemigos que encajaran bien con la calidad de imagen del personaje principal y con la temática general del juego. Al final encontré los dos mostrados a continuación con las animaciones que necesitaba y las balas para usar como proyectil de sus armas.

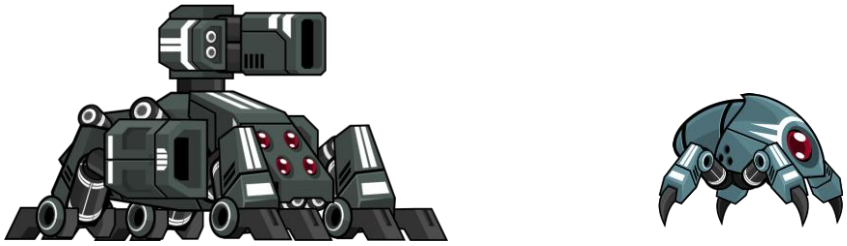


Figura 13-2. Imagen de los 2 enemigos añadidos al juego

13.2.2.1.2 Tarea 2.1.2: Implementación

Este apartado será muy parecido al de implementación del avatar del robot. Pegaremos los Sprites de los enemigos en la escena y le añadiremos todas las animaciones al animador de cada uno. Aparte, también le añadiremos el componente de Ridgibody2D y de Collider2D

13.2.2.1.3 Tarea 2.1.3: Pruebas

Para comprobar el correcto funcionamiento de las animaciones y de que cumplían con los requisitos realizamos una prueba con los dos tipos de enemigos. Una vez visto que cumplían con su cometido pasamos a la siguiente tarea.

13.3 Iteración 3 (del 28/12/20 al 11/01/21)

En estas dos semanas trabajaremos sobre las mecánicas de movimiento de los dos enemigos y las mecánicas de ataque, intentaremos hacerlas variadas y fluidas para que el usuario no conozca de antemano los patrones de los enemigos.

13.3.1 Grupo de tareas 2: Sistema de control enemigo

13.3.1.1 Subgrupo de Tareas 2.2: Mecánica de movimiento enemiga

13.3.1.1.1 Tareas 2.2.1: Diseño

Para el diseño del movimiento tendremos que coger la animación que consiste en andar, morir y estar parado para el enemigo ‘mite’ y para el otro enemigo deberemos tener en cuenta las animaciones como disparar, morir, andar y estar parado.

Para estos enemigos, no existía animación de serie de ser golpeado por lo que a partir de las animaciones de morir y con la ayuda de la herramienta ‘Animator’ de Unity, he creado las animaciones que consisten en ser herido cogiendo los fotogramas que me parecían interesantes.

13.3.1.1.2 Tareas 2.2.2: Implementación

Para estos dos enemigos, he creado un script llamado ‘enemy’ que contendrá todas las variables características de un enemigo y a partir de este script que tendrán todos los enemigos, he compuesto el script llamado ‘enemymovement’ que contendrá el algoritmo de movimiento de los enemigos. He realizado 3 tipos de movimiento los cuales enunciaré a continuación:

- ‘Static’: En este modo, el enemigo estará totalmente quieto y tendrá activada siempre la animación de estar parado.
- ‘Walker’: En este modo el enemigo tendrá activada siempre la animación de andar y le he definido dos objetos hijos invisibles, que indicarán si hay un collider en frente, es decir una pared o si el collider donde está apoyado se termina, en otras palabras, si hay un precipicio. De esta manera el enemigo al detectar un precipicio o una pared, llamará a la función ‘Flip’ que cambia la escala en x del enemigo es decir le da la vuelta en el eje x.
- ‘Patrol’: Para este caso definiremos también dos objetos hijos del enemigo, punto A y punto B, el enemigo ira andando de un punto a otro.

13.3.1.1.3 Tareas 2.2.3: Pruebas

Para ver su correcto funcionamiento junto con el personaje principal he construido un mapa pequeño de demostración que usaremos a partir de ahora con varias plataformas con sus respectivos Collider. Al realizar la prueba de los enemigos con el personaje, surgió el problema siguiente; los enemigos recibían un empujón demasiado grande de parte del jugador por lo que aumenté sus masas y para evitar colisiones entre los propios enemigos los puse en diferentes capas.

Figura 13-3. Enemigo en el mapa de Demo interactuando correctamente

13.3.1.2 Subgrupo de Tareas 2.3: Mecánica de ataque enemiga

13.3.1.2.1 Tareas 2.3.1: Diseño

Para el diseño de los diferentes ataques. tal y como hicimos con los movimientos crearemos las animaciones de disparo con la herramienta ‘Animator’ gracias a los sprites descargados con estos enemigos. Incluiremos también las balas como armas en el diseño de ataque

13.3.1.2.2 Tareas 2.3.2: Implementación

En este caso haremos dos tipos de ataque que podremos implementar en los enemigos que queramos. Para mi proyecto he decidido que el enemigo ‘mite’ no realice ningún tipo de ataque.

Primero, he realizado un script llamado ‘playerhealth’ donde usando la función ya mencionada antes Ontriggerenter2D comprobaremos si ha colisionado con un enemigo, si es así le restaremos vida a la vida total de nuestro personaje. Para los enemigos haremos un script parecido que se llamara ‘enemyhealth’ pero esta vez cal comprobar si la colisión es con arma se le restará vida al enemigo.

Una vez realizado esto, programaremos en un script que se llamará ‘enemyproyectil’ el lanzamiento de balas del único enemigo atacante llamado ‘gunner’. Antes deberemos convertir las balas de los enemigos en prefab para poder instanciarla en la pantalla, a estas les pondremos el tag de ‘enemy’ y un collider2D para que cuando colisione contra nuestro personaje, este reciba el daño.

El enemigo tendrá dos tipos de ataques:

- ‘freqshooter’: Nombraremos a una función llamada shoot, que instanciará la bala con cierta velocidad y activará la animación de disparar, cada cierto tiempo
- ‘Watcher’: En este caso crearemos otro collider invisible hijo del enemigo en forma de rectángulo alargado en el eje, este lo pondremos como ‘trigger’, lo que significa que cuando colisione con otro collider no se aplicarán las físicas de colisión, sino que simplemente servirá para avisar de una colisión. De esta manera solo cuando el jugador choque con este collider el enemigo disparará.

Tanto el jugador como el enemigo al recibir un daño, se les aplicará una fuerza de empuje hacia atrás y hacia arriba para simular el golpe.

Cuando la salud de los enemigos es menor que 0 se activará la animación de morir y se destruirá el objeto en varios segundos

13.3.1.2.3 Tareas 2.3.3: Pruebas

Después de varias horas de pruebas para ver el correcto funcionamiento de los ataques y que la vida reducía correctamente en ambos casos pase a la siguiente tarea. Antes de continuar, me di cuenta que en el transcurso de las pruebas, cuando el enemigo moría, al seguir activo el collider, este todavía producía daño por lo que corregí el código y en el momento de morir desactivé el collider y congelé la posición para que no se cayese a través de las plataformas, de esta manera todo funcionaba correctamente.

Figura 13-4. Enemigo eliminado interactuando correctamente con el jugador

13.4 Iteración 4 (del 11/01/21 al 25/01/21)

Para esta iteración explicaré el sistema de generación de ítems o objetos además de mencionar algunas pruebas junto a los testers y explicaré el diseño e implementación de la cámara y el sistema de experiencia.

13.4.1 Grupo de tareas 2: Sistema de control enemigo

13.4.1.1 Subgrupo de Tareas 2.4: Sistema de generación de ítems

13.4.1.1.1 Tarea 2.4.1 Diseño

Para el diseño de la generación de objetos utilizaremos los mismos objetos que estarán repartidos por el mapa descargados en la Asset store de unity. Dichos objetos se tendrán que establecer como prefabs.

13.4.1.1.2 Tarea 2.4.2 Implementación

En la implementación trataremos de instanciar los prefabs de los objetos cuando la vida de los enemigos esté por debajo de 0. En el script de 'enemyhealth' aplicaremos un algoritmo que instancie los objetos aleatoriamente con una probabilidad menor para los mejores objetos y una mayor para los menos importantes. Estos mismos objetos también estarán repartidos por todo el mapeado.

13.4.1.1.3 Tarea 2.4.3 Prueba

Comprobamos que con exactitud el enemigo, al ser eliminado, suelta los objetos con el porcentaje de aparición que le hayamos dado.

13.4.1.2 Subgrupo de Tareas 2.5: Pruebas

13.4.1.2.1 Tarea 2.5.1 Pruebas propias

Se ha probado varias veces para verificar que los enemigos pierden vida y el usuario también y se ha comprobado que no haya colisiones no deseadas.

13.4.1.2.2 Tarea 2.5.2 Pruebas de usuarios.

Estas pruebas son de lo mas importante a la hora de realizar un videojuego ya que mejorarán el acabado final del proyecto. A los usuarios se les hizo probar el juego durante varias horas y a medida que jugaban se les planteo varias preguntas por ejemplo si creían que la vida de los enemigos era excesiva, si creían que el rango de la guadaña era suficiente, si los enemigos hacían mucho o poco daño…etc. Con las respuestas de los usuarios se adaptó el videojuego para los gustos de los mismos.

13.4.2 Grupo de tareas 3: Sistema de cámara

13.4.2.1 Subgrupo de Tareas 3.1: Seguimiento del jugador

13.4.2.1.1 Tarea 3.1.1 Diseño

La cámara viene ya incluida con Unity, pero he creado un collider2D que ocupará el mapa para definir los bordes de la cámara.

13.4.2.1.2 Tarea 3.1.2 Implementación

He realizado un script que guardará la posición del jugador continuamente y definirá a la cámara con esa posición cada frame. En el mismo script he programado que si la cámara se acerca al borde no continúe de tal manera que no se vea algo a lo que nunca podremos acceder.

13.4.2.1.3 Tarea 3.1.3 Pruebas

Para este apartado lo único que tuve que realizar ha sido probar que la cámara siguiera correctamente al jugador y los fps se mantuvieran estables. También se ha debido ajustar el collider de los limites para no salir del rango del mapa.

13.4.2.2 Subgrupo de Tareas 3.2: Fluidez de la cámara

13.4.2.2.1 Tarea 3.2.1 Diseño

Para controlar la fluidez de la cámara he diseñado un rango de valores para los que esta misma seguirá al jugador con más o menos lentitud.

13.4.2.2.2 Tarea 3.2.2 Implementación

En el mismo Script de la cámara he implementado esos rangos de valores y los he puesto como variables publicas para poder controlarlo de la interfaz de Unity. Esta tarea consistía en establecer un retraso en la cámara para que cuando se empiece a mover el personaje no lo siga automáticamente al instante, lo mismo pasará cuando se pare el personaje. Esto dará una mejor sensación a la vista del usuario.

13.4.2.2.3 Tarea 3.2.3 Pruebas

Después de varias pruebas conseguí ajustar el rango de valores para que la cámara no perdiese del todo al personaje, pero sigue teniendo ese retraso

Figura 13-5. Imagen que muestra el limite inferior de la cámara

13.4.3 Grupo de tareas 4: Sistema de experiencia

13.4.3.1 Tareas 4.1 Diseño

Un nuevo nivel otorgará un aumento de características y para subir de nivel se necesitará experiencia que vendrá causada por la eliminación de los enemigos. Cada nivel necesitara más experiencia que el anterior.

13.4.3.2 Tareas 4.2 Implementación

Se ha añadido al jugador un script que se llamará 'ExperienceScript' donde se programará que tipos de atributos y cuanto aumentarán al subir de nivel, cuanta experiencia requiere el próximo nivel y añadirá experiencia cuando la vida de un enemigo sea menor o igual a 0.

Para añadir la experiencia en el script de 'enemy' creamos una nueva variable pública para poder elegir su valor, la cual será la cantidad de experiencia que recibamos según el enemigo.

Llegados a este punto podemos cambiar los enemigos a prefabs para que así una vez que hagamos cambios en uno se apliquen a todos los demás enemigos del mismo tipo.

13.4.3.3 Tareas 4.3 Pruebas

Se realizan pruebas con los enemigos para ver si otorgan la cantidad de experiencia deseada y si la experiencia necesaria para subir de nivel aumenta debidamente.

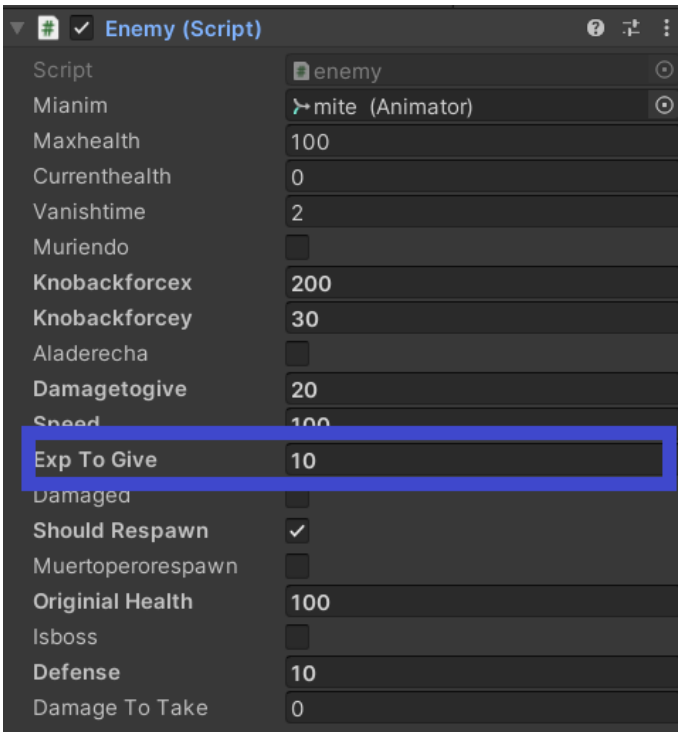


Figura 13-6. Imagen que muestra la variable modificable de la experiencia que da mite

13.5 Iteración 5 (del 25/01/21 al 08/02/21)

En esta iteración trabajaremos con los grupos de tareas 5 y 7 que son los grupos respectivos a la interfaz de usuario y los efectos visuales. Para este apartado evitaré poner imágenes para no resultar redundante ya que las imágenes referentes a estas tareas ya aparecen en el apartado 8.

13.5.1 Grupo de tareas 5: Interfaz de usuario

13.5.1.1 Subgrupo de Tareas 5.1: Barra de vida y nivel

13.5.1.1.1 Tarea 5.1.1 Diseño

Para este apartado me he ayudado de la herramienta Photoshop para crear algunas imágenes que añadir, como en los anteriores apartados, también he recurrido a la tienda de Unity para algunos fondos animados.

13.5.1.1.2 Tarea 5.1.2 Implementación

Para poder llevar a cabo la implementación de la interfaz de usuario en Unity deberemos de crear un nuevo objeto llamado ‘Canvas’. Es la escena donde aparecerán todos los gráficos. Dentro de este objeto añadiremos dos imágenes en forma de barra y le añadiremos el componente fill, que hará que se llenen con una variable que controlaremos. En nuestro caso esta variable será la vida y la experiencia de nuestro personaje, además dentro de los scripts que controlan estas dos características deberemos añadir la librería de la interfaz de usuario y con una función que llamaremos en cada fotograma vaciando la barra cuando nos golpeen o llenándola cuando cojamos una poción.

Al lado de estas dos barras implementaremos la imagen de nuestro avatar para proporcionar mejor estética a nuestro videojuego

13.5.1.1.3 Tarea 5.1.3 Prueba

La barra de experiencia y salud se han comprobado, al verificar si se ha llenado o vaciado correctamente, También se contempló la opción de, en vez de poner una imagen estática de nuestro avatar, poner una imagen cercana de nuestro avatar moviéndose en tiempo real, pero los resultados no fueron los deseados y decidí eliminarlo. Además, hicimos varias pruebas para confirmar que cuando se subía de nivel la vida máxima aumentaba

13.5.1.2 Subgrupo de Tareas 5.2: Conteo de objetos

13.5.1.2.1 Tarea 5.2.1 Diseño

El diseño del conteo de objetos será también creado en parte en Photoshop para después llevarlo como imagen tipo PNG dentro de Unity.

13.5.1.2.2 Tarea 5.2.2 Implementación

Una vez creado el objeto de Canvas dentro de Unity será sencillo implementar los demás elementos dentro de la interfaz de usuario. Esta vez en la esquina superior derecha implementaremos la imagen del conteo de objetos, que se irá actualizando gracias al script utilizado para recoger los objetos, consistirá simplemente en un texto con el número de monedas o munición que tenemos en el momento.

13.5.1.2.3 Tarea 5.2.3 Pruebas

Comprobamos si a la hora de recoger los objetos estos iban aumentando en el interfaz y a su vez añadimos el valor máximo de munición que podíamos tener y controlamos si se iba actualizando en el momento que subíamos de nivel.

13.5.1.3 Subgrupo de Tareas 5.3: Menú y submenús

13.5.1.3.1 Tarea 5.3.1 Diseño

Para este apartado como en los anteriores nos hemos ayudado de Photoshop para hacer un menú de pausa que aparecerá al haber pulsado la tecla ESC en el teclado. Este tendrá varios botones y opciones los cuales abrirán otros submenús dependiendo del botón que pulsamos.

13.5.1.3.2 Tarea 5.3.2 Implementación

Este menú estará siempre en la interfaz de usuario, pero al estar desactivado, mediante un script llamado ‘MenuPause’ activaremos el menú de pausa al pulsar la tecla ESC. Los diferentes botones se programarán de la siguiente manera:

Cuando implementamos un botón, Unity nos pide que le asociemos una función, por lo que, en el script mencionado anteriormente, crearemos una función para cada tipo de botón. Las funciones activarán el menú de ajustes, saldrán de la pantalla del juego, reanudarán el juego desde el ultimo punto guardado o saldrán del menú de pausa.

Cabe destacar que el tiempo en el juego quedará en pausa el tiempo que este activo el menú de pausa por lo que no nos tendremos que preocupar que nos ataquen mientras estamos ajustando el volumen del sonido en el submenú de ajustes.

13.5.1.3.3 Tarea 5.3.3 Pruebas

Mediante las pruebas comprobaremos que no hay ningún fallo a la hora de abrir los diferentes menús y que el tiempo queda parado mientras estamos en los menús. Realizando pruebas tuve un fallo persistente ya que no me paraba el tiempo mientras estaba en el menú de pausa, la función para parar el tiempo es Time.Timescale=0. Dicha función es bastante compleja y resultaba que otro script estaba actualizando el valor a 1 sin que me diera cuenta. Hay que tener mucho cuidado a la hora de utilizar esta función ya que si no lo hacemos correctamente nos puede causar muchos problemas.

13.5.1.4 Subgrupo de Tareas 5.4: Submenú de gráficos

Se ha excluido este apartado por falta de tiempo para el proyecto.

13.5.1.5 Subgrupo de Tareas 5.5: Pruebas

13.5.1.5.1 Tarea 5.5.1 Pruebas propia

Mediante las diferentes pruebas he logrado comprobar que todo funcionaba correctamente, la barra de vida se llenaba o se vaciaba según la acción pertinente, al igual que la barra de experiencia. Los objetos se contaban y actualizaban correctamente, según se subía de nivel y las características aumentadas se veían reflejadas. Además. todos los menús y submenús funcionaban perfectamente dejando parado el tiempo mientras estaban abiertos.

13.5.1.5.2 Tarea 5.5.2 Pruebas de usuarios

Se ha dado a los usuarios el mapa de demostración para que pudiesen comprobar que todo funcionaba correctamente y aportasen consejos pare mejorar el juego. Durante el periodo, a estos usuarios, se les preguntó si creían que la interfaz era demasiado invasiva, si les resulto demasiado difícil subir de nivel, si les resultaba atractiva la interfaz de usuario, si les resultaba útil la información dada o si les faltaba alguna…

Al recoger todas sus respuestas se pudo mejorar la interfaz de usuario y el sistema de experiencia.

13.5.2 Grupo de tareas 7: Efectos visuales

13.5.2.1 Subgrupo de Tareas 7.1: Perdidas de daño

13.5.2.1.1 Tarea 7.1.1 Diseño

Para el diseño de las pérdidas de vida crearemos con la herramienta de Unity de texto un texto en rojo llamativo que aparecerá encima del enemigo al recibir daño.

13.5.2.1.2 Tarea 7.1.2 Implementación

El apartado de implementación de esta tarea fue explicado anteriormente, para sacar el texto flotante por pantalla lo estableceremos como un prefab y así se podrá instanciar cada vez que se le aplique daño a un enemigo.

Pero lo más importante de este apartado es que se cambiará a partir de ahora la variable de ataque dependiendo en este momento de la variable de ataque del arma y la defensa del enemigo. A partir de ese momento, a los enemigos se le aplicará un daño aleatorio dentro de un rango predefinido dependiendo de las tres características de ataque

Cuando el valor de ataque exceda cierto rango aparecerá en pantalla como si le hubiésemos hecho un daño critico al enemigo.

13.5.2.1.3 Tarea 7.1.3 Prueba

Comprobaremos que el texto flotante se vea claro y conciso pero que no abarque toda la pantalla. Comprobaremos si es correcta la probabilidad de recibir un ataque criticó. En este apartado intenté que el texto hiciese una animación hacia arriba saliendo del enemigo, pero me fue imposible implementarla así que decidí dejar que el texto saliese arriba de los enemigos.

13.5.2.2 Subgrupo de Tareas 7.2: Transición entre escenas

13.5.2.2.1 Tarea 7.2.1 Diseño

Con ayuda de la tienda de Unity y Photoshop he realizado una escena de transición entre una escena y otra que queda de la siguiente manera

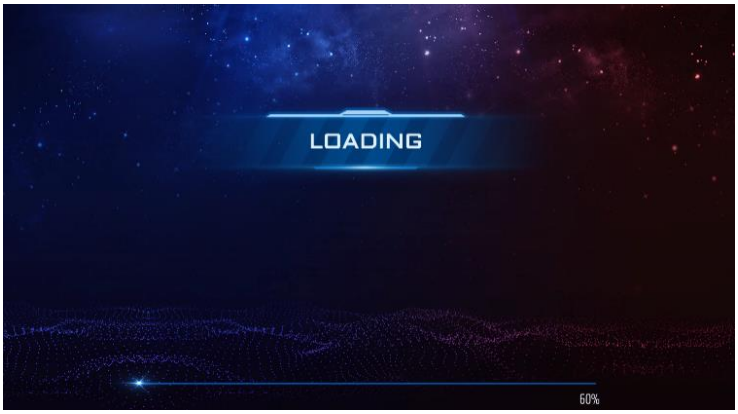


Figura 13-7. Imagen de muestra para la transición entre escenas.

13.5.2.2.2 Tarea 7.2.2 Implementación

Por falta de tiempo y ya que el juego no es muy pesado y las transiciones entre escenas son muy fluidas se ha decidido no implementar pantalla de carga entre estas.

13.5.2.2.3 Tarea 7.2.3 Pruebas

Se ha probado para que quedase correctamente con el aspecto de los menús y el ambiente general del videojuego, pero no se ha probado en tiempo real porque se decidió no implementarlo.

13.5.2.3 Subgrupo de Tareas 7.3: Subida de nivel

13.5.2.3.1 Tarea 7.3.1 Diseño

Al subir de nivel aparecerá un mensaje por pantalla que nos indicará que hemos subido de nivel

13.5.2.3.2 Tarea 7.3.2 Implementación

Este efecto visual se lo añadiremos como si fuese parte de la interfaz de usuario todo el tiempo desactivado, solo se lo activaremos durante un breve periodo de tiempo y solo cuando nuestra experiencia alcance la cantidad necesaria para subir de nivel.

13.5.2.3.3 Tarea 7.3.3 Pruebas

Para comprobar el correcto funcionamiento se subirá de nivel unas cuantas veces y veremos si el mensaje aparece en la pantalla y desaparece tras el tiempo que hemos programado.

13.5.2.4 Subgrupo de Tareas 7.4: Guardado

13.5.2.4.1 Tarea 7.4.1 Diseño

Para este efecto en concreto hemos usado una herramienta ya implementada en Unity llamada ‘Particle effect’ donde se pueden ajustar varias opciones para transmitir el efecto de partículas deseado.

13.5.2.4.2 Tarea 7.4.2 Implementación

Para llevar a cabo este apartado primero pasaremos el efecto antes creado a prefab para poder instanciarlo cuando accedamos a la parte de guardado que se definirá más tarde.

13.5.2.4.3 Tarea 7.4.3 Pruebas

Cuando accedemos a la zona de guardado al colisionar con el collider que activa el guardado deberemos ver aparecer el efecto de partículas creado.

13.5.2.5 Subgrupo de Tareas 7.5: Efectos al disparar

13.5.2.5.1 Tarea 7.5.1 Diseño

En el pack descargado de Unity del personaje principal existe ya un prefab de efecto al disparar que utilizaremos para el disparo del personaje y para el de los enemigos. Dentro de estos packs obtenemos también el efecto de explosión de la bala.

13.5.2.5.2 Tarea 7.5.2 Implementación

Dentro del script de disparo enemigo y disparo del robot instanciaremos el efecto de disparo en la posición donde esta el arma de tal manera que cuando salga la bala se apreciará un efecto de chispas imitando un disparo.

Para la colisión de las balas crearemos un script para la bala del personaje y la bala enemiga que consistirá en que cuando detecte un collider con el tag de enemigo o jugador se instanciará el efecto y el objeto se destruirá, resultando un efecto visual mejorado al ser alcanzado por una bala.

13.5.2.5.3 Tarea 7.5.3 Pruebas

Se hará varias pruebas con las que veremos si el efecto de la bala cuando colisiona es el correcto. Después de varias pruebas observé que el efecto de disparo estaba un poco adelantado con respecto a la animación de disparo por lo que antes de instanciar el efecto de disparo tuve que activar una corrutina para esperar cierto tiempo.

13.5.2.6 Subgrupo de Tareas 7.6: Pruebas

13.5.2.6.1 Tarea 7.6.1 Pruebas propias

En el mapa de demo se ha comprobado que todos los efectos funcionasen correctamente, el efecto al subir de nivel, se ha comprobado que los textos estuviesen bien colocados para todo tipo de enemigos y que al entrar en la zona de guardado el efecto de guardado se viese real.

13.5.2.6.2 Tarea 7.6.2 Pruebas de usuarios

Para que los usuarios puedan probar los efectos correctamente se ha decidido crear un ejecutable en Windows (x64) de la demo y se les ha ido preguntado como en las anteriores pruebas. Los usuarios han respondido a preguntas tales como, le parecen bonitos estéticamente los efectos en general, le resultan útiles los diferentes menús, mejoraría de alguna manera los menús o submenús… Todas las respuestas fueron tomadas en cuenta para seguir progresando con el proyecto

13.6 Iteración 6 (del 08/02/21 al 22/02/21)

Para la última iteración trabajaremos en los dos últimos grupos de tareas el sistema de guardado y el sistema de sonido para el cual hemos recibido ayuda de mi compañero David Varo Gomez, ingeniero de sonido.

13.6.1 Grupo de tareas 9: Sistema de guardado

13.6.1.1 Subgrupo de Tareas 9.1: Construcción zona de guardado

13.6.1.1.1 Tarea 9.1.1 Diseño

Gracias al 'Tile Palette 'de Unity, una herramienta muy útil para pintar plataformas 2D en el mapa con collider incluido, hemos creado una zona de guardado que tendrá un Collider en el centro muy pequeño.

13.6.1.1.2 Tarea 9.1.2 Implementación

Con la creación del Collider crearemos un script llamado 'Data Manager' el cual una vez detecte la colisión con el jugador guardará las monedas que tengamos en el momento y el nivel con las características que conlleva la subida de este.

13.6.1.1.3 Tarea 9.1.3 Prueba

Hasta que no se implemente el guardado de objetos no se sabrá si se guarda realmente, pero para comprobar que la zona funciona, imprimimos por la consola un mensaje al entrar en contacto con esta. Tras 3 pruebas al llegar a la zona los efectos de guardado antes mencionados aparecen y se imprime el mensaje por la consola.

13.6.1.2 Subgrupo de Tareas 9.2: Guardado de características

13.6.1.2.1 Tarea 9.2.1 Diseño

Diseñaremos un sistema de guardado usando los 'player prefs' que son objetos dentro del juego que nunca se destruirán, aunque salgamos del juego

13.6.1.2.2 Tarea 9.2.2 Implementación

Como ya expliqué en el diseño utilizaremos los 'player prefs' que nunca se destruirán y de esta manera haremos estático el script 'Data Manager' para poder instanciarlo en cualquier momento. En este mismo, crearemos funciones para cada característica que nos interese guardar. Una vez que entremos en la zona de guardado llamaremos a esa función que guardará la característica en el player prefs y al iniciarse siempre lo hará la variable con el valor de player prefs guardado, así, cada vez que iniciemos el juego apareceremos con los últimos datos guardados.

13.6.1.2.3 Tarea 9.2.3 Prueba

Para comprobar el correcto funcionamiento, he tenido que hacer varias pruebas ya que son muchas cosas las que he querido guardar y para ello, tuve que comprobar varias veces que al salir y al entrar al juego se guardaba. También me preocupe en verificar que no pasará lo contrario, es decir cuando un jugador saliese sin guardar y volviese a entrar más tarde, no hubiese perdido todo lo no guardado.

13.6.2 Grupo de tareas 10: Sistema de sonido

13.6.2.1 Subgrupo de Tareas 10.1: Efectos de sonido

13.6.2.1.1 Tarea 10.1.1 Diseño

Para el diseño de efectos de sonido hemos descargado un paquete de efectos tipo ciencia ficción en la tienda de Unity

13.6.2.1.2 Tarea 10.1.2 Implementación

En Unity añadiremos un nuevo objeto llamado ‘AudioSource’. En este objeto crearemos por ahora una pista de efectos y dentro de la pista de efectos iremos añadiendo cada efecto que queramos reproducir. Después, realizaremos un script llamado ‘AudioManager’ el cual tendrá una función para cada efecto que consistiría en reproducirlo simplemente. Luego con una instancia reproduciremos cada efecto llamando a la función oportuna para cada acción en su respectivo script.

13.6.2.1.3 Tarea 10.1.3 Pruebas

Para el funcionamiento de los efectos comprobaremos cada uno de ellos al ponerlos en práctica en el juego y verificaremos si están coordinados con los movimientos.

13.6.2.2 Subgrupo de Tareas 10.2: Música ambiental

13.6.2.2.1 Tarea 10.2.1 Diseño

El diseño de la música ambiental ha sido realizado por David Varo Gomez con ayuda del software cubase 5 para la masterización y con la ayuda de 4 programas para los instrumentos virtuales. Se establecerán tres músicas ambientales en total: una durante el juego normal, otra para el jefe de la demo y otra para el menú de pausa

13.6.2.2.2 Tarea 10.2.2 Implementación

La implementación será igual que la de los efectos, pero esta vez daremos a la música ambiental la característica de ‘Loop’ es decir, una vez que termine, volverá a sonar para que no dejemos de escucharla mientras estemos jugando. Dentro del ‘AudioSource’ crearemos una nueva pista de música donde se conservarán las 3 canciones que se instanciarán al cargar algunas de las escenas pertinentes.

13.6.2.2.3 Tarea 10.2.3 Pruebas

Para las pruebas de este apartado deberé incluir las pruebas también realizadas con David Varo para elegir el tipo de música ambiental que mejor le conviniese a este proyecto. Después de varias pruebas y elegir la mejor música, comprobé que, al entrar en el menú principal, en el juego o en la escena del boss la música sonase clara y se repitiese de forma fluida.

13.7 Seguimiento y control del proyecto

Este proyecto contiene un total de 10 grupos de tareas divididos como ya hemos visto entre las 6 iteraciones mencionadas. Esta forma de organización permite un control mayor sobre los retrasos. Ahora se podrá ver una comparativa y desviación entre tiempo real y estimado.

TABLA DE ITERACIONES				
ITERACIONES	TAREAS	FECHA INICIO	DURACION ESTIMADA	DURACION REAL
1	1.1, 1.2	30/11/20	42	51
2	1.3, 2.1	14/12/20	41	36,5
3	2.2, 2.3	28/12/20	51	62
4	2.4, 2.5, 3, 4	11/01/21	40	36
5	5, 7	25/01/21	44	39,5
6	9,10	08/02/21	38	40
		TOTAL:	256	265

Tabla 13-1. Duración estimada y real de iteraciones

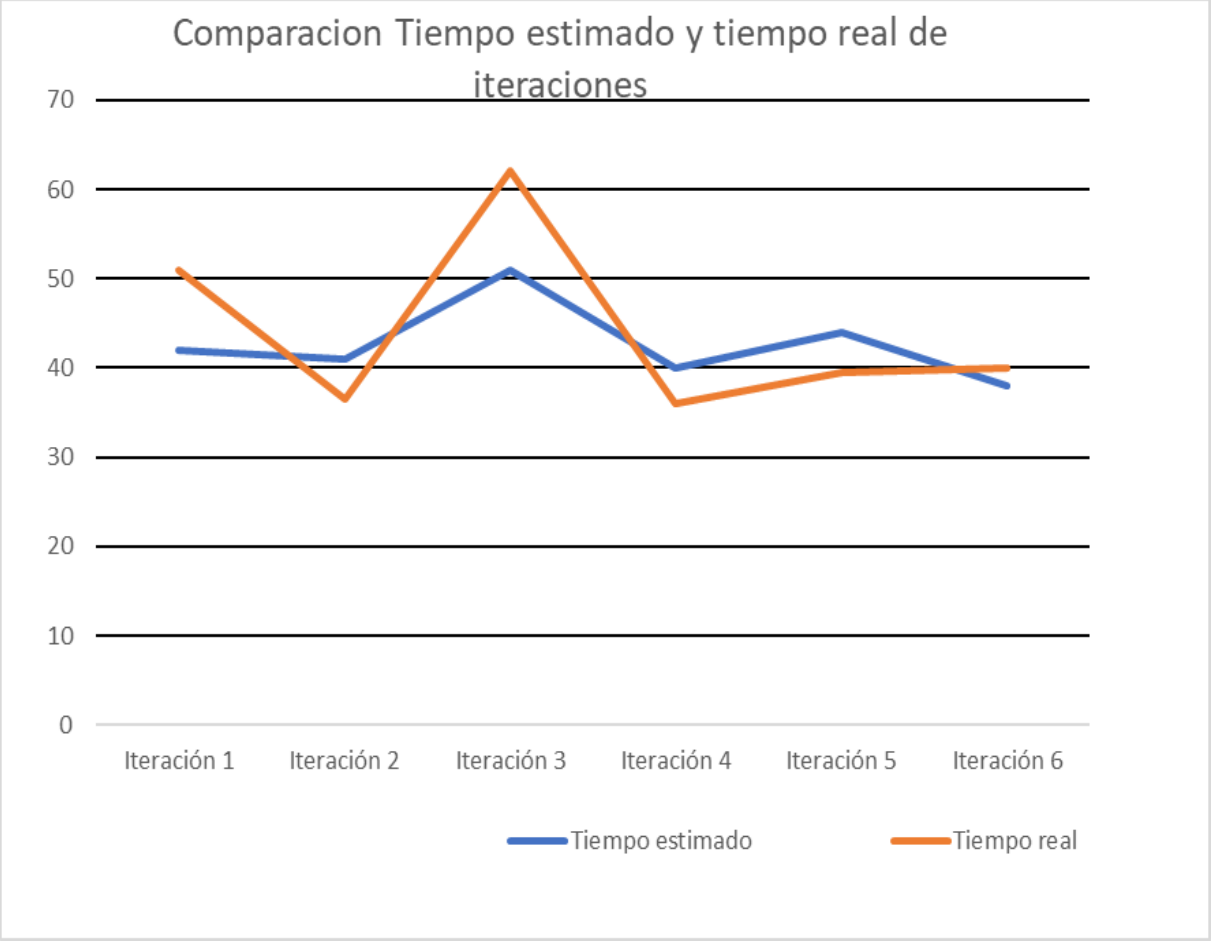


Figura 13-8. Diagrama de comparación de tiempo estimado y real por iteraciones

Como se puede observar en la gráfica no hay una diferencia muy grande entre el tiempo real y el estimado excepto en las iteraciones 3 y 1 ya que, al no tener experiencia en el entorno de Unity, al principio, ha sido difícil familiarizarme con la mecánica de programación sobre todo para los dos enemigos. Por lo tanto, al principio fue bastante complicado, pero luego logré acercarme al tiempo estimado.

Estimación Resumida por grupo de tareas		
Grupo de Tareas	Tiempo Estimado (Horas)	Tiempo Real (Horas)
1	73	77
2	82	93
3	6	5,5
4	13	10
5	24	21
7	20	18,5
9	12	14
10	26	26
TOTAL:	383	265

Tabla 13-2. Duración estimada y real de grupos de tareas

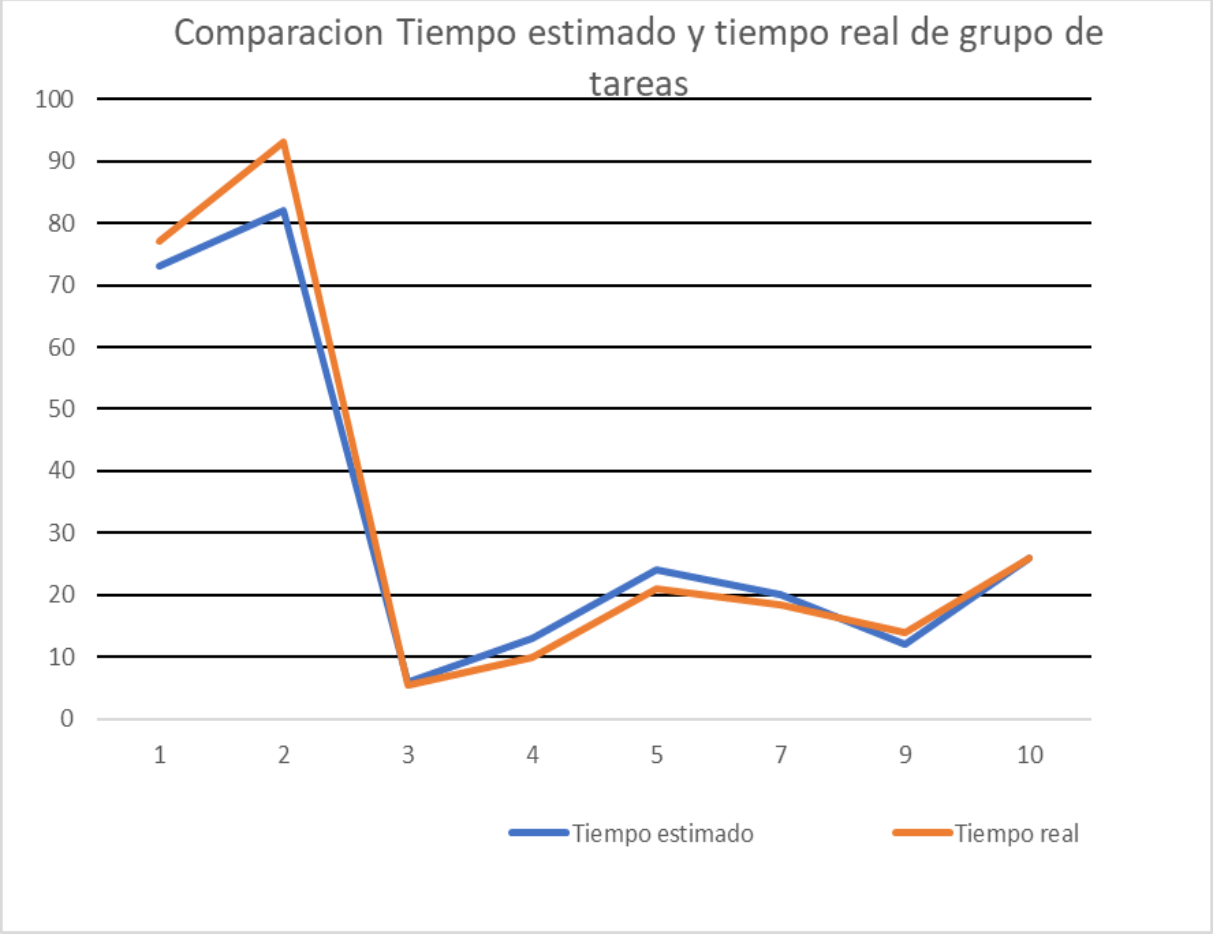


Figura 13-9. Diagrama de comparación de tiempo estimado y real por grupo de tareas

Como se puede observar la tarea 1 y 2 son las que mayor diferencia tienen, como ya he dicho el periodo de aprendizaje y decisión al principio fue muy lento pero una vez hice mío lo básico, Unity se vuelve muy intuitivo tanto para programar como para crear.

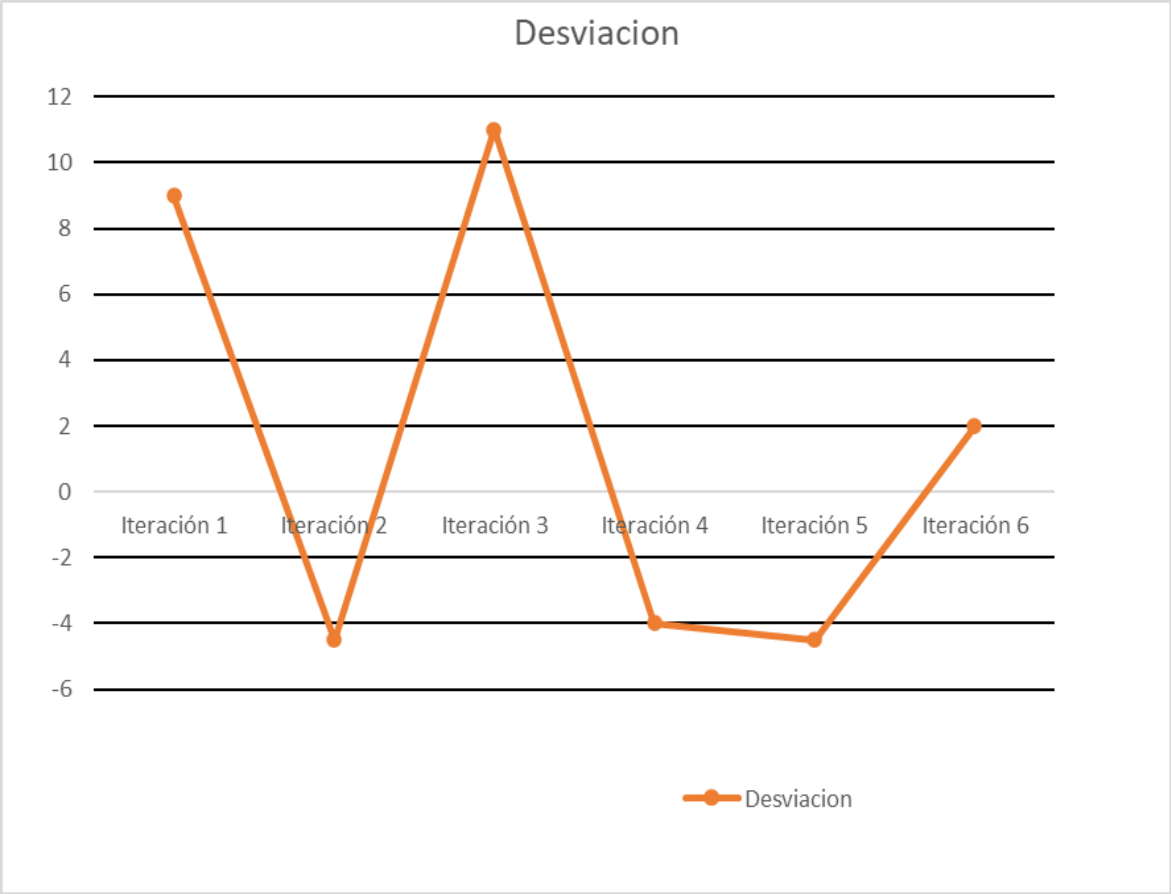


Figura 13-10. Diagrama de desviación de tiempo real y estimado

Las iteraciones 3,1 y 6 han tenido un retraso entre 2 y 10 horas que ya ha sido justificado con anterioridad. Esto se compensa con el adelanto de la iteración 2 4 y 5 de aproximadamente 4 horas cada una.

Para más detalle también se realizará una gráfica de la desviación para cada grupo de tarea

14. FINALIZACION DEL PROYECTO

Para finalizar el proyecto se generará un documento para el cliente y otro para el equipo de desarrolladores, el informe del cliente será una memoria del proyecto que se corresponde con el apartado 13 y un resumen de las funcionalidades correspondiente al apartado 7. El siguiente apartado es el informe dirigido al equipo de desarrolladores para una mejora en el futuro con proyectos similares:

14.1 Informe para el equipo de desarrolladores.

- Se ha seguido el esquema de diseño, implementación y prueba.
- Todas las funcionalidades fueron probadas individualmente antes de llevar una prueba del sistema completo.
- Se realizaron pruebas del proyecto con personas ajenas a este.
- Se usaron modelados gratis debido a la incapacidad de modelar objetos 2D.
- Cada día, se realizaron copias de seguridad en un disco duro externo.
- Se manejaron numerosas fuentes de información para investigar la implementación y mejora de las características.

14.1.2 Fortaleza y debilidades manifestadas

Las fortalezas mostradas en este proyecto son las siguientes:

- Capacidad para resolver problemas antes situaciones totalmente nuevas y desconocidas.
- Rápida adaptación al uso de tecnología.
- Constancia en el proceso de desarrollo.
- Aplicación buena del ciclo de vida de la gestión del proyecto.
- Estimación bastante decente para ser el primer proyecto realizado de esta envergadura.
- Colaboración con personas externas al equipo para realización de pruebas.

Las debilidades manifestadas se quedan de la siguiente manera:

- Experiencia mínima o nula en tareas de modelado 2D.

14.1.2 Recomendaciones al equipo

- Para el próximo proyecto de este tipo sería beneficioso contar con algún experto en modelado de personajes.
- Seguir usando las técnicas que funcionaron reforzando los puntos fuertes.

15. POR HACER

Se contemplará las funcionalidades que podrían ser implementadas para que el prototipo sea un videojuego apto para su comercialización.

- Sustituir modelos 2D usados por modelos propios.
- Añadir al interfaz de usuario un mapa.
- Añadir recompensas al superar los jefes.
- Optar por un sistema de escalado de dificultad según el nivel del jugador.
- Incluir cinemáticas para ambientar al usuario.
- Crear un mapeado completo.
- Incluir contenido adicional como otros modos de juegos.
- Incorporar puzles en los niveles.
- Adjuntar mayor cantidad de enemigos.
- Optimizar el sistema para una funcionalidad más eficiente.
- Incluir videojuego en producto registrado por la propiedad.

16. ENLACE DE DESCARGA

Compartiré un enlace que llevará a una carpeta compartida en Google Drive donde se podrá encontrar: una carpeta con el ejecutable del juego, otra con el proyecto en Unity, la documentación requerida y un video demostrativo.

https://drive.google.com/drive/folders/13dNPrc5Mr3ucBjgfKSywujKrlOCMq-rh?usp=sharing

CONCLUSIONES

Este trabajo ha sido una gran oportunidad para poner en práctica todo el conocimiento adquirido en el grado de Ingeniería en Tecnologías Industriales referente a la gestión de proyectos y metodologías de desarrollo. Desde el primer momento, haber estado apoyado en técnicas ampliamente probadas y de validez contrastada, han propiciado al éxito en la gestión y desarrollo de este ambicioso proyecto. Se han alcanzado los objetivos dentro del plazo estimado y las pequeñas variaciones de tiempo han sido suplidas gestionando mejor el tiempo.

En mi opinión, el desarrollo software de este tipo de productos puede ser de los procesos más complejos a los que puede enfrentarse un gestor de proyectos. Esto es debido a que el desarrollo de un videojuego concentra muchas disciplinas diferentes en un mismo proyecto

Como un proyecto de estas dimensiones excedía los límites temporales del trabajo final de grado, se propusieron soluciones que han ayudado a acelerar algunas de las tareas críticas del proyecto. Usar modelos 2D gratuitos de terceros para la implementación de los agentes ha supuesto un ahorro de tiempo muy importante. Aun habiendo ahorrado tiempo, existe funcionalidades que han quedado fuera del proceso, como la implementación de un mapa completo. Estas características se recogen en el apartado 15.

Una parte importante de los esfuerzos han sido dirigidos a obtener un prototipo que resulte atractivo y amigable al usuario final. Se ha tenido la inestimable ayuda de un grupo de conocidos que han realizado las pruebas de usuario del producto en las distintas fases del proyecto. Su contribución a este trabajo ha sido muy importante. También he dedicado más tiempo a la interfaz y la jugabilidad (sistema de combate), pues creo que son elementos clave en el éxito de la mayoría de los videojuegos.

La participación en este proyecto ha supuesto un enriquecimiento a nivel de conocimientos muy importante. Sin duda ha sido una buena experiencia, que ha puesto a prueba mi capacidad de disciplina y auto planificación, así como la habilidad para tomar conciencia de las partes críticas propias del proceso de planificación y desarrollo de un producto software complejo.

En un futuro no descarto la idea de desarrollar algún videojuego para dispositivos móviles aprovechando la facilidad de distribución desde las plataformas Google App Store y Apple App Store.

No teniendo experiencia previa en proyectos de esta naturaleza, finalizo este trabajo con una gran satisfacción y orgullo. He disfrutado mucho aprendiendo a utilizar numerosas herramientas nuevas y observando que paso a paso se consiguen los resultados deseados en el tiempo estipulado.

En conclusión, empezar a desarrollar productos de este tipo puede parecer duro debido a la complejidad, pero con el personal y medios adecuados puede ser uno de los procesos de desarrollo más divertidos e interesantes. Son proyectos que aúnan el arte, la tecnología y el entretenimiento en un único producto que puede ser disfrutado por todo el mundo

REFERENCIAS

Las fuentes mostradas a continuación fueron de gran ayuda para la realización del proyecto, han supuesto un gran origen de información. Las fuentes consultadas fueron:

- Documentación oficial de Unity2D (https://docs.unity3d.com/es/2019.4/Manual/Unity2D.html)
- El foro oficial de Unity 2D (https://forum.unity.com/forums/2d.53/)
- Varios hilos de GitHub (https://github.com/)
- Comunidad de desarrolladores de Unity(https://unityspain.com/)
- Canales de YouTube de desarrolladores de videojuegos (‘JoshCodes’,’ClubGamerZone’,’Brackeys’,’Blackthornprod’)

BIBLIOGRAFÍA

La bibliografía en la que me he apoyado para realizar este proyecto ha sido:

- Contenido teórico de la asignatura de Informática
- Contenido teórico de la asignatura de Informática industrial
- Contenido teórico de la asignatura de Proyectos
- Contenido teórico de la asignatura Organización y Gestión de Empresas

Estos contenidos han sido consultados para aplicar algunas de las metodologías que presentaba este proyecto, aunque no se ha citado contenido como tal se ha hecho un uso general de las asignaturas antes mencionadas

ANEXO: CÓDIGOS MÁS IMPORTANTES

Script del movimiento enemigo

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemymovement : MonoBehaviour
{
    public float speed;
    public Rigidbody2D rb;
    public bool isStatic;
    public bool isWalker;
    public bool walksRight;
    public bool ispatrol;
    public bool shouldWait;
    public float timeToWait;
    public bool isWaiting;
    Animator anim;
    public Transform wallcheck, pitcheck, groundcheck;
    private bool walldetected, pitdetected, isgrounded;
    public float detectionradius;
    public LayerMask whatisground;

    public Transform pointA, pointB;
    private bool goToA, goToB;

    public static enemymovement instance;
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
    }
    // Start is called before the first frame update
    void Start()
    {
        goToA = true;
        speed = GetComponent<enemy>().speed;
        rb = GetComponent<Rigidbody2D>();
        anim = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
        speed = GetComponent<enemy>().speed;
        pitdetected = !Physics2D.OverlapCircle(pitcheck.position,
detectionradius, whatisground);
        walldetected= Physics2D.OverlapCircle(wallcheck.position,
detectionradius, whatisground);
        isgrounded= Physics2D.OverlapCircle(groundcheck.position,
detectionradius, whatisground);
        if ((pitdetected || walldetected)&& isgrounded)
        {
            Flip();
        }
```

```
    }

    private void FixedUpdate()
    {
        if (GetComponent<enemy>().muertoperorespawn && isWaiting)
        {
            isWaiting = false;

        }


        if (!GetComponent<enemy>().muertoperorespawn)
        {


                if (!GetComponent<enemy>().damaged)
                {


                    if (isStatic)
                    {
                        anim.SetBool("idle", true);

                        rb.velocity = new Vector2(0, 0);
                    }

                    if (isWalker)
                    {
                        anim.SetBool("idle", false);
                        if (!walksRight)
                        {
                            rb.velocity = new Vector2(-speed * Time.deltaTime,
rb.velocity.y);
                        }
                        else
                        {
                            rb.velocity = new Vector2(speed * Time.deltaTime,
rb.velocity.y);
                        }
                    }

                    if (ispatrol)
                    {

                        if (goToA)
                        {

                            if (!isWaiting)
                            {
                                anim.SetBool("idle", false);
                                rb.velocity = new Vector2(-speed *
Time.deltaTime, rb.velocity.y);
                            }


                            if (Vector2.Distance(transform.position,
pointA.position) < 0.2f || Vector2.Distance(transform.position, pointB.position)
> Vector2.Distance(pointA.position, pointB.position))
                            {

                                if (shouldWait)
                                {
```

```
                                    StartCoroutine(Waiting());
                                }
                                Flip();
                                goToA = false;
                                goToB = true;
                            }
                        }

                        if (goToB)
                        {


                            if (!isWaiting)
                            {
                                anim.SetBool("idle", false);
                                rb.velocity = new Vector2(speed * Time.deltaTime,
rb.velocity.y);
                            }

                        if (Vector2.Distance(transform.position, pointB.position)
< 0.2f || Vector2.Distance(transform.position, pointA.position) >
Vector2.Distance(pointB.position, pointA.position))
                        {

                            if (shouldWait)
                                {
                                    StartCoroutine(Waiting());
                                }
                                Flip();
                                goToA = true;
                                goToB = false;
                            }
                        }

                    }
                }
            }

        }

        IEnumerator Waiting()
        {
            anim.SetBool("idle", true);
            isWaiting = true;

            yield return new WaitForSeconds(timeToWait);
            isWaiting = false;

            anim.SetBool("idle", false);

        }
        public void Flip()
        {
            walksRight = !walksRight;
            transform.localScale = new Vector2(-transform.localScale.x,
transform.localScale.y);
        }
}
```

Script de ataque enemigo

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class enemyproyectil : MonoBehaviour
{
    public GameObject projectile;
    public float timeToShoot;
    public float shootCoolDown;
    public Transform fireposition;
    public bool freqshooter;
    public bool watcher;
    private bool estamuerto;
    public Animator mianim;
    public GameObject muzzEffect;
    public float velocidad;
    public bool destruir = false;
    public static enemyproyectil instance;
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
    }
    // Start is called before the first frame update
    void Start()
    {

        shootCoolDown = timeToShoot;
    }

    // Update is called once per frame
    void Update()
    {
        estamuerto = GetComponent<enemy>().muriendo;
        if (freqshooter && !estamuerto)
        {
            shootCoolDown -= Time.deltaTime;
            if (shootCoolDown < 0 && !estamuerto)
            {
                shoot();

            }
        }
        if (watcher && !estamuerto)
        {

        }


    }


    public void shoot()
    {
        mianim.SetTrigger("shoot");
        AudioManager.instance.PlayAudio(AudioManager.instance.flame);
```

```
        GameObject cross = Instantiate(projectile, fireposition.position,
Quaternion.identity);

        if (transform.localScale.x < 0)
        {

            //Instantiate(muzzEffect, fireposition.position, transform.rotation);
            cross.GetComponent<Rigidbody2D>().AddForce(new Vector2(-500f, 0f),
ForceMode2D.Force);
        }
        else
        {

            //Instantiate(muzzEffect, fireposition.position, transform.rotation);
            cross.GetComponent<Rigidbody2D>().AddForce(new Vector2(500f, 0f),
ForceMode2D.Force);
        }
        shootCoolDown = timeToShoot;


    }
}
```