ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

DESARROLLO DE UN VIDEOJUEGO SERIO COMO HERRAMIENTA DE INICIACIÓN A LA PROGRAMACIÓN ESTRUCTURADA

DEVELOPMENT OF A SERIOUS VIDEOGAME AS AN INITIATION TOOL TO STRUCTURED PROGRAMMING

Realizado por
Alberto Reyes Martín

Tutorizado por
Antonio José Fernández Leiva

Departamento
Lenguajes y ciencias de la computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

Resumen

En este proyecto se pretende desarrollar un videojuego serio para PC usando para ello la herramienta Unity. Este videojuego se llama Killing Code, se trata de un juego 2D de disparos mezclado con plataformas que tiene la finalidad de iniciar a los jugadores en el mundo de la programación estructurada.

Esta función de enseñar al jugador se llevará a cabo por medio de cuestionarios puntuales en ciertos momentos del desarrollo del juego, recibiendo en función de su rendimiento en los mismos, recompensas que faciliten el completar el juego.

Para terminar el juego, el usuario deberá recorrer varios mundos derrotando a enemigos mientras va saltando entre plataformas y va aprendiendo sobre la programación de la forma más amena posible.

Como objetivo secundario también trata de concienciar a los jugadores sobre los malos hábitos a los que se puede llegar por la adicción a los videojuegos, algo que día a día se dan con más frecuencia entre los jóvenes.

Palabras clave: Videojuego, juego serio, PC, Unity, programación, cuestionarios, recompensas, plataformas, 2D, malos hábitos.

Abstract

This project aims to develop a serious video game for PC using the Unity tool. This video game is called Killing Code, it is a 2D shooting game mixed with platforms that has the purpose of initiate players in the world of structured programming.

This function of teaching the player will be carried out through specific questionnaires at certain moments in the development of the game, receiving rewards that facilitate the completion of the game based on their performance.

To end the game, the user must travel several worlds defeating enemies while is jumping between platforms and is learning about programming in the most enjoyable way possible.

As a secondary objective, the serious game also tries to raise awareness among the players about the bad habits that can be reached by addiction to video games, something that occurs more frequently every day among young people.

Keywords: Video game, serious game, PC, Unity, programming, quizzes, rewards, platforms, 2D, bad habits

Índice

Capítulo 1. Introducción

En este capítulo se encontrará una breve descripción del problema a satisfacer y una parte final en la que se hablará sobre la estructuración del proyecto.

Los videojuegos representan el área de entretenimiento que más ha crecido en los últimos años a nivel mundial y este campo ya genera más actividad comercial que el cine y la música juntos (Asociación Española de Videojuegos, 2019). El videojuego está además reconocido por el Parlamento español como industria Cultural desde el año 2009, lo que se traduce en una mayor cobertura a todos los niveles. Adicionalmente está demostrado que los videojuegos producen una mecánica positiva tanto a nivel personal como a nivel socioeconómico en un sentido más general de sociedad, que va más allá del ocio, y pueden provocar efectos positivos en sectores transversales y en campos de aplicación que pueden afectar las vidas de miles de personas y contribuir al progreso y la mejora social.

Un ejemplo de ello es la aparición de nuevos conceptos asociados directamente con los videojuegos tales como el término de juego serio que consiste en un juego real que es implementado con fines que van más allá del mero entretenimiento (Frida Díaz, 2016) y normalmente otros objetivos principales tales como incrementar la motivación en el aprendizaje de ciertos conceptos o involucrar a las personas en participaciones colectivas en redes sociales, por poner algunos ejemplos.

El desarrollo de un videojuego serio demanda un alto esfuerzo, debido en a enorme cantidad de trabajo a realizar en muchas fases distintas, ya sea a nivel de diseño (de personajes, de niveles, la narrativa y la interacción con el jugador), a nivel de programación (de la lógica, las mecánicas, los interfaces de usuario y las inteligencias artificiales) y nivel gráfico (escenarios, personajes, ambientación e interfaces).

En este contexto, el proyecto que aquí se propone busca desarrollar, desde cero, el videojuego Killing Code, un videojuego 2D de disparos mezclado con plataformas (Wikipedia, 2019) que tiene como finalidad principal iniciar a los usuarios en el mundo de la programación estructurada de una forma más

amena para que, si disfrutan con lo aprendido en el juego, avancen más en su aprendizaje de la programación con una buena base.

Por medio de preguntas en ciertos momentos del juego, el usuario responderá sobre conceptos básicos de la programación estructurada y recibirá mejoras para su personaje en función de los aciertos conseguidos. Se le permitirá repetir los cuestionarios tantas veces como quiera con el fin de que consiga aprender bien los conceptos.

El videojuego serio que se crea en este proyecto tiene además un segundo objetivo adicional al de introducir al usuario los conceptos básicos de la programación estructurada. Este segundo objetivo consiste en concienciar al jugador de algunos problemas que se pueden derivar al jugar a los videojuegos. Así, la historia principal también contará con fases de diálogos entre personajes y tramas argumentales en las que nos enfrentaremos a enemigos que simbolizan los hábitos tóxicos a los que se pueden llegar al jugar demasiado a los videojuegos, haciendo así que el usuario se conciencie sobre estos malos hábitos e intente evitarlos en su vida cotidiana. Ejemplos de estos malos hábitos son las trampas, la agresividad, el aislamiento del mundo exterior, la falta de ejercicio y el empeoramiento de la salud entre otros (Ana Isabel Ledo Rubio, 2018).

El desarrollo de un videojuego serio demanda un alto esfuerzo, debido en a enorme cantidad de trabajo a realizar en muchas fases distintas, ya sea a nivel de diseño (de personajes, de niveles, la narrativa y la interacción con el jugador), a nivel de programación (de la lógica, las mecánicas, los interfaces de usuario y las inteligencias artificiales) y nivel gráfico (escenarios, personajes, ambientación e interfaces).

Este proyecto se estructura de la siguiente manera:

- En el primer capítulo se hace una breve presentación del problema a abordar en el proyecto y de los objetivos a alcanzar en el desarrollo del proyecto.
- En el segundo capítulo se da una explicación detallada sobre todo lo que debe saber el lector para entender completamente el proyecto.

- En el tercer capítulo se pasa a explicar en más detalle el problema que trata de satisfacer el proyecto y el diseño de su solución.
- En el cuarto capítulo se detalla en profundidad la implementación de la solución diseñada en el Capítulo 3 y la metodología de desarrollo seguida.
- En el quinto y último capítulo se habla sobre las conclusiones a las que ha llevado el proyecto y su posible evolución en un futuro.

Capítulo 2. Antecedentes

En este capítulo, se plasmarán los conceptos base que el lector deberá conocer para poder entender por completo la totalidad de la temática tratada en este proyecto.

Capítulo 2.1 La programación informática y su dificultad de aprendizaje.

La programación en el ámbito de la informática se refiere a la acción de crear programas usando para ellos un conjunto de ordenes que hace al ordenador comportarse de cierta forma.

Existen una gran cantidad de nomenclaturas y estándares para escribir las líneas de código que los ordenadores convierten en operaciones. Estos estándares son los lenguajes de programación y según su legibilidad se clasifican en diferentes subgrupos.

Lo importante de la programación es que independientemente del lenguaje o su dificultad, todos se suelen regir por los mismos pasos que el programador debe seguir:

-Análisis del problema, es decir entender bien el problema a resolver, obtener una idea general de lo que se pide y que preguntas deben ser respondidas antes de resolver el problema.

-Implementación de la solución con la que se tratará llevar la solución que hemos pensado previamente del lenguaje humano al lenguaje de programación.

Por ello, la dificultad de la programación reside, además de en aprender un lenguaje determinado, en ser capaz de tener una mente abierta a la creatividad y poder resolver los problemas que se crucen de la manera más eficiente, clara y simple, lo cual no siempre es fácil.

Capítulo 2.2 Desarrollo de videojuegos.

A la hora de desarrollar un videojuego, también se deben abordar todas las fases que presenta la implementación de cualquier programa informático, sumando además ciertos aspectos que los hacen únicos. Estas fases extra pueden ser tales como una fase de diseño bien definida y recogida en un documento llamado GDD (Jorge Vallejo, 2017), con el cual los programadores podrán hacerse una idea de las claves principales del videojuego a implementar, tales como definir un público objetivo, la plataforma que soportará el juego, las reglas sobre las que se regirá, etc.

Además del GDD, se le añade la dificultad de tener que crear (si los diseña uno mismo) o adquirir (si están a la venta o no tienen derechos de autor que impidan su uso) recursos gráficos que plasmen la acción del juego en la pantalla. Es decir, se añaden al extenso trabajo de diseño y a la compleja labor de programación la necesidad de unos recursos gráficos y multimedia que permitan al usuario quedar más inmerso en la partida que está jugando. Claro queda que hace falta una enorme creatividad para hacer un buen juego y sea capaz de atraer al público esperado y tengo positivas valoraciones por parte de ese público.

Por todo esto, los videojuegos están pasando por su mejor momento, celebrándose debido a los mismos, convenciones, adaptaciones al cine, concursos de creatividad, etc. De hecho, los juegos son cultura (Raúl González, 2018) para una gran parte de la población, asemejándolos así con la industria del cine, el teatro y de los libros y cada año se invierten mayores cantidades de capital en este sector.

Capítulo 2.3 Videojuegos serios.

Los videojuegos tienen la principal función de entretener al jugador por encima de todo, pero no todos los videojuegos dan la misma prioridad a ese objetivo que, en ocasiones toma un segundo plano. Un claro ejemplo de ello son los videojuegos serios
(José Vicente Pons, 2017), un tipo de juego que tiene como una de sus finalidades la comprensión de conceptos, el uso como medio publicitario o incluso como entrenamiento en el campo de la simulación virtual.

Lo que diferencia a los buenos juegos serios del resto es la capacidad de saber mantener una línea entre la fantasía y el mundo real, evitando decantarse más por uno de los dos mundos. Además, hay que saber elegir bien el tipo de juego serio ya que los hay de tipo publicitario, educativo, de entrenamiento, de salud (los cuales

tratan temas de prevención de enfermedades o la vida sana), políticos (diseñados por ejemplo para educar sobre los derechos humanos) y religiosos entre otros.

Capítulo 2.4 Ingeniería de software ágil (scrum).

Las metodologías tradicionales de gestión de proyectos tenían una orientación básicamente predictiva, pues a partir del problema que se quiere abordar, se elaboran fases planificadas perfectamente en el tiempo basándose en los recursos en disposición del equipo de desarrollo (Project Management Institute, 2017). Sin embargo, estas metodologías tienen la desventaja de que son muy poco receptivas a cambios en el proyecto una vez se empieza a desarrollar, haciendo muy costoso aplicar estas variaciones en cualquier etapa del desarrollo.

Por ello, surgen las metodologías ágiles (Marble Station, 2008), las cuales parten de una idea de producto final poco precisa y se va perfilando a medida que se van avanzando en el desarrollo. Esto, permite a estas metodologías integrar muy fácilmente cualquier cambio en los requisitos iniciales de producto, puesto que no tiene un carácter predictivo tan fuerte como las metodologías tradicionales.

Actualmente, la metodología ágil que más se conoce es la metodología Scrum (Jeff Sutherland, 2014). Este método se basa en trabajar siguiendo una serie de iteraciones que se suelen planificar por semanas y tras las cuales se revisará todo el trabajo realizado en la iteración. Se centra también en ajustar resultados en base a las exigencias del cliente, haciendo reuniones con el mismo cada cierto tiempo y ver si el proyecto está tomando el rumbo que el cliente tenía planeado.

Otra de las características más destacables del Scrum es el solapamiento de distintas fases de desarrollo, algo bastante diferente a la tradicional planificación de cascada en la que no se puede volver a fases de desarrollo ya superadas.

Capítulo 2.5 Tecnologías de desarrollo de videojuegos.

Actualmente, existe un amplio abanico de posibilidades en lo que se refiere a tecnologías usadas para crear videojuegos. En función del ámbito, la seriedad y la plataforma objetivo del juego, se limitará en mayor o menor medida las opciones a la hora de elegir una herramienta u otra.

En el caso de este proyecto, se ha utilizado la herramienta de Unity 2D

(Dave Calabrese, 2014), debido a su extendido uso en la industria del videojuego a nivel profesional y que permite crear también juegos en 3D. Además de disponer de librería muy amplias y completas con bastante documentación, hay gran cantidad de preguntas y respuestas en foros de la propia comunidad de Unity en las que los mismos empleados de Unity hacen sus respuestas en el menor tiempo posible.

Algunas de las tecnologías de las que dispone Unity y se han usado en el desarrollo de este proyecto son:

- *Tile Palette*: utilizada para pintar los mapas de forma fácil y eficaz, con gran variedad de opciones como copiar y pegar trozos enteros de mapa simplemente seleccionando la zona con el ratón.
- *Cinemachine*: empleada para la lógica de las cámaras, cambio de planos y seguimiento de los personajes.
- *Timeline*: usada para crear cinemáticas y secuencias de juego añadiendo además efectos de sonido y música.

Unity dispone de muchas más funcionalidades que pueden ser consultadas en su página web (Unity3D, 2019).

Capítulo 2.6 Lenguaje de programación C#

C# es un lenguaje de programación orientada a objetos que surge con la idea de mejorar a los lenguajes de C y C++ pero añadiendo también las ventajas de Java. Por ello, tiene una sintaxis muy familiar a la de Java o a la de C++, pero añade tanto a Java las funcionalidades de las lambdas (añadidas en las últimas versiones de Java) y los accesos directos a memoria, como a C++ la facilidad de hacer operaciones complejas.

Otras de las fortalezas de C# son que fomenta el encapsulamiento de los componentes, su proceso de compilación es mas ligero que en C o C++. Por último, cuenta con librerías de Unity tales como UnityEngine (para comportamientos de los gameObjects) o UnityEngine.UI (Para interfaces de usuario y componentes del Canvas) que facilitan bastante la implementación de funcionalidades que serían bastante complicada de implementar sin usar las librerías.

Capítulo 3. Problema y diseño de la solución

En este capítulo, se explicará a fondo el problema a solventar con este proyecto y el diseño de la solución que se llevó a cabo en la fase previa al inicio de la implementación.

Capítulo 3.1 Objetivos del proyecto.

Este proyecto surge ante la necesidad que hay en el mundo actual de programadores informáticos en casi todos los sectores de la industria. Esto queda reflejado en las bajísimas tasas de paro de los estudiantes que terminan cualquier grado informático (AmericaEconomía, 2016) como se puede ver en las tasas de empleo de la Tabla 3.1.1. Por ello, se realizó un estudio previo para averiguar cuáles podrían ser las mejores formas de introducir a nuevos estudiantes en el mundo de la programación. Este estudio llevó a descubrir una opción de aprendizaje con la cuál se podía llevar al usuario a prender nociones básicas sobre cualquier cosa de una manera fácil y amena, haciendo que incluso lo pasara bien aprendiendo.

SECTORES CON MÁS OFERTAS DE EMPLEO 1S 2016	
1	Hostelería / Turismo / Restauración
2	Informática / Tecnología
3	Sanidad
4	Industrial
5	Telecomunicaciones
6	Tecnologías de Información
7	Automoción
8	Consultoría / Asesoría / Auditoría
9	Educación / Formación
10	Alimentación y Bebidas

Tabla 3.1.1: Sectores con más ofertas de empleo de 2016 en Latino América

Esta opción fue la de hacer un videojuego serio (explicados en el Capítulo 2.3) que permitiera a los usuarios de este, aprender sobre los conceptos más

básicos de la programación y eliminar parte de la incertidumbre que genera en los nuevos estudiantes sobre si la programación está hecha para ellos o no.

A esta incertidumbre inicial, se suman el desconocimiento que tiene la sociedad ajena a la informática de cómo funciona un programa informático, debido a que en el instituto o el colegio no se tocan (o apenas se hacen) los temas informáticos a nivel de líneas de código. Por ello, muchos alumnos que entran el primer año a carreras relacionadas con la programación informática empiezan con ganas porque no saben bien que van a dar y tienen ganas de aprender cosas nuevas y que se demandan mucho en trabajos actuales, pero no tardan en darse cuenta de la realidad de qué es la programación y cómo se hace un programa informático. Esto conlleva a que, si no se tiene la suficiente determinación en lo que se está estudiando, se abandone la carrera pensando que no está hecha para todo el mundo.

Según la Fundación Conocimiento y Desarrollo (Shutterstock, 2017), se llega a la estadística de "un fracaso escolar de 22,5% durante el primer año para el período 2015-2016. Una de las causas más preocupantes es la falta de orientación al momento de elegir una carrera."

Esta falta de orientación era un motivo más que idóneo para realizar un proyecto que pudiera, por una parte, ayudar a los estudiantes a decidirse si deben o no empezar una carrera informática y por otra atraer a gente de todas las edades hacia este mundo de la programación de la forma más original posible.

Así que con estos datos ya encima de la mesa se hacía clara la idea de desarrollar un videojuego educativo que enseñara conceptos básicos sobre la programación, aquellos que más echaran para atrás a los nuevos estudiantes a la hora de matricularse en una carrera relacionada con la informática, ya que se consideró idóneo el solventar todas las dudas posibles que se pudieran tener a lo largo de la historia principal del videojuego.

Algunos de estos básicos debían ser tales como: qué es un lenguaje de programación, como se estructuran los programas informáticos, que tipos de lenguajes de programación hay, que lenguajes son más fáciles para iniciarse en la programación, que son las variables, los tipos de datos, las estructuras iterativas, etc.

Se pensó que con esta finalidad como base podrá quedar un producto final bastante interesante de cara al público al que iba enfocado, pero en el éxito de un

videojuego serio está presente también la jugabilidad y una trama argumental que envuelva al jugador y le permita disfrutar de un videojuego en la que la habilidad tome también un papel importante.

Pero antes incluso de elegir qué género iba a poseer el videojuego aparte del de juego serio, había que elegir la herramienta de desarrollo que iba a emplearse en el proyecto y para ello, se estudiaron de forma cuidadosa cual de todas las herramientas más usadas del momento era la adecuada.

Las dos herramientas más usadas para este tipo de juegos casuales eran Unity y Unreal Engine (Imagen 3.1.1), por lo que se debían analizar bien las fortalezas y debilidades de cada una antes de tomar una decisión final.

Estas diferencias pasarán a explicarse a continuación:

Imagen 3.1.1: Unity contra Unreal Engine

Unity

Unity dispone de una gran cantidad de documentación en sus librerías y esto, unido a que utiliza C#, un lenguaje muy sencillo de aprender si se sabe programar en algún otro lenguaje orientado a objetos (Miguel Angel Álvarez, 2001), lo que daba un gran punto a favor para esta herramienta en lo que a curva de aprendizaje se refiere.

Unreal Engine

Por otra parte, Unreal Engine ofrece una mayor optimización de recursos del sistema, debido a que su desarrollo está una capa por debajo a Unity, permitiendo así hacer ciertas cosas de una forma más sencilla.

Debido a hay más documentación disponible y que el uso de Unity está mucho más extendido en la industria de desarrollo de videojuegos (sobre todo en el mundo del desarrollo independiente) se tomó finalmente como mejor opción la de Unity, para poder así aprender desde cero el como hacer un videojuego de una forma más rápida.

Una vez seleccionada la herramienta de Unity, era hora de elegir en primer lugar si el juego debiese tener una vista en 2 dimensiones o si fuera más aconsejable desarrollarlo en 3 dimensiones, es decir si usar Unity 2D o en su lugar aventurarse con Unity 3D. Para tomar esta decisión, se estudiaron las diferencias principales entre los tipos de juegos realizados con Unity 2D y los desarrollados usando Unity 3D.

El primer detalle a destacar en la investigación desarrollada fue el descubrir que, en el motor de 2 dimensiones, Unity 2D emplea básicamente todas las funcionalidades de su versión en 3D, permitiéndote explorar un juego en 2D con una tercera dimensión añadida.

A medida que se siguió estudiando qué tipos de vistas usaban los videojuegos serios en su mayoría, triunfaba la visto 2D frente a la 3D, en gran parte debido a que no necesitas que haya escenarios increíbles o animaciones de película, sino que lo más importante es transmitir un mensaje al usuario a la vez que se entretiene jugando.

Cabe a destacar también lo limitados que son los recursos gráficos disponibles gratis por Internet en 3D en comparación con los de 2D. Como sería un esfuerzo demasiado grande el crear unos recursos gráficos o multimedia propios además de hacer toda la implementación del videojuego, se pensó desde un primer momento en descargarlos de forma gratuita y respetando los derechos de autor de páginas como (OpenGameArt).

Un ejemplo de estos juegos serios en 2D encontrados durante el estudio es TimeMesh, un juego educativo de hacer clics en el que se enseñan datos sobre eventos históricos (Imagen 3.1.2)

Imagen 3.1.2: Captura de TimeMesh, un juego serio en 2D

Ya con herramienta con la que trabajar, motivo principal del proyecto y con el tipo de 2D seleccionado como vista del juego, restaba ya una última toma de decisiones antes de empezar a esbozar el juego: ¿qué subgénero debía poseer el juego?

Se necesitaba un género que fuera popular y que además combine bien con su funcionalidad de enseñar al jugador. Por ello, y al tratarse de un juego 2D, se eligió un juego de plataformas ya que son ideales para empezar a manejar Unity sin realizar un trabajo ni muy complejo ni demasiado fácil. Además de todo esto, se eligió el género de plataformas debido en gran parte a la afinidad con las vistas 2D, debido a la necesidad de precisión en los movimientos para poder completar niveles muy complejos y que requieran de una habilidad moderada por parte del jugador. Un ejemplo de un juego 2D de plataformas muy conocido es la saga de Super Mario Bros (Imagen 3.1.3).

Imagen 3.1.3: Ejemplo de juegos de plataformas 2D, saga Super Mario Bros

Una vez tomadas todas estas decisiones, se procedió al planteamiento de una solución inicial y de un diseño que serviría de base en la futura fase de implementación de la solución. Este diseño debería ser aprobado por el cliente (en este caso el tutor del proyecto) mediante reuniones concertadas.

Capítulo 3.2 Fase de diseño de la solución

Una vez tomadas todas las decisiones como el problema a solucionar con el proyecto, la temática del videojuego, la herramienta con la que se hará el desarrollo, el lenguaje usado en la implementación, el tipo de vista y el género que poseerá el juego (todo ello explicado en detalle en el Capítulo 3.1), se redactó un documento que sirve como pilar en cualquier proyecto de creación de videojuegos, el GDD o Game Design Document.

Capítulo 3.2.1 Game Design Document (GDD)

El GDD es básicamente el documento donde se recogen todas las indicaciones que el diseñador del videojuego ha creado para los desarrolladores.

El diseñador del videojuego se encarga de hacer el diseño (valga la redundancia), y las reglas por las que se va a regir el juego, esto es, plasmar antes

de empezar a programar las mecánicas que debe cumplir el videojuego y como interactuará el usuario con el mismo.

El GDD es un documento que aun pareciendo que es algo innegociable una vez que llega a los programadores, puede verse sometido a cambios en función de la metodología elegida para el proyecto (en el caso de este proyecto se optó por una metodología Scrum (Jeff Sutherland, 2014)).

Estas modificaciones en el GDD deben ser aprobadas tanto por el diseñador como por el cliente, en reuniones periódicas que se deben ir haciendo con regularidad (el avance de las iteraciones del proyecto y el registro de las reuniones será explicado en el Capítulo 4.2).

Habiendo entendido la importancia del GDD e investigar que todos los videojuegos a nivel profesional tomaban esta forma de trabajo, se empezaron a tomar decisiones en función de ciertas directrices que se encontraron en una página web y que parecía una lista bastante completa (Leandro González, 2016).

A continuación, se enumerarán las decisiones tomadas para el GDD y en el Anexo se podrá encontrar la última versión realizada sobre el mismo por si se quiere consultar más en profundidad sobre las decisiones tomadas:

- PERSONAJES

Se tomó como idea tener un protagonista que asesorado por un mago poderoso fuera capaz de enfrentarse a un virus informático que había infectado su juego favorito.

Esto se basó en otros juegos de aventuras con títulos como *The Legend of Zelda Phantom Hourglass*, en el que el personaje principal, llamado Link es asesorado por un espíritu volador que le acompaña en sus aventuras para salvar a una princesa (Imagen 3.2.1).

Imagen 3.2.1. Juego The Legend of Zelda Phantom Hourglass

Además, había que dar forma a los enemigos y buscar una finalidad más allá de que fueran los malos para que el jugador pudiera asociar el derrotar a esos enemigos con hacer algo realmente bueno. Por ello, se llegó a la idea de agregar una historia asociada a la existencia de cada enemigo y finalmente se hizo que los enemigos fueran por así decirlo reencarnaciones de las personas que llevaron demasiado al extremo su afición por los juegos y se quedaron atrapados en sus consolas. Estos extremos son, por ejemplo, el aislamiento del mundo exterior, la agresividad, y la obesidad entre otras cosas (Informática I, 2014).

Los recursos gráficos fueron descargados de este enlace gracias a un usuario llamado Robert: https://0x72.itch.io/dungeontileset-ii

- HISTORIA PRINCIPAL

La trama principal se resume básicamente en que un día dentro de un videojuego de realidad virtual muy de moda, aparece un virus que atrapa a todos los jugadores en su interior. Ante esto, el maestro del juego coge al protagonista y lo lleva a un lugar seguro para ponerle al día de la situación y le explica que para poder combatir el virus habrá que desarrollar un código lo suficientemente potente para liberar a todos los jugadores.

Dicho esto, se explica que el juego será serio e intentará hacer que el jugador aprende conceptos claves sobre la programación además de tener que ir derrotando a los enemigos que se crucen por su camino.

Esta temática se basó en historias de películas y series como la que hay en el anime *Sword Art Online* (Imagen 3.2.2), en el que el personaje principal se queda atrapado en su juego favorito de realidad virtual por un fallo del propio juego.

Imagen 3.2.2. Kirito, protagonista del anime Sword Art Online

- PROGRESO EN LA HISTORIA

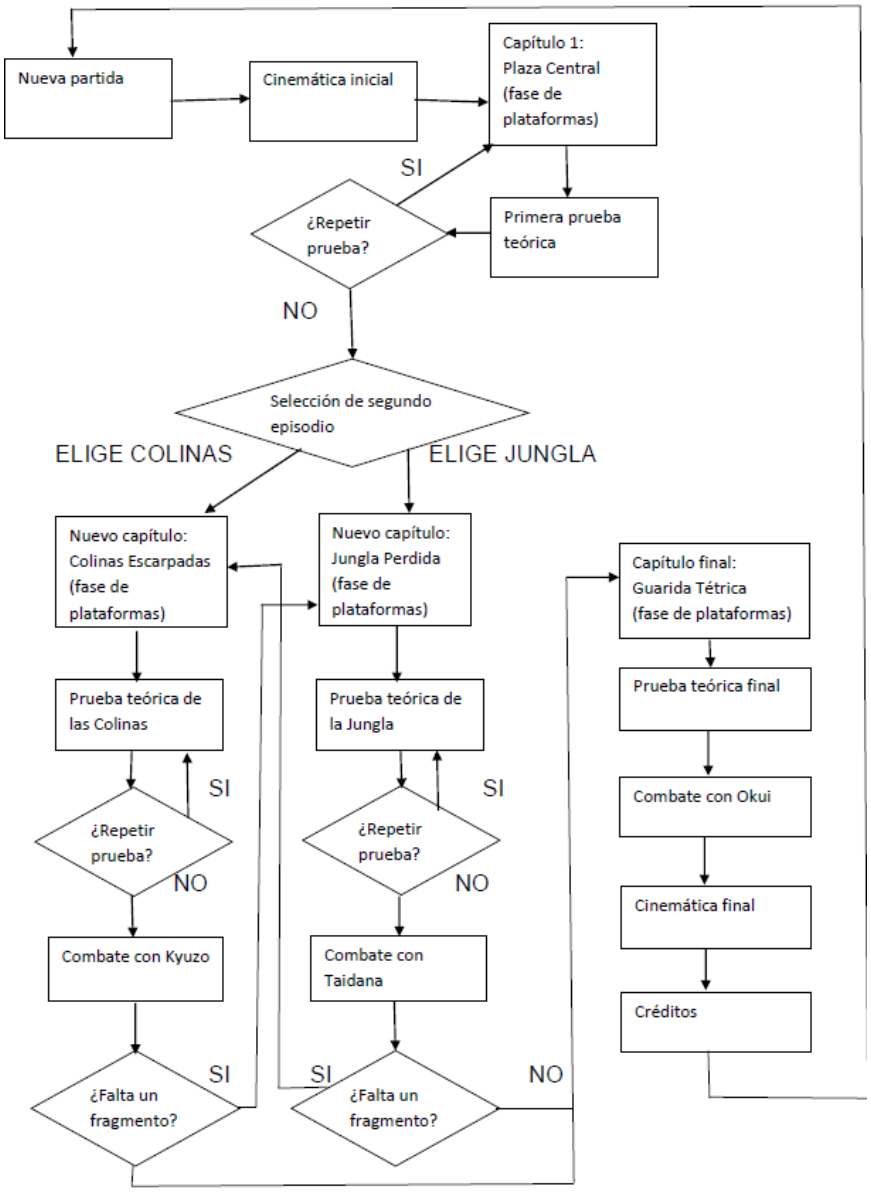


Figura 3.2.3. Diagrama de fujo de la historia principal.

El jugador empezará en un nivel que servirá a modo de tutorial para que, una vez completado elija en qué nivel meterse usando uno de los dos portales que habrá al final del nivel.

Cada uno de los niveles contará con un jefe final en su último tramo y cuando se derroten a los dos jefes, se desbloqueará el mundo final donde esperará el jefe definitivo del juego. Tras derrotarlo, el juego acabará.

Cada nivel contará también con una prueba teórica en la que el maestro del juego pondrá a prueba al jugador enseñándole conceptos básicos de programación intercalándolo con comentarios divertidos para hacer más amena la explicación.

En el GDD se especificó un diagrama de flujos explicando el progreso que se seguiría en el juego (Figura 3.2.3).

- JUGABILIDAD

Se marcaron dos objetivos principales para el proyecto:

- A largo plazo, el jugador debía derrotar a los tres jefes para salvar el mundo virtual del juego.
- A corto plazo, el jugador debería avanzar por los niveles desbloqueando habilidades nuevas y aprendiendo poco a pocos conceptos base de la programación.

Para ello, el juego contaría con ciertas mecánicas que debían ser definidas en la fase de diseño. Estas serían, tratándose de un juego de género plataformas, características físicas como gravedad y rozamiento con las plataformas, las teclas para mover al personaje y poder usar sus diferentes habilidades, la existencia de objetos interactivos como cofres e interruptores y la posibilidad de guardar el progreso interactuando con algún tipo de punto de guardado.

Los combates, se llevarían a cabo en función de los enemigos a derrotar, es decir os enemigos más débiles tendrán patrones de comportamiento sencillo y serían relativamente fáciles de derrotar uno a uno, pero los jefes tendrían comportamientos mucho más complejos e incluso tendrían habilidades especiales, todo ello para que el derrotarlos se convierta en un verdadero reto para el jugador.

Se decidió también que el personaje tendría vidas infinitas, pero contaría con una barra de salud que, en caso en que llegue a cero, hará que el jugador reaparezca en el último punto de guardado.

- RECURSOS DE LOS MAPAS Y EFECTOS MULTIMEDIA

Se decidió cambiar los aspectos de las plataformas y los fondos del mapa en cada uno de los mundos que se diseñaran teniendo las 4 diferentes paletas de plataformas que se muestran en las imágenes 3.2.4, 3.2.5, 3.2.6 y 3.2.7

Las imágenes de las casillas de las plataformas fueron descargadas de este enlace gracias a OnixGames:

https://opengameart.org/content/construct-2-tileset-4x-tilemaps-4x-backgrounds-4x-objects

Imagen 3.2.4. Paleta de plataformas primer mundo

Imagen 3.2.5. Paleta de plataformas mundo jungla

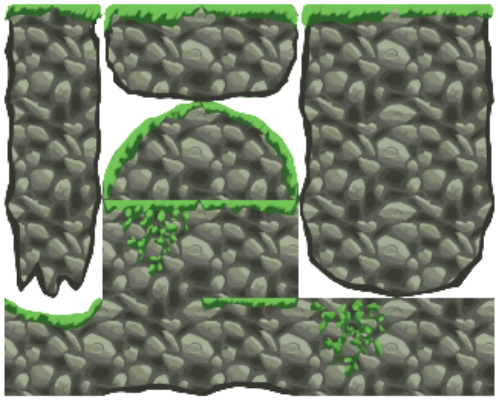


Imagen 3.2.6. Paleta de plataformas mundo montañas

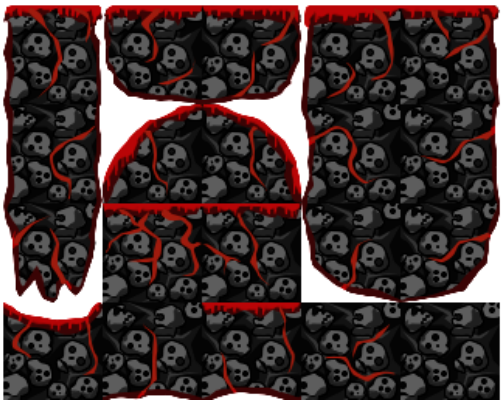


Imagen 3.2.7. Paleta de plataformas mundo final

En lo que a los recursos multimedia se refiere, se decidieron asignar músicas de fondo de más tensión en las peleas con los jefes, de una tensión moderada en las fases normales, relajante en las pruebas teóricas para fomentar la concentración, neutral en los menús y efectos sonoros al recibir daño por parte de los enemigos o cuando lancen un ataque especial.

- DESCRIPCIÓN TÉCNICA

En este aspecto, se definieron que se usarían la herramienta de desarrollo de Unity 2D (cuya decisión quedó explicada en el Capítulo 3.1) y en la plataforma de Windows, puesto que es el sistema operativo usado en el desarrollo del juego y haría más fácil la fase de pruebas.

- MERCADO

Aun sabiendo que el juego tenía que ser gratuito ya que este tipo de proyectos no se pueden monetizar, era necesario el concretar quienes iban a ser el público objetico y el idioma usado en el juego.

El público objetivo (ampliamente explicado en el Capítulo 3.1) serían los estudiantes que necesitaran algo de información antes de iniciar una carrera de

programación o cualquier persona que se quisiera sumergir en el mundo de la programación sin importar su edad o sexo.

El juego estaría disponible completamente en español a excepción de ciertos términos que al provenir del inglés tendrían que escribirse en ese idioma.

- OTRAS IDEAS

Se barajó también la posibilidad de añadir un inventario en el que el jugador pudiera guardar pociones para usarlas cuando quisiera.

Capítulo 3.2.2 Bocetos de mapas

Una vez diseñado por completo el GDD y teniendo en cuenta que el juego iba a ser de tipo plataformas, se hicieron diseños de mapas antes de empezar la implementación de la solución. Para ello se definieron diferentes tipos de plataformas con diferentes comportamientos, inspirados en otros juegos de plataformas para dar así un mayor nivel de dificultad al juego.

Estas plataformas fueron:

- *Plataformas unidireccionales*, en las cuales el jugador puede pasar sin problema si lo hace desde abajo hacia arriba y que son sólidas una vez se está sobre ellas (inspiradas en las plataformas de la Imagen 3.2.8).

Imagen 3.2.8. Plataforma unidireccional Super Mario Bros para la consola Nintendo DS

- *Plataformas temporales*: plataformas que en el momento en el que el jugador las pise, tengan un temporizador que provoque que, al terminar, la plataforma se caiga. La plataforma volverá a aparecer al cabo de un tiempo (inspiradas en las plataformas de la Imagen 3.2.9).

Imagen 3.2.9. Plataforma temporal Super Mario Bros.

- *Plataformas móviles* en las que en el momento en que el jugador las toca por primera vez, empiezan su recorrido hasta llegar al punto de destino designado. Repetirán este recorrido infinitamente (inspiradas en las plataformas de la Imagen 3.2.10).

Imagen 3.2.10. Plataforma móvil Super Mario Bros

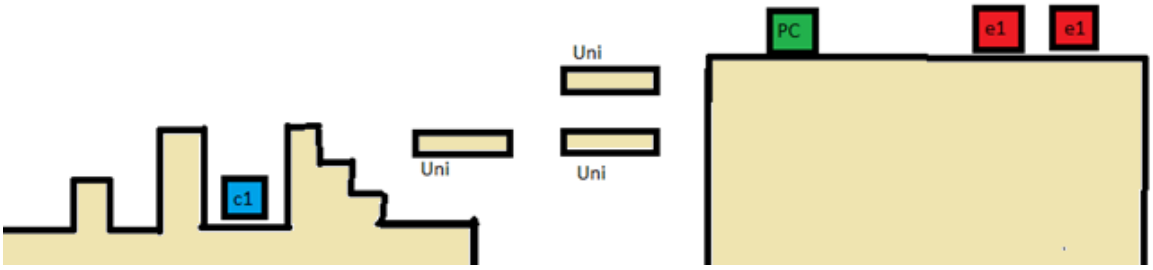
Establecidas las diferentes plataformas, se asignó una leyenda para identificar fácilmente los componentes en los bocetos. A continuación, se explicará dicha leyenda:

- **E1, E2, E3:** Enemigos de tipo 1 (Akuma), tipo 2 (Odei) o tipo 3 (Hakka) (consultar enemigos en el GDD del Anexo para tener más detalles sobre sus comportamientos y aspectos).
- **PC:** Punto de control

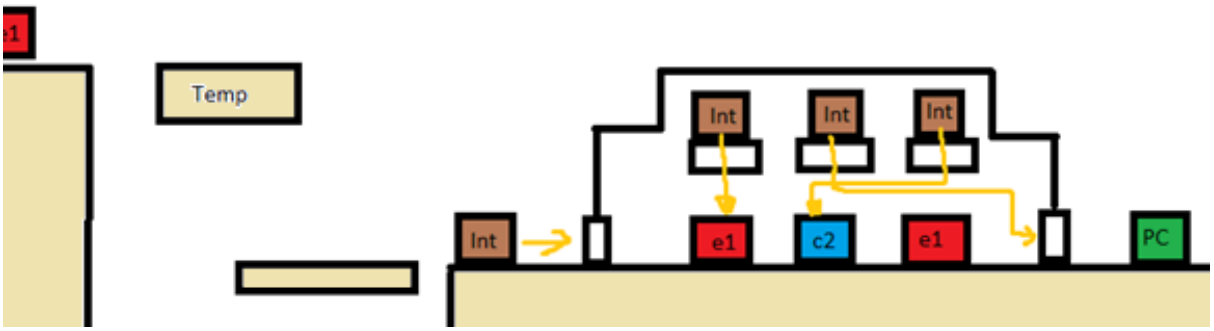
- **C1, C2, C3, C4:** Cofre de tipo 1, 2, 3 o 4 (hace aparecer un consumible azul, verde, rojo o amarillo respectivamente, consultar Anexo para ver efectos de las pociones)
- **Int:** Interruptor. Tendrá una flecha asociada indicando si hacen desaparecer alguna puerta o si hacen aparecer un cofre o enemigo.
- **Port:** Indica el fin de la fase y la entrada a otra nueva.
- **Mov:** Plataforma móvil que tendrá dibujado también el recorrido que seguirá
- **Temp:** Plataforma temporal.
- **Uni:** Plataforma unidireccional.
- **Jefe:** La flecha verde indica el elemento de mapa que aparece al iniciar el combate y la flecha roja señala el elemento que desaparece cuando el enemigo es derrotado (consultar jefes en el GDD del Anexo para tener más detalles sobre sus comportamientos y aspectos).

Una vez aclarados todos los estándares tomados en los bocetos de los niveles, se adjuntarán a continuación todos los que fueron creados para el proyecto.

El primer nivel, al servir de tutorial, debía tener un poco de cada elemento que fuera a haber en el juego sin ponerle las cosas demasiado complicadas al jugador:

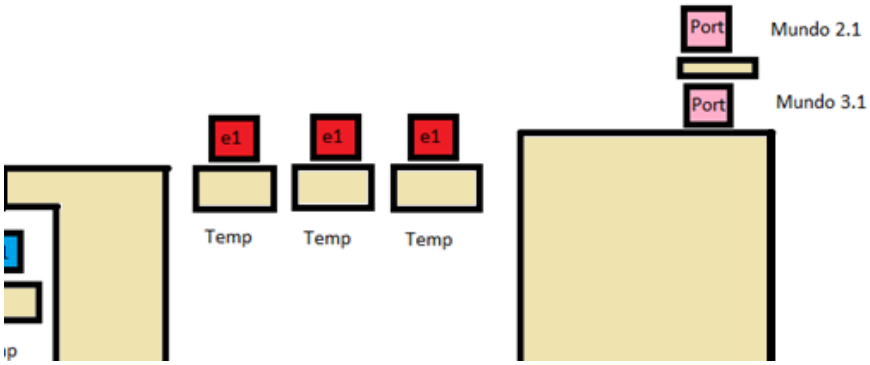


Boceto Nivel 1 (Parte A)



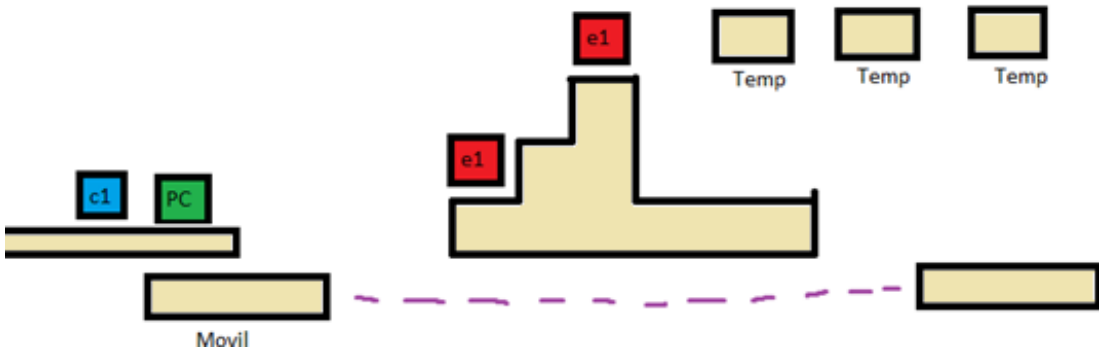
Boceto Nivel 1 (Parte B)

Boceto Nivel 1 (Parte C)

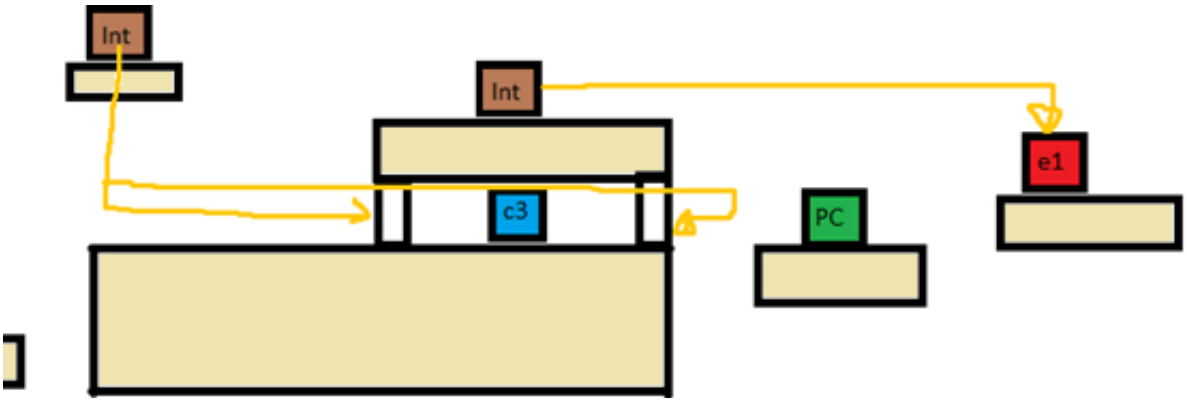


Boceto Nivel 1 (Parte D)

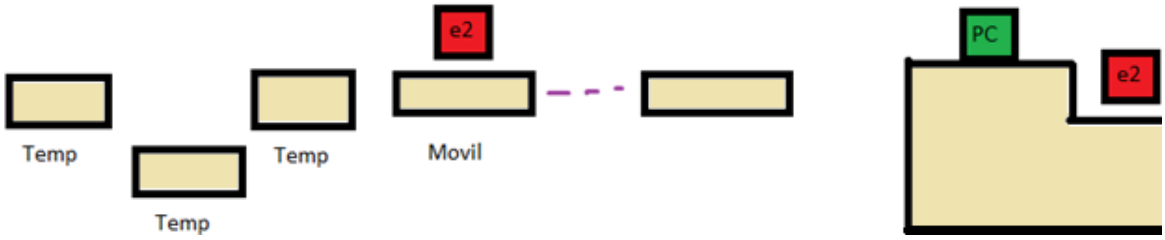
El mundo 2, es el mundo de la jungla y, por tanto, tendría que tener una mayor dificultad que el primer nivel. Esta dificultad residiría en saltos que demandaban una mayor precisión por parte del jugador y un juego constante con las plataformas especiales explicadas al principio de este Capítulo 3.2.2.



Boceto Nivel 2 (Parte A)

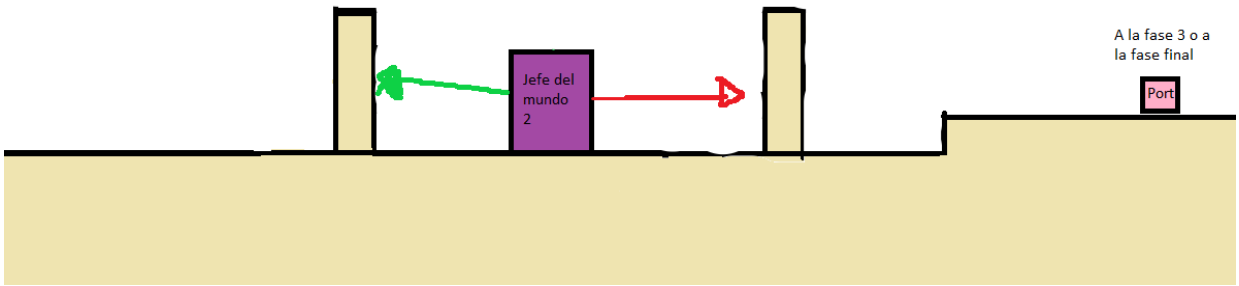


Boceto Nivel 2 (Parte B)



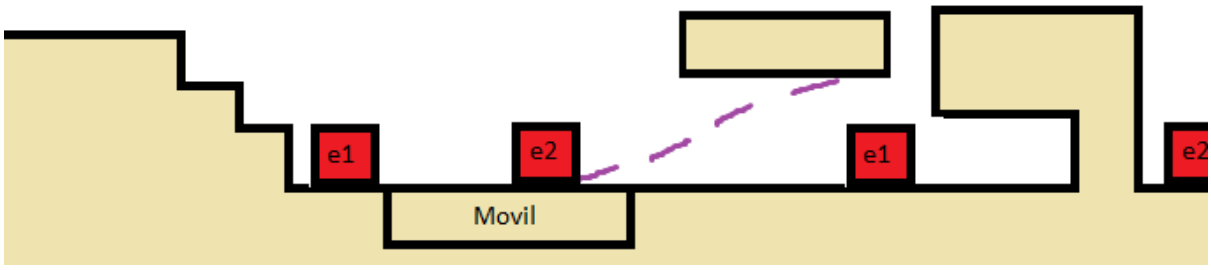
Boceto Nivel 2 (Parte C)

Boceto Nivel 2 (Parte D)

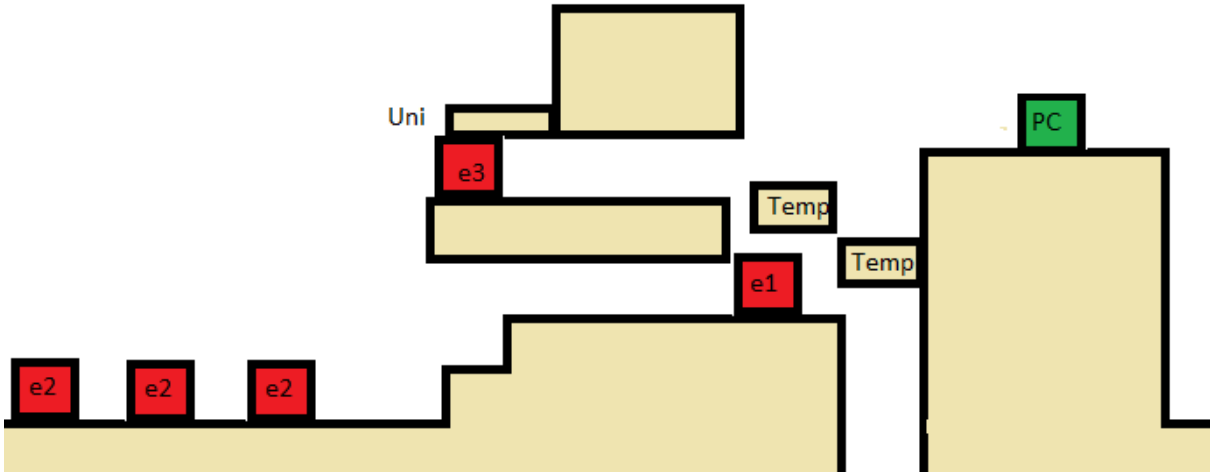


Boceto Nivel 2 (Fase de jefe)

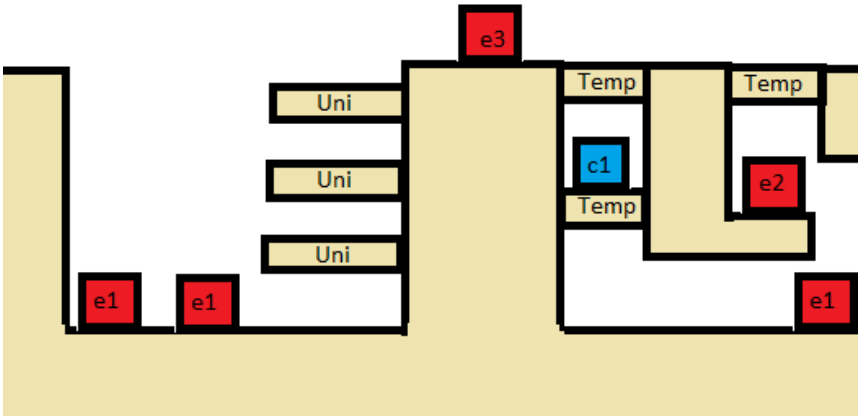
En el mundo 3 se buscaba un nivel de dificultad similar al del mundo 2 pero que, en lugar de que esta dificultad residiera en la coordinación del jugador a la hora de hacer saltos, se midiera por la habilidad a la hora de enfrentarse a gran variedad de enemigos.



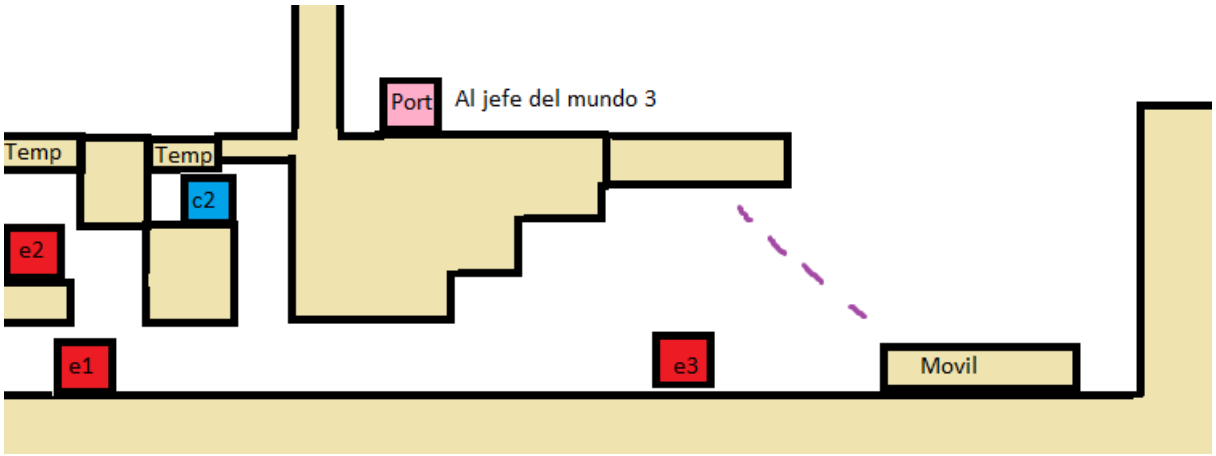
Boceto Nivel 3 (Parte A)



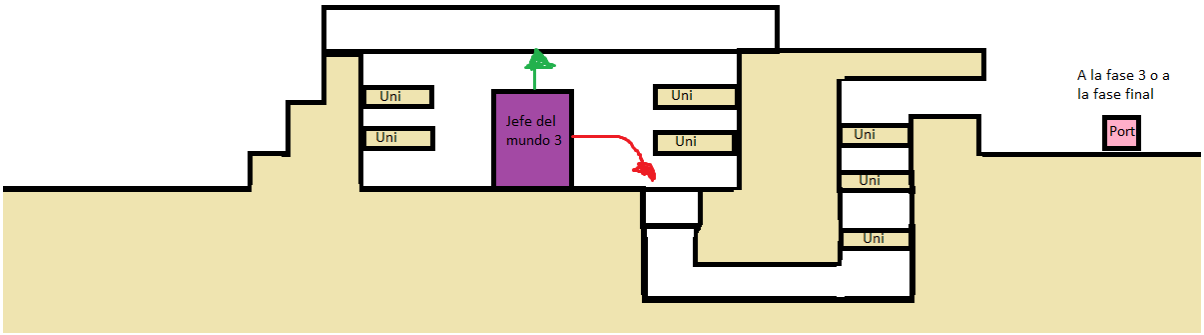
Boceto Nivel 3 (Parte B)



Boceto Nivel 3 (Parte C)

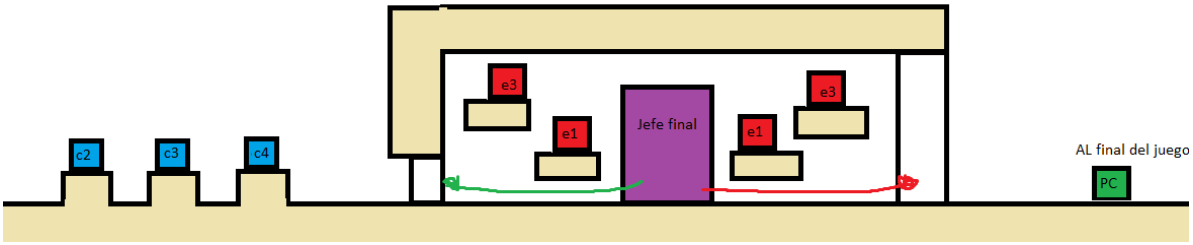


Boceto Nivel 3 (Parte D)



Boceto Nivel 3 (Fase de jefe)

El mundo final, se basaría exclusivamente en la batalla con el jefe final, por lo que la fase sería corta y contendría consumibles para ayudar al jugador en su batalla final.



Boceto Fase final

Ahora que todo el diseño previo estaba realizado, se podía empezar a hacer los correspondientes diagramas de clases, interfaces de usuario y casi casi empezar a programar. Algunos de los niveles o componentes plasmados en esta sección finalmente no fueron añadidos en el juego, pero en el Capítulo 5 se hablará de hacia donde evolucionará el juego en el futuro.

Capítulo 4. Implementación de la solución

En este capítulo, se detallará a fondo la fase de implementación que siguió a toda la etapa de diseño explicada en el Capítulo 3.

En primer lugar, se explicará cómo se hizo la implementación del código que hizo posible el funcionamiento correcto del juego para seguidamente proceder a la explicación de cómo se ha integrado una metodología ágil Scrum en la planificación realizada.

Capítulo 4.1 Implementación mediante Unity 2D y C#

El resultado final del videojuego fue fruto de combinar a la perfección dos tecnologías bastante distintas entre ellas.

Por un lado, se aprendieron a utilizar las herramientas que nos ofrece el entorno de trabajo de Unity 2D para crear interacciones con cámaras, dibujar escenarios, crear animaciones, etc.

En el otro lado del sumatorio que dio a luz la resolución final de *Killing Code* fueron los scripts en C# que tuvieron que ser escritos desde cero y que se acabaron comunicando entre ellos para lograr interacciones entre componentes de lo más pulidas.

A continuación, se explicarán estas dos fuentes tan interesantes por separado, empezando en este caso con las herramientas de Unity2D.

Capítulo 4.1.1 Uso de herramientas de Unity2D

Antes de proceder a explicar las herramientas usadas, es necesario destacar que Unity cuenta con muchas más herramientas de las usadas en este proyecto, pero se intentaron usar las más extendidas y efectivas de entre las disponibles.

TILE PALETTE

La primera herramienta que se va a tratar (y también la primera que se aprendió a usar y se implementó en el juego) es la de Tile Palette (Unity TilePalette, 2019). Tile Palette, es utilizada para crear de forma asombrosamente rápida y eficaz escenarios y plataformas con un esfuerzo mínimo. Para usarla, en primer lugar, hay

que crear una Tile Palette, el cual tendrá asignado las imágenes usadas a la hora de diseñar el mapa deseado.

En la Figura 4.1.1 se puede observar una de las paletas de casillas creadas en el proyecto para el nivel inicial.

Figura 4.1.1. TilePalette del mundo inicial creada usando la herramienta TilePalette

Una vez creada la paleta, simplemente se arrastran las casillas que queramos sobre la escena sobre la que queramos pintar el escenario, generando así un Tilemap en la escena que engloba a todas las casillas pintadas con TilePalette. Esto queda mejor representado en las Figuras 4.1.2 y 4.1.3.

En la figura 4.1.2 se ve cómo se selecciona el icono de la brocha en la parte superior y luego se selecciona la casilla de la paleta que queramos usar.

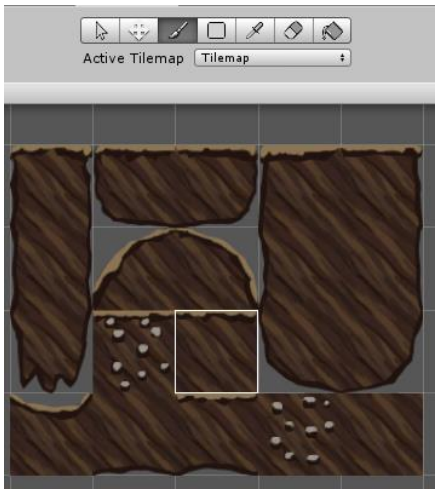


Figura 4.1.2. Selección de casilla con la que pintar en la TilePalette

Ahora solo hay que mantener el ratón mientras se arrastra por la escena y se irá pintando el escenario. En la imagen 4.1.3 se ve un escenario creado con TilePalette en unos pocos segundos.

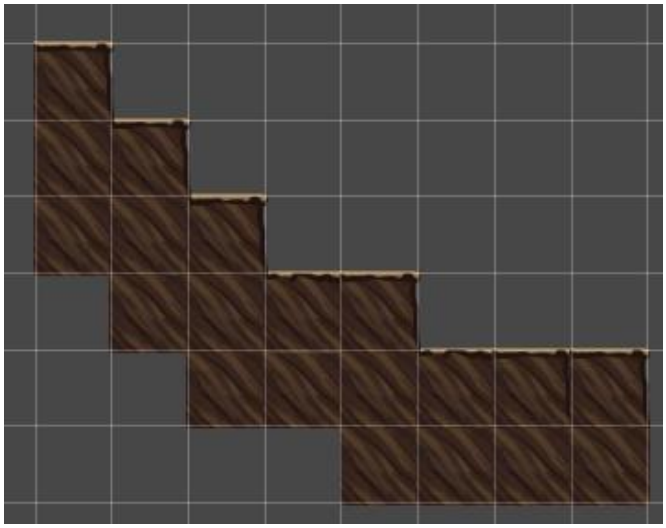


Figura 4.1.3. Parte de escenario creado con una Tile Palette

Pero esta herramienta nos permite también hacer cosas como crear agrupaciones de casillas propias pulsando el botón que pone Edit (Figura 4.1.4). Esto nos permite de forma aún más fácil hacer escenarios, puesto que puedes generar por ejemplo plataformas de 2 casillas de alto y 2 de ancho simplemente seleccionando 4 casillas que se hayan puesto contiguas previamente (Figura 4.1.5).

Figura 4.1.4. Agrupación personalizada de casillas en TilePalette

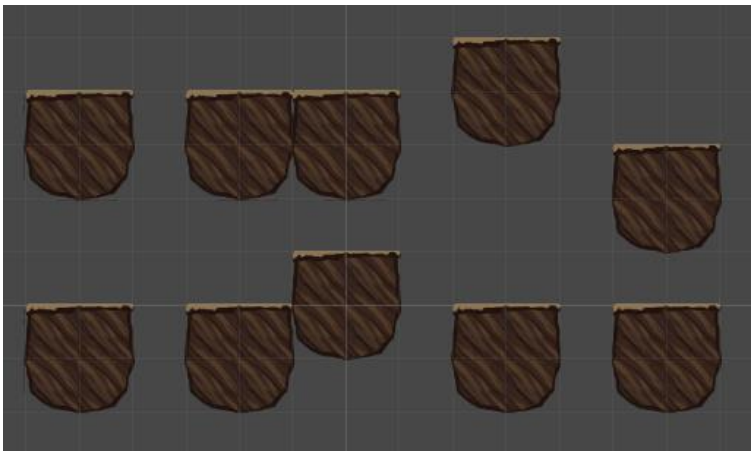


Figura 4.1.5. Uso de la agrupación personalizada de la Figura 4.1.3

Por último y para hacer aún más sencillo si cabe la creación de mapas, es posible configurar TilePalette de forma que cada una de las casillas pintadas tengan asociadas su propio controlador de colisiones que impida por ejemplo a nuestro personaje el atravesar la plataforma. Para ello, hay que asignar al Tilemap asociado al mapa pintado con TilePalette un componente llamado Tilemap Collider 2D, seleccionar la opción Used By Composite y luego asignar un Composite Collider 2D (Figura 4.1.5)

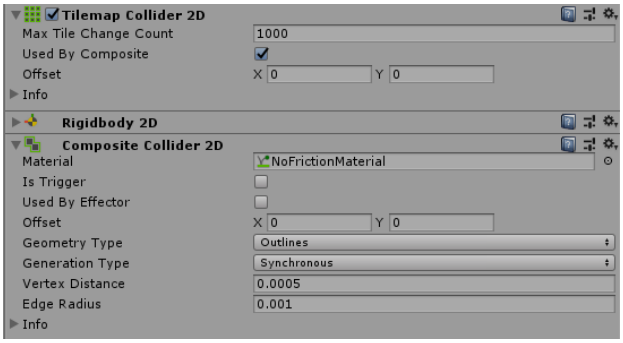


Figura 4.1.5. Componentes necesarios para asignar controladores de colisiones a un TileMap

Aplicando todas las utilidades que se han citado sobre TilePalette se han creado todos los mapas de los niveles del proyecto, a excepción de las plataformas especiales, las cuales serán explicadas en el Capítulo 4.1.2.

CINEMACHINE

Cinemachine es una herramienta de Unity (Unity TilePalette, 2019) muy popular incluso a nivel profesional debido a que sus funcionalidades se van ampliando cada poco tiempo y permite hacer muy sencillo el cambio entre cámaras, efectos de vista e incluso cinemáticas. Sus funcionalidades son tan amplias que no ha sido posible aprender a usar la herramienta a su máximo potencial en este proyecto, pero se repasarán las que fueron capaces de aplicarse en el mismo. Para comenzar, hay que descargarse la herramienta tal y como se indica en el enlace proporcionado.

Cinemachine basa su funcionamiento en hacer que una cámara principal (la que crea Unity por defecto al crear un nuevo proyecto) gestione transiciones entre distintas cámaras virtuales como si se tratara del director de una orquesta.

Por ello, lo primero que hay que hacer es crear una cámara virtual y asignarle la mayor prioridad de todas en el inspector (Unity Inspector window, 2019) para que sea nuestra cámara principal. Al crear la primera cámara virtual, se puede ver como la cámara principal tiene ahora un nuevo componente (Figura 4.1.6), el Cinemachine Brain, el cual se encargará de gestionar las transiciones de cámaras virtuales automáticamente. Además, cuando más de una cámara virtual está activa, mostrará la que mayor prioridad tenga asignada, por lo que se deberá asignar una cámara virtual con la mayor prioridad de todas las creadas y se usará como cámara principal.

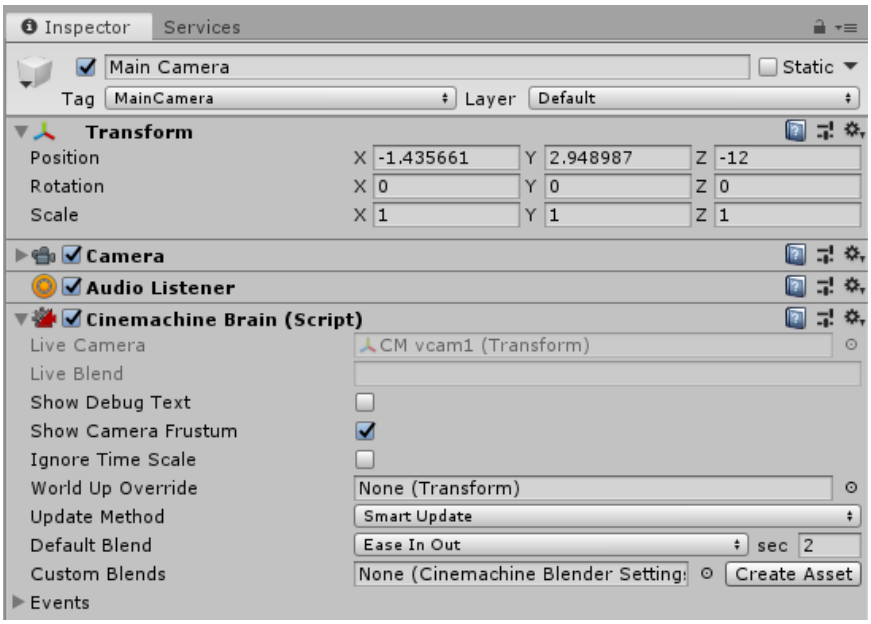


Figura 4.1.6 Componente Cinemachine Brain de la cámara principal

Ya con una cámara virtual gestionada automáticamente por la cámara principal de la escena, podemos hacer que esta cámara virtual siga a el objeto que queramos de la escena. Además, podemos definir los márgenes del movimiento de seguimiento, es decir, en que zonas de la pantalla la cámara seguirá más rápidamente al objeto o cual es considerado el centro de la cámara en el que debe estar el objeto. En la Figura 4.1.7 se pueden ver las diferentes zonas que tiene en cuenta la cámara virtual al hacer el seguimiento a un objeto:

- La zona roja o zona muerta, hará un movimiento inmediato para que el objeto salga de ella.
- La zona azul o zona blanda, hará un movimiento suave para que el objeto salga de ella.
- La zona sin color será considerada el centro de la cámara por lo que mientras el objeto permanezca ahí no se moverá la cámara.

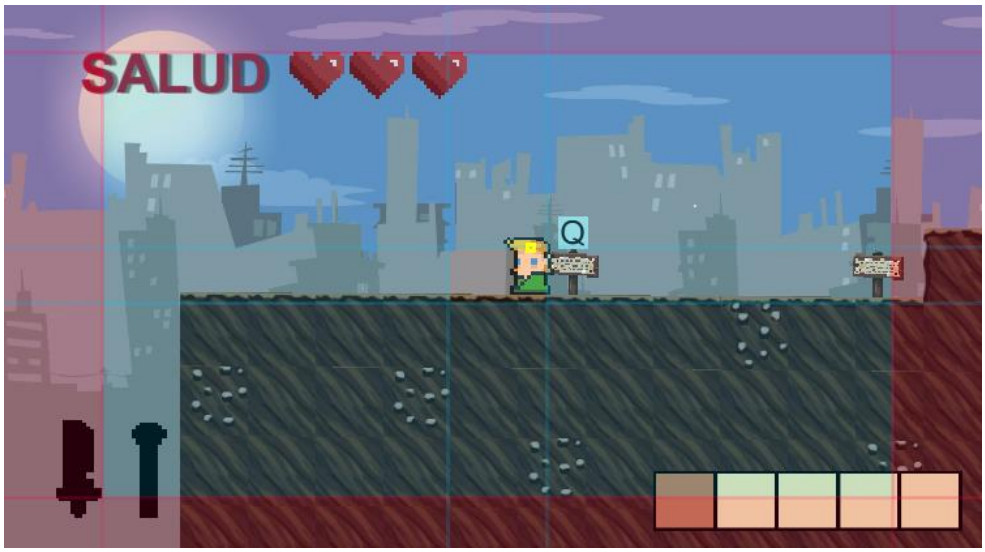


Figura 4.1.7 Distintas zonas de una cámara virtual

Otro detalle más a destacar sobre Cinemachine es que es posible crear todas las cámaras virtuales que se quieran así que, por ejemplo, en una pelea de jefe, podremos asignar una cámara secundaria al jefe apuntando a la sala donde vaya a ser la pelea y al comenzar el combate gestionar vía script el desactivar la cámara actual y activar la cámara asociada al jefe, haciendo un suave efecto de transición bastante pulido.

La última funcionalidad que se tratará sobre esta herramienta es la de Timeline, la cual nos permite hacer cinemáticas de una forma fácil intercalando movimientos de personajes, efectos de sonido, música y animaciones. Para ello, se crea un Timeline usando el clic derecho y se añaden distintos Tracks (Figura 4.1.8), cada uno asociado a un comportamiento en el tiempo, ya sea animación, movimiento de objeto, efecto sonoro o música, haciendo muy sencillo el intercalar sus apariciones durante la cinemática.

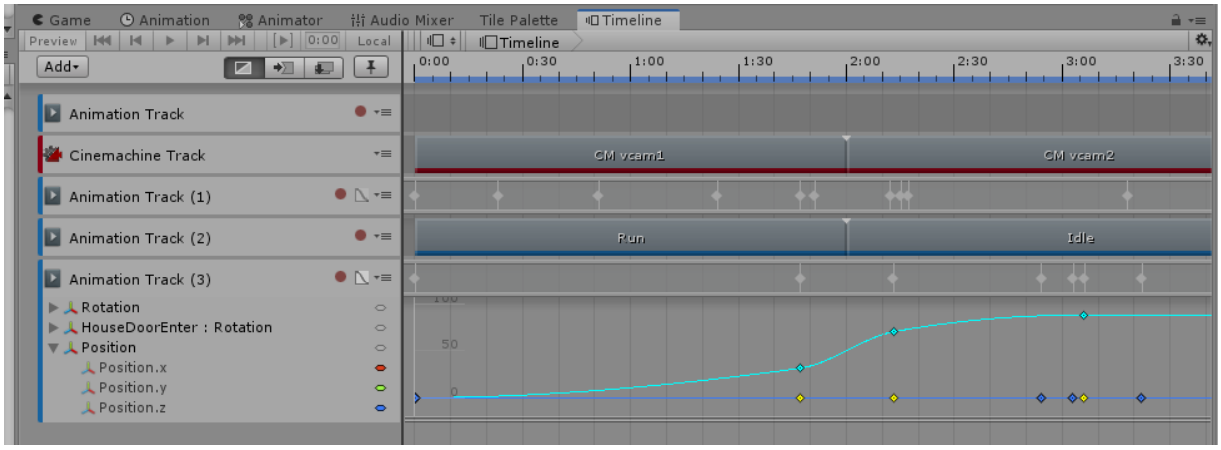


Figura 4.1.8 Timeline con varios Tracks asociados.

ANIMATOR

Esta herramienta se usa para el manejo de las animaciones asociadas a los objetos de una escena (Unity Animator, 2019). En dos sencillos pasos se podrá animar cualquier objeto deseado a partir de los sprites que queramos en la animación.

En los videojuegos, los efectos de que, por ejemplo, un personaje esté corriendo se consigue igual que en las series animadas, concatenando imágenes de forma lo suficiente rápida como para simular el movimiento. Estas imágenes se llaman sprites en el mundo de los videojuegos y suelen organizarse en sprite sheets, es decir, hojas en las que se encuentran todos los sprites que se necesiten para hacer una animación (Figura 4.1.9).

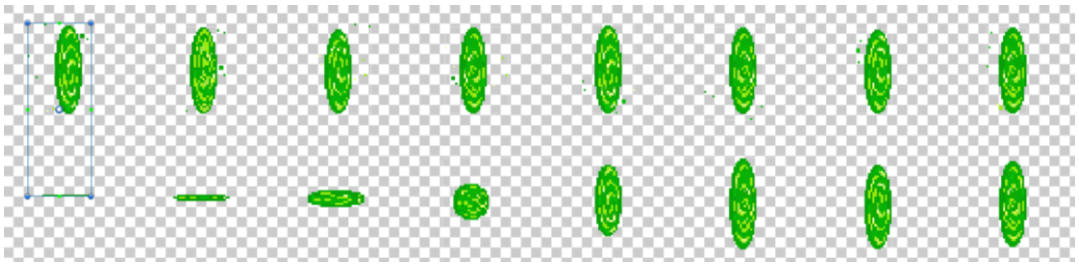


Figura 4.1.9 Spritesheet de animación de un portal

Para obtener los sprites de una hoja, hay que trocear la hoja por los puntos exactos para no coger ni más ni menos de lo que ocupa cada uno de los sprites.

Ahora bien, una vez se tienen los sprites, hay que asignar al objeto a animar un componente llamado animator y seguidamente en el submenú de Animation, se arrastrarán los sprites y se establecerán en que momento se harán los cambios entre ellos (Figura 4.1.10).

Figura 4.1.10 Asignación de los tiempos de transición de sprites de una animación hecha con la herramienta Animator.

Animator, también permite definir diagramas de estados para gestionar las transiciones entre distintas animaciones de forma automática. Para cambiar entre estados se definen parámetros que serán leídos y sobrescritos en los scripts del correspondiente objeto.

Se pueden observar la variedad de animaciones a las que se puede llegar usando esta funcionalidad en la Figura 4.1.11. Estas animaciones se mostrarán en función de los dos parámetros de la izquierda: speed y ground. Por ejemplo, para mostrar una animación de correr, el personaje deberá tener una velocidad distinta a cero y estar en el suelo.

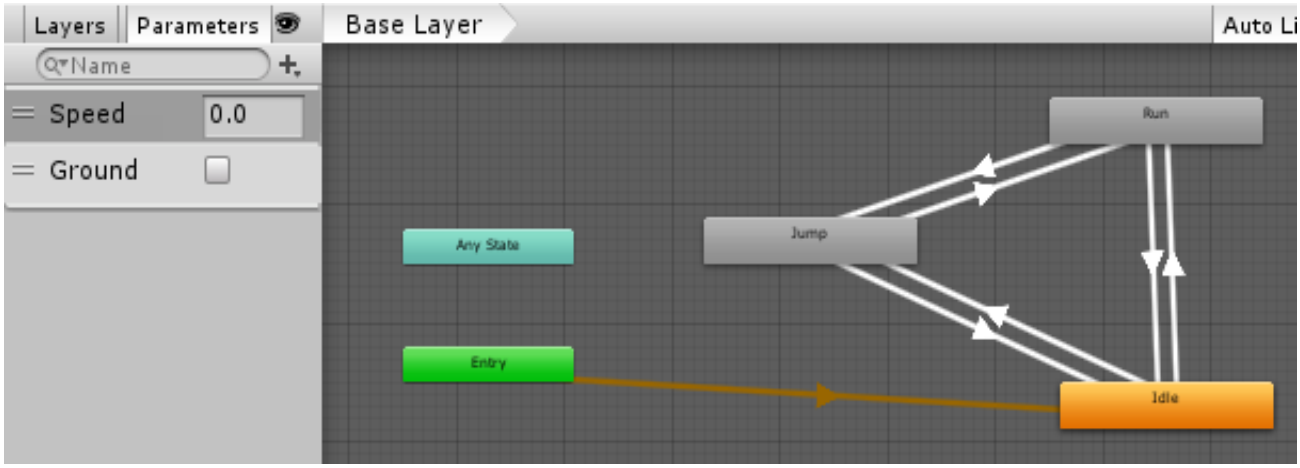


Figura 4.1.11 Diagrama de estado de las animaciones del protagonista del juego.

CANVAS

El Canvas es el objeto que se encarga de recoger todos los elementos de interfaz de usuario de un videojuego en Unity (Unity Canvas, 2019). Por ello, debe ser padre de todos los elementos de la interfaz del juego.

Si a la hora de crear un elemento de interfaz desde el inspector no existiera un Canvas se creará automáticamente uno y se hará al nuevo elemento hijo de este Canvas. En la Figura 4.1.12 se muestran todos los elementos asociados al Canvas del primer nivel.

Figura 4.1.12 Estructura del Canvas del nivel 1

Por tanto, todos los elementos de interfaz del proyecto se han creado siguiendo estas pautas. Estos elementos son tales como inventario del jugador, indicador de salud (Figura 4.1.13) y menús (Figura 4.1.14)

Figura 4.1.13 Interfaz de usuario del juego

Figura 4.1.14 Menu de pausa creado usando el Canvas

Ahora bien, aún siendo el Canvas una herramienta de Unity, la navegación entre los distintos menús pasarán a explicarse en el Capítulo 4.1.2 ya que sus transiciones y las opciones que nos ofrecen han sido programadas mediante scripts en C# asociados a elementos del Canvas.

PREFABS.

Los Prefabs son una de las funcionalidades de Unity más potentes y usadas de todas (Unity Prefabs, 2019). Permiten almacenar instancias de objetos de juego para poder usarla luego todas las veces que quieras simplemente arrastrándolo desde el inspector hacia la escena.

Estos Prefabs mantienen todos los valores, scripts, hijos y componentes asociados que tuvieran en el momento de ser guardados (a excepción de las relaciones con objetos que no sean hijos propios). En este proyecto, se ha hecho un uso bastante continuo de esta funcionalidad llegando a tener unos 46 Prefabs en total (Figura 4.1.15) separados en carpetas según su funcionalidad, teniendo así carpetas con consumibles, enemigos, menús, el propio personaje protagonista, etc.

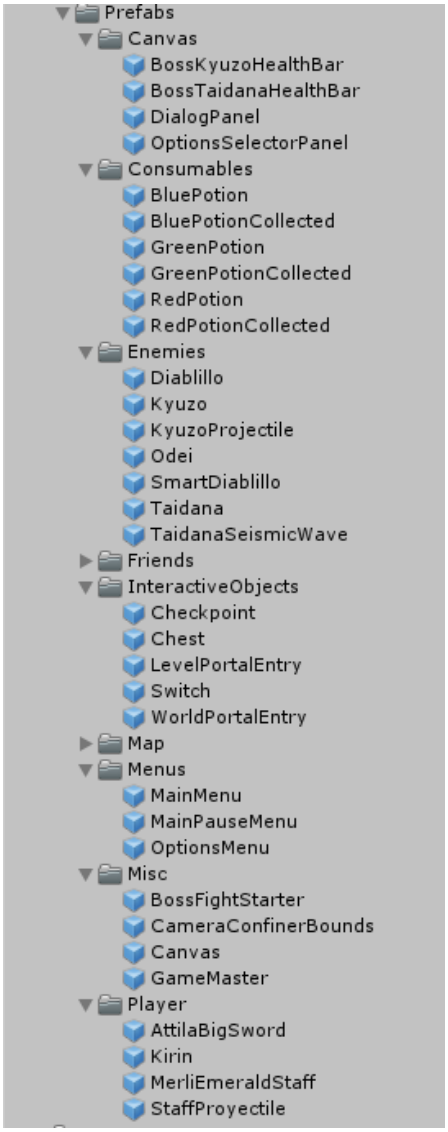


Figura 4.1.15 Jerarquía de Prefabs creados para el proyecto

Para terminar de comprender la potencia de esta herramienta, es posible, por ejemplo, crear 3 pociones en 2 segundos arrastrando 3 veces el Prefab a la escena (Figura 4.1.16).

La facilidad de creación de objetos con el uso de Prefabs dependerá de la dependencia que tenga con el resto de los objetos de la escena, ya que, tal y como se explicó al principio la sección, las relaciones con objetos no hijos del Prefab se pierden y habrá que establecerlas de nuevo.

Figura 4.1.16 Creación de objetos idénticos mediante el uso del Prefab de poción verde

Además, otra de los poderosos usos de los Prefabs es la posibilidad de poderlos crear dinámicamente desde un script usando para ello la ruta del Prefab dentro del proyecto. Por ejemplo, para crear una poción verde de forma dinámica al abrirse un cofre se llamará a la siguiente función (si el Prefab se llama GreenPotion y se encuentra en la carpeta de consumibles):

```
Resources.Load("Prefabs/Consumables/GreenPotion",typeof(GameObject));
```

Dicho todo esto, quedan demostradas las ventajas de usar Prefabs en Unity, además que condiciona un diseño del juego en el que los componentes estén lo más aislados posible, lo cual ayuda a cumplir muchos de los principios de la programación orientada a objetos, a la cual pertenece el lenguaje C# (Wikilibros, 2019).

Capítulo 4.1.2 Uso y desarrollo de scripts en C# dentro de Unity

Habiendo ya explicado todas las herramientas del entorno de desarrollo de Unity usadas, se pasará con los detalles de la implementación de los diferentes comportamientos conseguidos en el proyecto. Para facilitar la comprensión de la explicación se suministrará en cada subsección el diagrama de clases UML (Víctor Gómez, 2015) que representa la relación entre los scripts implementados para ese comportamiento.

PLATAFORMAS ESPECIALES

Continuando la sección de TilePalette del Capítulo 4.1.1 se explicarán las implementaciones de las plataformas con comportamientos especiales por script. Cabe a destacar que finalmente llegaron a implementarse las tres plataformas que fueron planificadas la fase de diseño.

La plataforma temporal estará anclada en el aire hasta que entra en contacto con el jugador y pasa cierto tiempo (variable delay). La plataforma también reaparecerá en su posición inicial al pasar un tiempo de reaparición (variable respawn). Para conseguir este funcionamiento cuenta con 4 funciones (Figura 4.1.17):

- La función Start, usada para la inicialización de las variables del gameObject.
- La función OnCollisionEnter2D que detectará la colisión con el jugador.
- La función Fall que deja caer la plataforma desde donde está.
- La función Respawn que hace reaparecer la plataforma en la posición de la variable start.

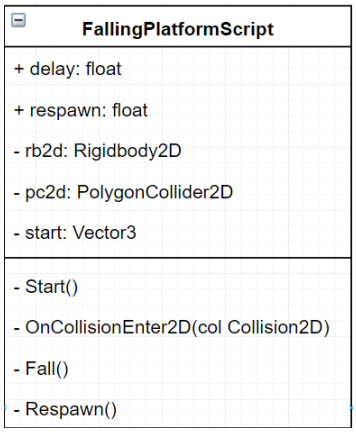


Figura 4.1.17: Diagrama UML de la implementación de las plataformas temporales

La plataforma móvil permanecerá inmóvil hasta que el jugador haga contacto con ella. A partir de ese momento, esta se moverá entre la posición de salida y la posición especificada en la variable target de forma indefinida. Para conseguir este funcionamiento cuenta con 5 funciones (Figura 4.1.18) :

- La función Start, usada para la inicialización de las variables del gameObject.
- La función Update, usada para el movimiento permanente de la plataforma una vez se activa.
- La función OnCollisionEnter2D que detectará la colisión con el jugador e iniciará el movimiento de la plataforma, haciendo al jugador también hijo de la plataforma para que se muevan a la misma vez.
- La función OnCollisionStay2D hará que cualquier objeto que no sea el jugador y esté en la plataforma no pueda moverse.
- La función OnCollisionExit2D que desanclará al jugador como hijo de la plataforma.

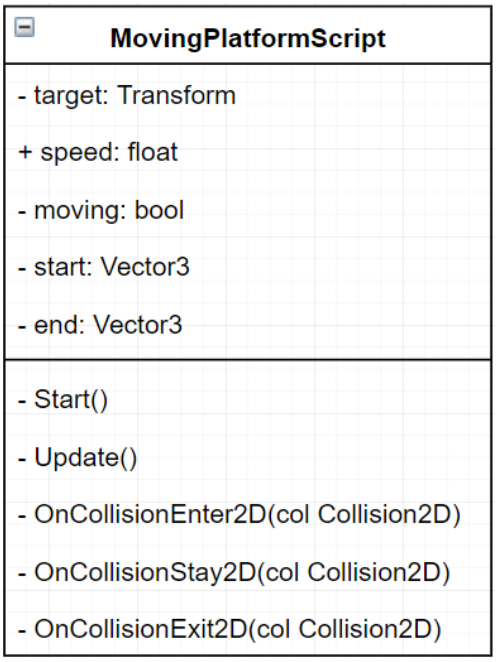


Figura 4.1.18: Diagrama UML de la implementación de las plataformas móviles

Por último, la plataforma unidireccional ha sido implementada usando un componente llamado PlatformEffector2D, el cual define un rango de ángulos en los cuales la plataforma detectará colisiones representado en la Figura 4.1.19 como un sector circular azul.

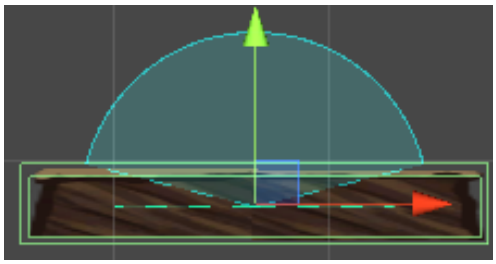


Figura 4.1.19: Uso de PlatformEffector2D en una plataforma unidireccional.

Además, también tiene un script asociado que permite al jugador atravesarla hacia abajo si se mantiene la tecla de flecha hacia abajo. Para conseguir este funcionamiento cuenta con 4 funciones (Figura 4.1.20):

- La función Start, usada para la inicialización de las variables del gameObject.
- La función Update, que controlará la pulsación de la tecla de bajar cuando el jugador esté ene contacto con la plataforma (este efecto de atravesar la plataforma se lleva a cabo invirtiendo el arco azul de la Figura 4.1.19 añadiéndole 180 grados de rotación) .
- La función OnCollisionEnter2D que detectará la colisión con el jugador y habilitará el pulsar el botón de bajar para atravesar la plataforma.
- La función OnCollisionExit2D que cuando el jugador abandone la plataforma y deshabilitará el pulsar el botón de bajar para atravesar la plataforma.

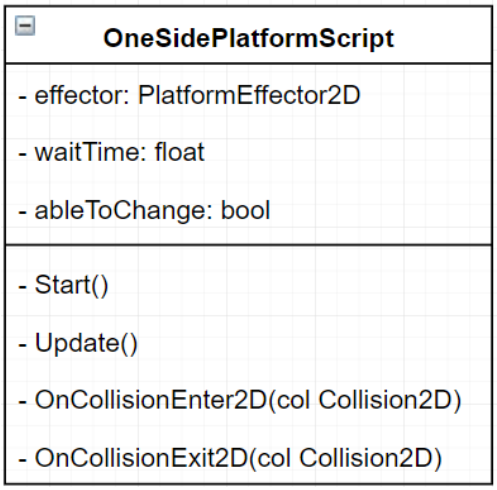


Figura 4.1.20: Diagrama UML de la implementación de las plataformas unidireccionales

MENÚS

Continuando la sección de Canvas del Capítulo 4.1.1 se explicarán las implementaciones de los menús y se mostrará un diagrama de flujo (Figura 4.1.21) que documente la navegación entre los mismos.

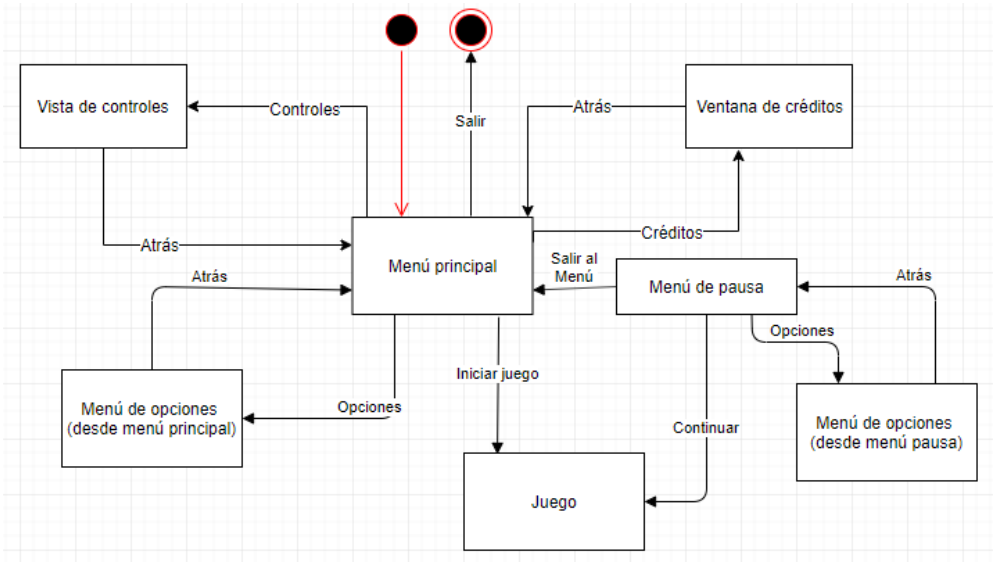


Figura 4.1.21: diagrama de flujos de los menús.

En el juego se tienen 5 pantallas de menú diferentes, una que toma el papel de menú principal y se muestra nada más acceder al juego, otra que sirve como menú de pausa y se abrirá cuando el jugador pulse la tecla de escape en cualquier momento de la partida, el submenú que muestra las opciones del juego, la ventana de créditos y la vista de ayuda de controles.

El menú inicial cuenta con 5 botones, uno para iniciar el juego desde el último punto de guardado, otro para abrir las opciones, otro para cerrar el juego y en las dos esquinas inferiores los botones para abrir los créditos y los controles (Figura 4.1.22).

Figura 4.1.22: Menú de inicio del juego.

El menú de pausa cuenta con 3 botones, uno para reanudar el juego y cerrar el menú, otro para abrir las opciones y otro para volver al menú principal (Figura 4.1.23).

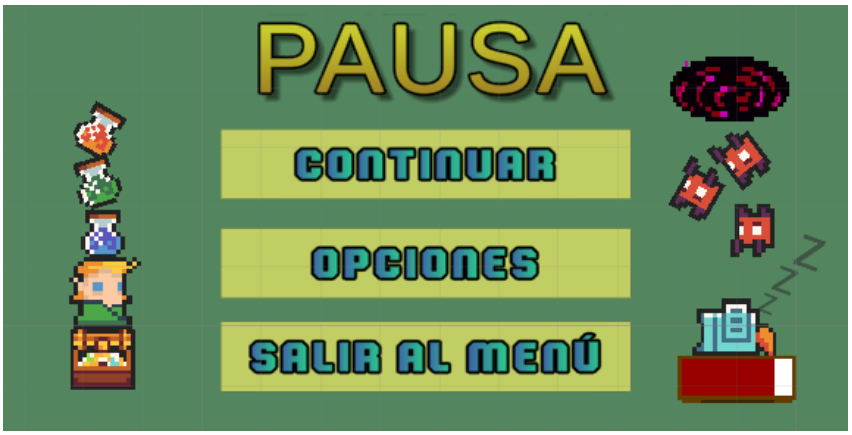


Figura 4.1.23: Menú de pausa del juego

El submenú de opciones abierto por ambos menús es el mismo a excepción de que el fondo variará en función del menú en que nos encontremos (Figura 4.1.24). Este menú permitirá al usuario cambiar el volumen maestro del juego por medio de una barra deslizante y modificar la resolución del juego usando una lista desplegable de resoluciones. También cuenta con un botón de atrás para cerrar el submenú.

Figura 4.1.24: Menú de opciones

El cambio de las características del juego en función de lo seleccionado en el menú se llevó a cabo registrando controladores de cambios (o listeners) a los componentes del Canvas (desplegable y barra deslizante) usando acciones delegadas (conocidas también como callbacks). Esto se puede ver en la siguiente instrucción, en la que se le asigna un listener que, cuando detecta un cambio de

```
slider.onValueChanged.AddListener(delegate { OnVolumeChange(); });
```

valor en la barra deslizante, llama a la función OnValueChange.

La modificación de las resoluciones se implementó en un script tomando todas las resoluciones para la pantalla en la que se ejecuta el juego, se muestran las opciones en el desplegable y una vez se selecciona una se asigna la nueva resolución mediante la siguiente instrucción:

```
Screen.SetResolution(resolutions[resolutionsDropdown.value].width,
resolutions[resolutionsDropdown.value].height,
true);
```

Por aclarar un poco lo que sucede en el fragmento de código, se toma el valor seleccionado de la lista, se le asigna esa altura y anchura a la pantalla y se indica con un booleano a verdadero que el juego se siga mostrando en pantalla completa.

En el caso de modificar el volumen, se toma el valor de la barra deslizante y se multiplica por 100 (internamente la barra deslizante toma valores de 0 a 0.1) y se

asigna al Valor de volumen general mediante la instrucción especificada

```
AudioListener.volume = slider.value;
```

seguidamente:

La ventana de créditos expone la finalidad del proyecto desarrollado y presenta a los creadores del juego (Figura 4.1.25).

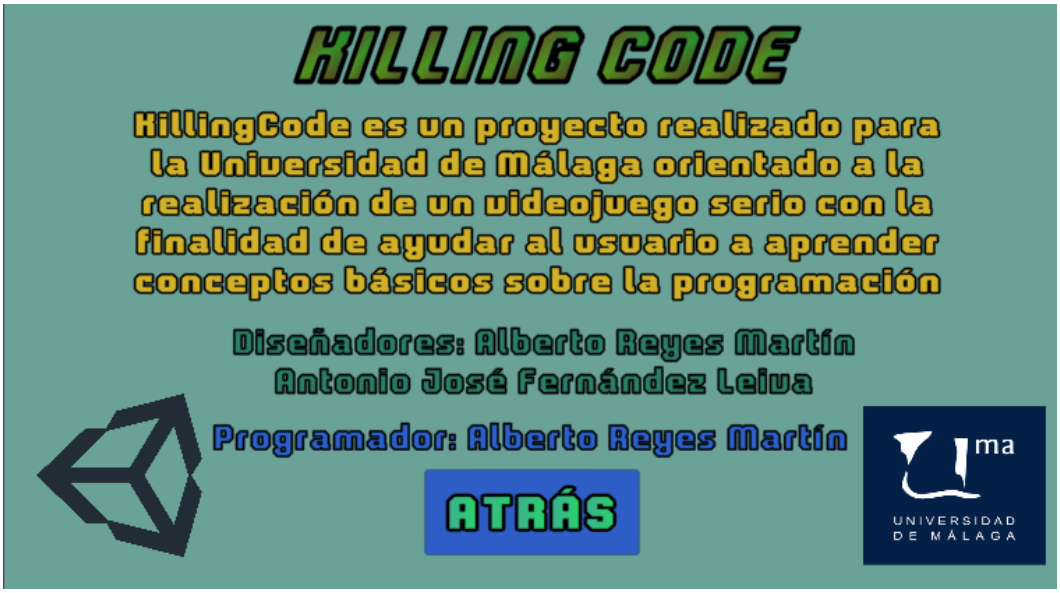


Figura 4.1.25: Ventana de créditos

Por último, la vista de los controles expone todas las teclas que el jugador podrá usar durante el juego y que comportamiento tienen asociadas (Figura 4.1.26).

Figura 4.1.26: Vista de los controles

INVENTARIO

Aún estando plasmado en el GDD como una idea secundaria que no se sabía si finalmente se llegaría a implementar, llegó a añadirse con éxito la funcionalidad de un inventario (Figura 4.1.27).

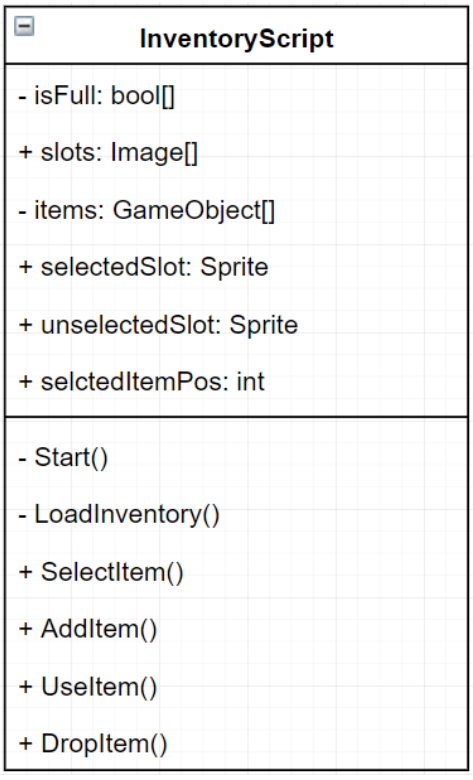


Figura 4.1.27: Diagrama UML de la implementación del comportamiento del inventario.

Este inventario permite al jugador tomar pociones que encuentre en los niveles y poder usarlas cuando quiera pulsando la tecla E (función UseItem) para obtener salud perdida o soltar la poción pulsando la tecla R (función DropItem).

El inventario cuenta con 5 huecos en los cuales se alojará una imagen de la poción guardada en ese lugar. Para seleccionar entre los distintos objetos guardados, se deberán pulsar las teclas de la 1 a la 5 (función SelectItem), las cuales están relacionadas con su respectivo espacio de inventario.

Los huecos de inventario están pintados en el Canvas (variable slots del script del inventario) de la escena del nivel y en el momento en que se hace contacto con una poción (sea cual sea el tipo de poción), se llama al script asociado a la misma (Figura 4.1.28) y, si hay algún hueco libre en el inventario, llamará a la función AddItem del script de inventario del jugador y destruirá la poción, pintando la poción recién recolectada en su lugar del Canvas (Figura 4.1.29).

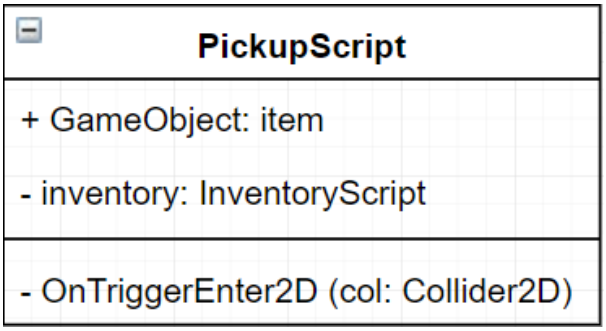


Figura 4.1.28: Diagrama UML de la implementación del comportamiento de las pociones.

Figura 4.1.29: Inventario con 4 pociones guardadas.

SALUD

El sistema de salud del personaje se basa en una serie de corazones pintados en la interfaz que pueden tener dos estados (Figura 4.1.30) :

- Negro: Se ha perdido ese punto de salud de alguna forma y se necesita una poción para restaurarlo.
- Rojo: El jugador tiene disponible ese punto de salud.

Figura 4.1.30: Interfaz de usuario de salud.

El script que se encarga de gestionar las variaciones de salud del personaje y actualiza los corazones en la interfaz es el PlayerHealthScript (Figura 4.1.31) y está asignado al propio personaje.

En el momento en que el jugador tenga todos los corazones en negro o se caiga del mapa, se recargará la escena (función KillPlayer) haciendo aparecer al personaje en el último punto de control, o en su defecto al inicio del nivel. Este comportamiento queda implementado en un PlayerSpawnScript asociado al jugador (Figura 4.1.32).

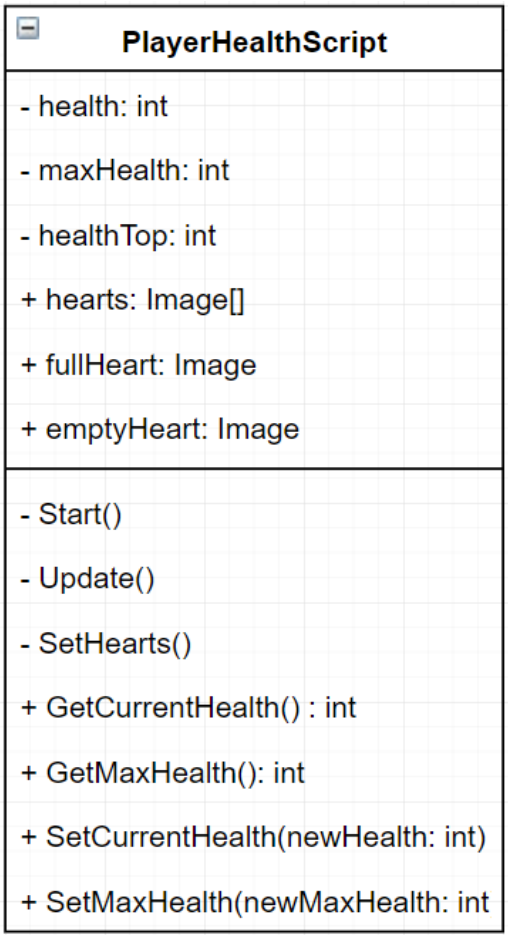


Figura 4.1.31: Diagrama UML de la implementación del sistema de salud del jugador

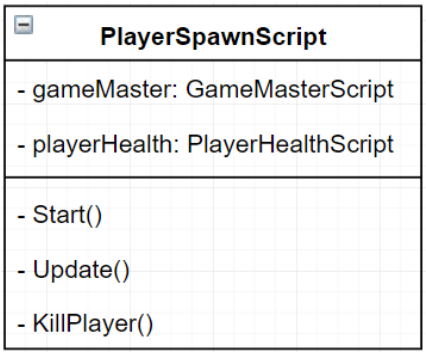


Figura 4.1.32: Diagrama UML de la implementación del sistema de reaparición del jugador.

GAMEMASTER

En la Figura 4.1.32 se puede observar como hay una variable de tipo GameMasterScript. Este GameMasterScript está relacionado con una de las partes más fundamentales y complejas del proyecto, *el sistema de guardado de la partida.*

Este sistema de guardado se basa en un objeto que hace de maestro del juego y coordina todas las cargas de escenas y guardado de ciertos valores cuando se van a recargar los niveles, ya sea producido por haber muerto o por lanzar el juego desde el menú principal. Este comportamiento es implementado por el GameMasterScript (Figura 4.1.33).

Actualmente, se guardan los valores de salud máxima del jugador, salud del jugador en el momento en que se guardó la partida, habilidades desbloqueadas, punto de reaparición, nivel a cargar y objetos del inventario, tal y como se pueden ver en la clase Game de la Figura 4.1.33.

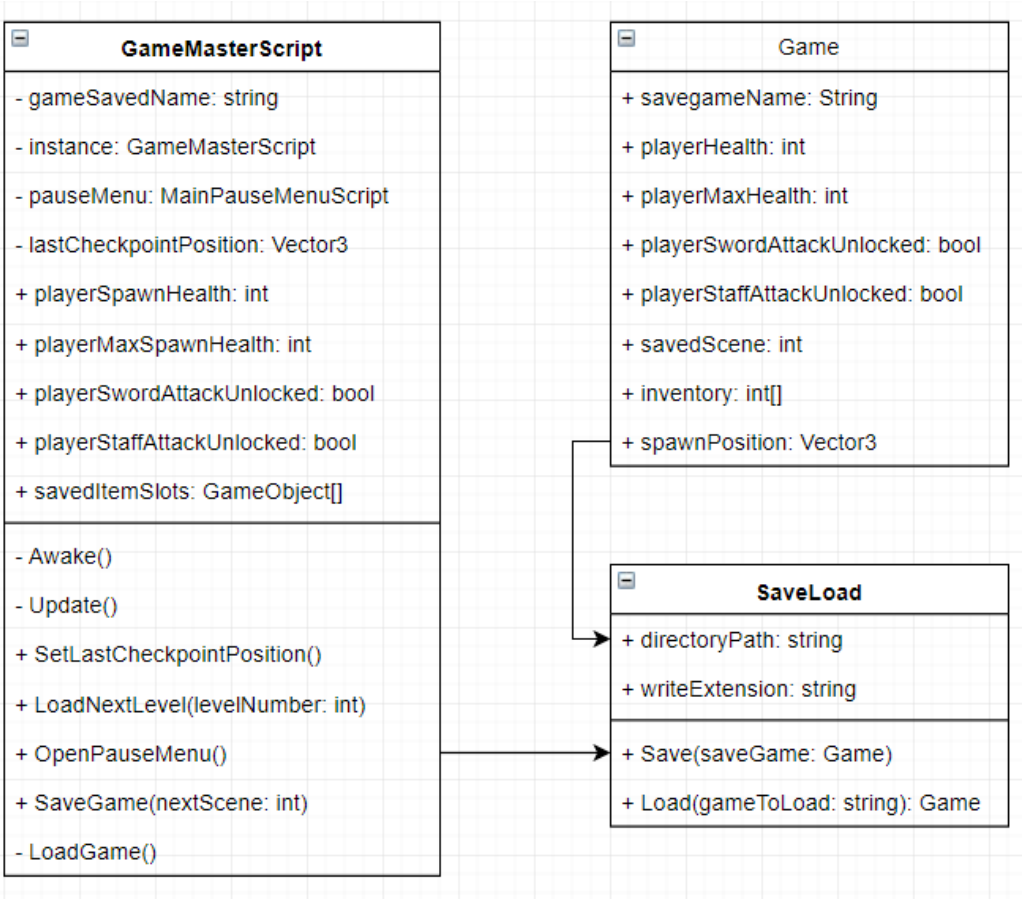


Figura 4.1.33: Diagrama UML de la implementación del sistema de guardado

Cuando se hace contacto con un punto de control o un portal que lleva a otro nivel, se llama en primer lugar al método SaveGame del GameMasteScript y luego, si es el caso del portal, se llamará al método LoadNextLevel pasando como

parámetro el número de escena a cargar (esto se explicará más a fondo en la subsección dedicada a los portales de este mismo Capítulo).

El guardado se lleva a cabo en el método Save del script SaveLoad según se muestra en el fragmento de código siguiente. En ese método se lleva a cabo escribiendo en un fichero (que se crea al momento si no existiera partida guardada) un objeto de tipo Game serializado(DevelopMania, 2009).

```
      BinaryFormatter bf = new BinaryFormatter();

       //Sets vector3 serializer
       SurrogateSelector ss = new SurrogateSelector();
       Vector3SerializationSurrogate v3ss = new Vector3SerializationSurrogate();
       ss.AddSurrogate(typeof(Vector3),
            new StreamingContext(StreamingContextStates.All), v3ss);
       bf.SurrogateSelector = ss;

       FileStream file = File.Create(directoryPath + saveGame.savegameName +
writeExtension);
       bf.Serialize(file, saveGame);
       file.Close();
```

Para hacer la lectura del fichero, se hace el mismo proceso, pero a la inversa, comprobando en primer lugar si existe el fichero que se va a intentar leer. A continuación, se deserializa el objeto creado dentro del fichero y se genera una variable de tipo Game. El gameMaster toma este Game que ha generado la carga de la partida y sustituye sus propios valores por los del objeto deserializado. El fragmento de programa siguiente muestra el proceso indicado:

```
if (File.Exists(directoryPath + gameToLoad + writeExtension))
        {
            BinaryFormatter bf = new BinaryFormatter();
            //Sets vector3 serializer
            SurrogateSelector ss = new SurrogateSelector();
            Vector3SerializationSurrogate v3ss = new
Vector3SerializationSurrogate();
            ss.AddSurrogate(typeof(Vector3), new
StreamingContext(StreamingContextStates.All), v3ss);
            bf.SurrogateSelector = ss;
            FileStream file = File.Open(directoryPath + gameToLoad +
writeExtension, FileMode.Open);
            Game loadedGame = (Game)bf.Deserialize(file);
            file.Close();
            return loadedGame;
        }
        else
        { return null; }
```

Por último, cabe destacar qué hace diferente al objeto que tiene el GameMasterScript asociado del resto de los objetos del juego y por qué cuenta con una instancia de sí mismo como variable. Esto se debe a que este script sigue un patrón de instancia única (Albert Capdevila, 2017) combinado con una de las funciones propias de la librería de Unity DontDestroyOnLoad, el cual evita que el gamemaster sea destruido cuando se recarguen las escenas.

PUNTOS DE CONTROL

Tal y como se mencionó en el apartado dedicado al gamemaster, se implementaron también en el proyecto los puntos de guardado. Estos tienen dos estados (Figura 4.1.34) tal y como se especificó en el GDD, activado (color verde) y desactivados (color rojo).

El punto de control se encuentra por defecto desactivado y una vez se activa no vuelve a encontrarse en estado desactivado hasta que el jugador no active otro punto de control en el mismo nivel (no se permiten tener dos puntos de control diferentes activados).

El punto de control se activa en el momento en el que el jugador hace contacto con ellos y se encuentran desactivados. Esta activación implicará que se llame a la función de guardado del GameMasterScript.

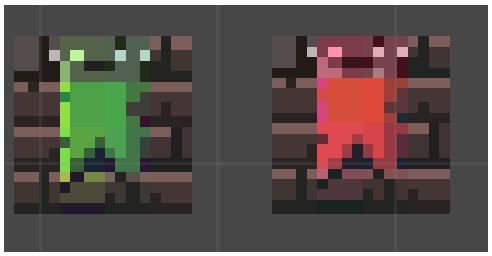


Figura 4.1.34: Estados de los puntos de control

PORTALES

Los portales son objetos que en el juego se usan para servir de puente entre los niveles (Figura 4.1.35). En el momento en el que el jugador toca el portal, se produce una pausa de un segundo para simular el proceso de cargado y se llama a función de guardado del GameMasterScript, además de a la función que le hace cargar una determinada escena según un valor entero.

Figura 4.1.35: Portal en el juego

Las escenas se ordenan en función de un índice numérico. Este índice puede consultarse si se pulsa sobre File y luego sobre Build Settings (Figura 4.1.36). Así, por ejemplo, el índice necesario para cargar el nivel llamado JungleLevel será el 5.

Figura 4.1.36: Índices de las diferentes escenas de un proyecto

INTERRUPTORES

Los interruptores (Figura 4.1.37) dan mucho juego a la hora de crear puzles en los niveles ya que, en este proyecto, pueden tener una de entre tres funciones distintas.

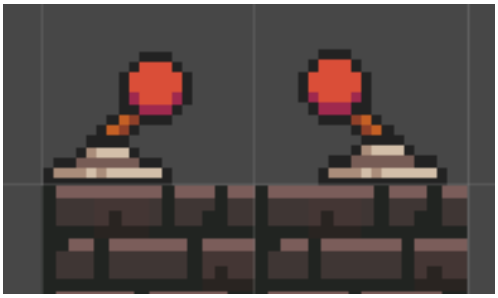


Figura 4.1.37: Estados de los interruptores. A la izquierda activado, a la derecha desactivado.

Los interruptores no pueden desactivarse una vez se activan y cuando el jugador está lo suficientemente cerca como para activarlos, se mostrará un recuadro

con la letra Q indicando que para interactuar con ellos es necesario pulsar dicha tecla (Figura 4.1.38).

Figura 4.1.38: Ayuda del interruptor mostrándose al estar el jugador cerca

Los interruptores por tanto tienen un SwitchScript (Figura 4.1.39) que gestiona el cambio de aspecto al activarse, la aparición de la pista cuando el jugador está cerca y los diferentes comportamientos que puede tener su activación.

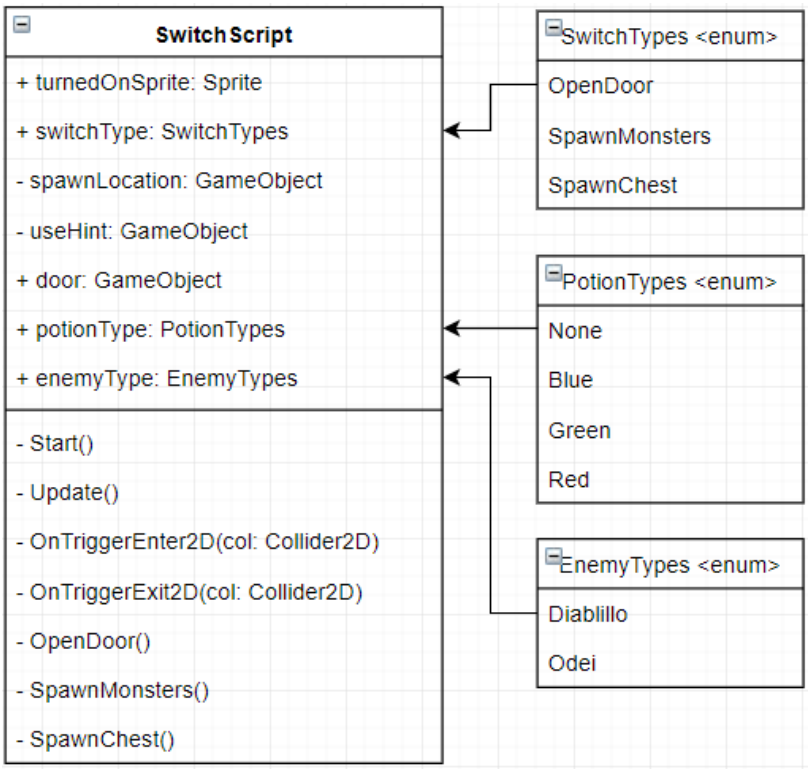


Figura 4.1.39: Diagrama UML de la implementación de los interruptores.

Algo a destacar de este script es que todos sus métodos son privados, ya que se configuró de forma que una vez seleccionado un valor para la variable de tipo enumerado switchType (Figura 4.1.40), el script automáticamente seleccionará la acción correspondiente llamando a la función OpenDoor (hace desaparecer el gameObject especificado para simular la apertura de una puerta), SpawnMonsters

(hace aparecer a un enemigo, en el punto especificado, del tipo seleccionado como enemyType por medio del nombre de su Prefab asociado) o SpawnChest (hace aparecer un cofre en el lugar especificado con una poción en su interior del tipo especificado como potionType).

Figura 4.1.40: En Unity, los tipos enumerados de visibilidad pública muestran sus posibles valores mediante un desplegable.

Ya que para explicar el funcionamiento de los interruptores hemos hablado de los enemigos y los cofres, se procederá a continuación a explicar sus implementaciones.

COFRES

Los cofres tienen un script asociado (Figura 4.1.41) que gestiona su animación de apertura, la aparición de una pista con la letra Q sobre él y la generación de la poción correspondiente al activarlos.

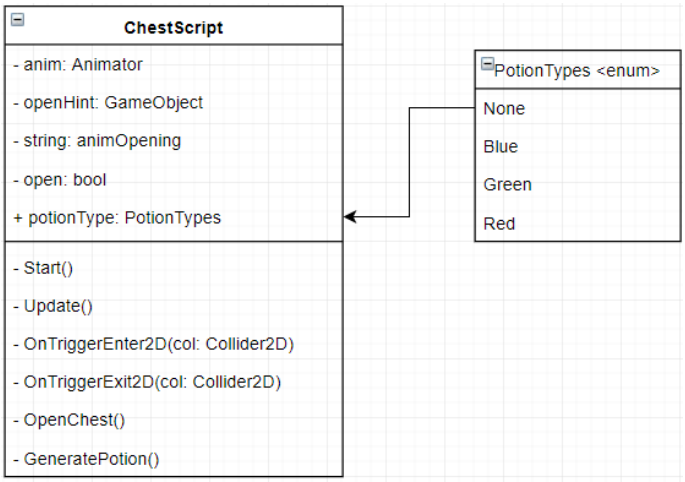


Figura 4.1.41: Diagrama UML de la implementación de los cofres

Este script funciona de forma muy similar al de los interruptores. Solo se tiene una variable de visibilidad pública, debido a que se programó el componente para que, en función del valor seleccionado como potionType, se generara esa poción tras pulsar la Q estando cerca del cofre. Además, se configuró una corrutina (Antonio Moon, 2019) que al cabo de 1.5 segundos (para dar tiempo a que se complete la animación de apertura del cofre) hace aparecer la poción sobre el cofre (Figura 4.1.42).

Figura 4.1.42: Cofre haciendo aparecer una poción roja.

ENEMIGOS

Los enemigos son un punto clave de un videojuego como este, pues permiten al jugador sentirse aún mas desafiado al tener unos objetivos a los que derrotar. Esto hace que el tema del desafío intelectual que proporciona un videojuego serio pueda ser complementado con el desafío de coordinación al tener que derrotar multitud de enemigos diferentes. Desafortunadamente no dio tiempo a implementar a todos los enemigos que se plasmaron en el GDD y se completaron la implementación de 2 enemigos básicos y 2 de los jefes. Aun así, se hablará de la evolución futura del proyecto en el Capítulo 5 y si se llegarán a implementar o no.

Dicho esto, se explicarán, por un lado, los comportamientos de los enemigos base y, por otro, el de los jefes, debido a sus comportamientos tan diferenciados, además de usar scripts completamente diferentes.

ENEMIGOS BASE

Los enemigos base se limitan a moverse de un lado hacia otro del mapa y girarse si colisionan con algún elemento inamovible. Además, pueden asignárseles puntos de límites en los cuales el enemigo se girará al alcanzarlo. Esto puede utilizarse por ejemplo para mantener a un enemigo patrullando una plataforma sin caerse.

Su script asociado (Figura 4.1.43), también posee valores públicos para definir ciertos atributos del enemigo, tales como si los saltos le hacen daño, la cantidad de vida que tiene, cuanto le dañan los saltos, golpes de espada o bolas de magia y el daño que puede producirle al jugador con un ataque.

Además, si el jugador golpea por encima al enemigo y ese enemigo recibe daño de los saltos, el jugador de vera levemente impulsado hacia arriba (si el enemigo no recibe daño de los saltos, la colisión se resolverá con el jugador recibiendo daño igual al valor de ataque del enemigo).

Figura 4.1.43 : Diagrama UML de la implementación del comportamiento del enemigo base.

JEFES

Los jefes actúan de una forma algo más compleja que los enemigos normales para así proporcionar al usuario un desafío mayor para sus habilidades (Figura 4.1.44).

En primer lugar, podemos ver en el diagrama de clases 8 funciones para la clase KyuzoScript:

- La función Start, usada para la inicialización de las variables del gameObject.
- La función Update que se encarga de actualizar los estados de las animaciones (mediante la variable anim de tipo Animator) y de llamar a la función GenerateRancomBehaviour.
- La función GenerateRandomBehaviour se encarga de lanzar corrutinas cada cierto tiempo para que el jefe haga una de las acciones de las que dispone (andar, pararse y lanzar un ataque especial).
- La función ThrowProjectile instanciará un Prefab del proyectil del jefe (tal y como se explicó con la poción verde en la sección de prefabs del capítulo anterior) cuando lance su ataque especial. Este proyectil iniciará su animación de movimiento automáticamente, ya que cuenta con un script KyuzoProyectileScript asociado que gestionará tanto el daño que hará al jugador como la velocidad y la distancia que recorrerá antes de desaparecer.
- La función CheckHealth que actualizará la barra de vida restante del jefe y comprobara si ha llegado al umbral necesario para que se enfade y aumente la dificultad de la pelea.
- Por último, la función ActivateBoss será llamada por un hijo del jefe que detectará cuando entra el jugador en cierto rango. Este hijo tendrá asociado un BossFightScript y se encargará de iniciar el diálogo al acercarse el jugador, cambiar la vista usando 2 cámaras virtuales de Cinemachine y mostrar un diálogo al derrotarse al jefe.

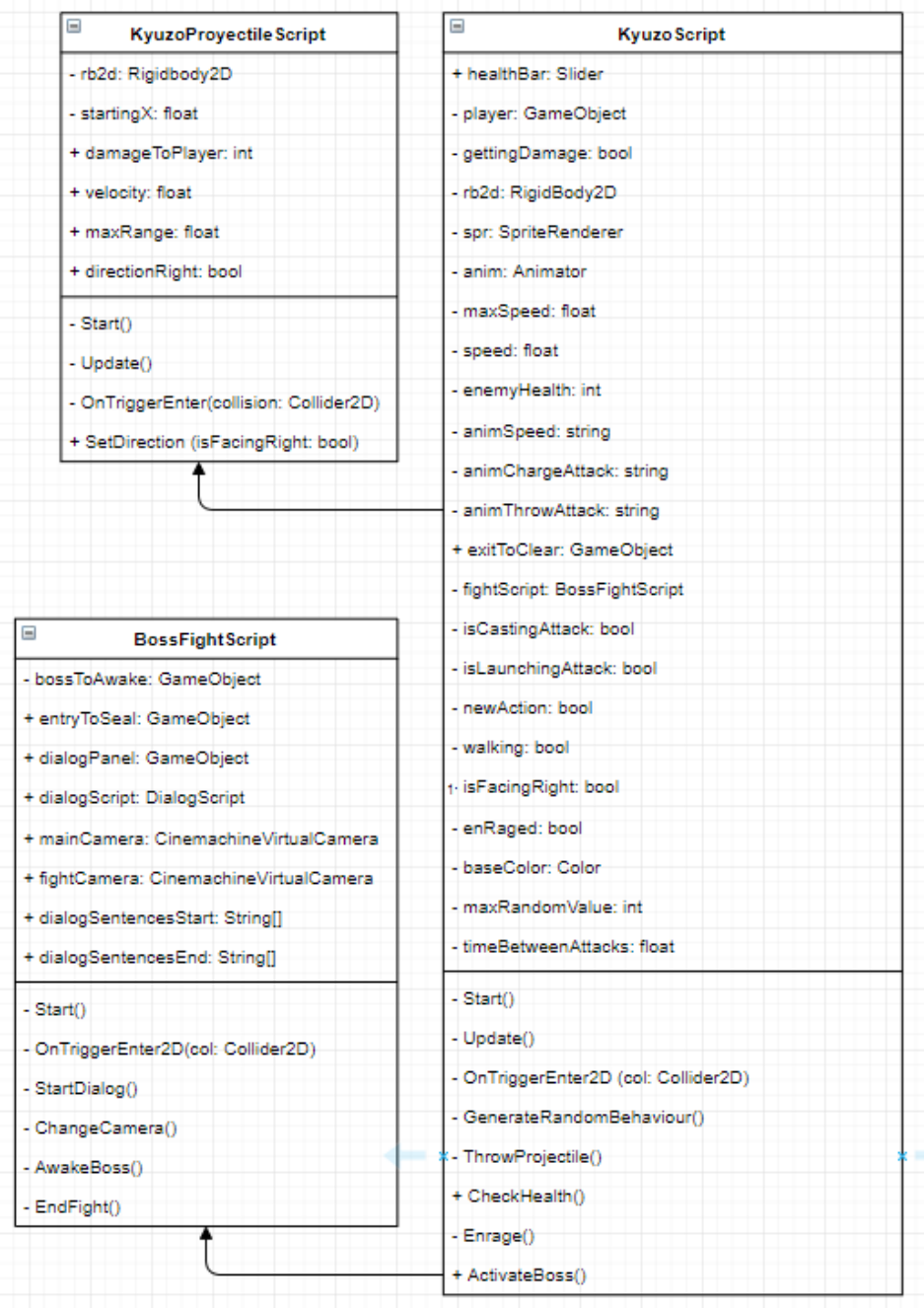


Figura 4.1.44: Diagrama UML de la implementación del comportamiento del jefe.

El otro jefe implementado también tiene asociado un proyectil y un hijo que cuenta con un BossFightScript, diferenciándose exclusivamente de la implementación de este jefe en que el ataque especial es un salto que al tocar el suelo hace aparecer dos proyectiles y que tiene 2 fases de enfado haciéndolo mucho más difícil de vencer.

Por ser las diferencias tan pequeñas a nivel de implementación no se ve necesario explicar de nuevo todo el comportamiento de este segundo jefe.

PROTAGONISTA

Ya que se han explicado todas las mecánicas del juego que envuelven al personaje controlado por el jugador se hace necesario explicar el propio funcionamiento del personaje. Este personaje tiene asociados un total de 5 scripts como se pueden ver en la Figura 4.1.45.

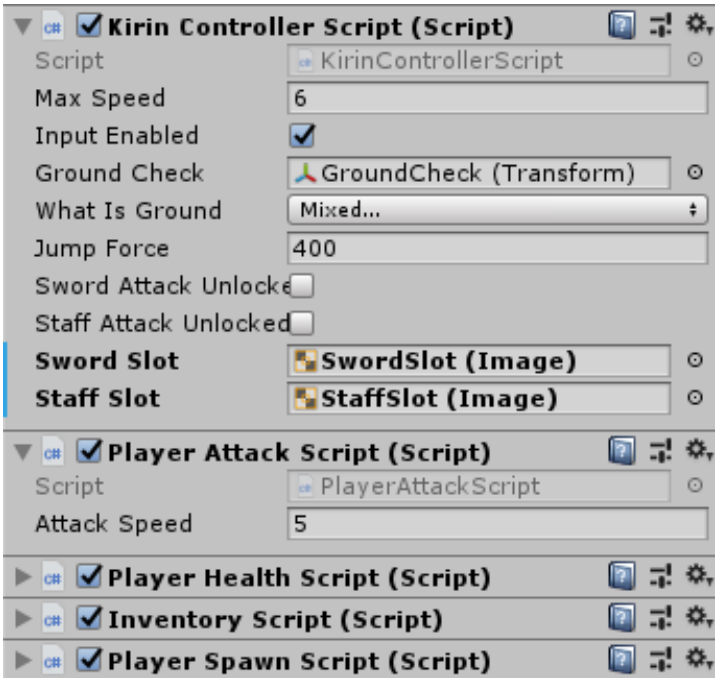


Figura 4.1.45 : Scripts asociados a Kirin, el personaje principal.

De estos 5 scripts, restan por explicar los dos primeros, ya que los tres últimos (referentes a la reaparición, la salud y al inventario) ya fueron explicados previamente.

Dicho esto, se comenzará explicando el script que permite al jugador atacar con la espada o con su bastón mágico (Figura 4.1.46). Estas dos habilidades mágicas se desbloquearán a lo largo de la historia y una vez desbloqueadas, con la tecla Z se hará un ataque frontal con un espadón y si se pulsa la tecla X se lanzará un proyectil mágico hacia adelante (Figura 4.1.47).

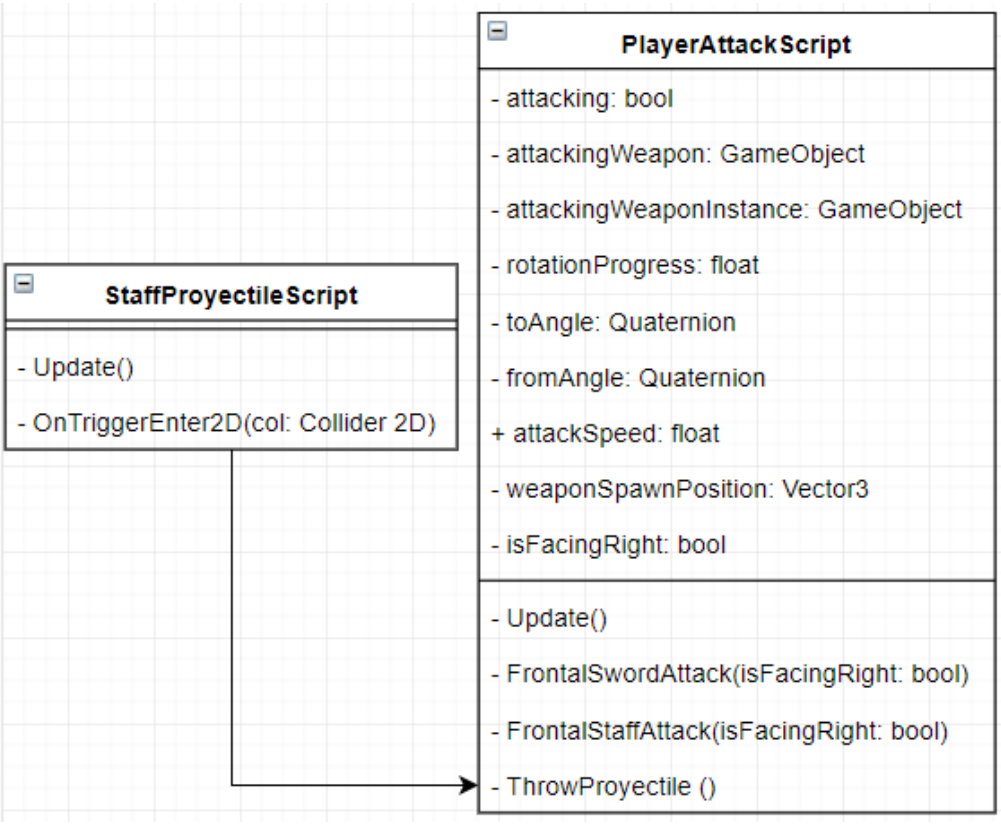


Figura 4.1.46 : Diagrama UML de la implementación del ataque del jugador.

Figura 4.1.47 : Ataques de Kirin. A la izquierda ataque de espada, a la derecha ataque mágico.

Al no disponer de los sprites necesarios para hacer la animación de ataque con la espada, se ha tenido que simular la animación haciendo que el objeto rote sobre un eje al mostrarse y desaparezca al llegar a cierto ángulo.

En el caso del bastón, también se ha hecho que, al inicio de la rotación, aparezca un proyectil el cual tiene definido su comportamiento propio en un StaffProjectileScript.

El haber tenido que gestionar a mano esta simulación de la animación hizo evidente la comodidad que aporta la herramienta de Animator de Unity.

El otro script que falta por explicar sería el KirinControllerScript (Figura 4.1.48), el cual sirve como coordinador de todos los comportamientos del personaje que dependen de alguno de los otros 4 scripts. Además, gestiona las distintas animaciones del personaje, el movimiento, los saltos e incluso las colisiones con los enemigos que aplican daño sobre el jugador.

Figura 4.1.48: Diagrama UML de la implementación del controlador principal del jugador.

DIÁLOGOS Y CUESTIONARIOS

Llegando ya a la parte final de este Capítulo, se hablará de una de las partes más elaboradas y complejas de la solución implementada, el sistema de diálogos y cuestionarios. Esta parte de la implementación era crucial debido que, al tratarse de un juego serio, debía de haber alguna forma de interactuar con el usuario para, en primer lugar, poder hacer los cuestionarios que validaran el aprendizaje del jugador y, en segundo lugar, proporcionarle la información necesaria para estas pruebas teóricas.

La solución del sistema de diálogos se consiguió creando un gameObject en el Canvas con un script asociado que, suministrándole una cadena de cadenas de caracteres, fuera capaz de gestionar el pintado de las frases y mostrara el respectivo botón de continuar para avanzar en la conversación, cerrando el cuadro de diálogo automáticamente cuando se terminaran las frases por decir (Figura 4.1.49).

Figura 4.1.49: Ejemplo del sistema de diálogos del juego al interactuar con un cartel.

Para los cuestionarios se hizo un script similar al de los diálogos, pero algo más complejo, puesto que permite que se le suministren mensajes de introducción, mensajes para las preguntas, enteros para comprobar respuestas, mensajes de explicación de las respuestas, mensajes para cuando termine el cuestionario y la capacidad de dar al jugador corazones extra y habilidades como recompensas por completar el test (Figura 4.1.50).

Figura 4.1.50: Ejemplo del sistema de diálogos del juego al hacer un test.

Capítulo 4.2 Metodología de desarrollo aplicada

Ya habiendo cubierto toda la explicación de la implementación realizada, toca explicar otra parte igualmente importante y sin la cual el proyecto hubiera finalizado con una calidad muchísimo menor a la conseguida, se trata de la metodología usada en la gestión de las tareas y las prioridades de las funcionalidades a implementar.

En el caso de este proyecto, se ha aplicado una metodología scrum, en la cual se partió elaborando el GDD con el esbozo general de lo que había que implementar y seguido a ello comenzaron los sprints de la metodología.

Se acordó con el cliente (en este caso el tutor) tener reuniones en lapsos de tiempo no superiores a dos semanas en las cuales habría que exponer los cambios realizados en la etapa anterior, las inconveniencias que habían surgido y que funcionalidades nuevas debían ser implementadas para la siguiente reunión.

Siguiendo estas pautas, por lo general todas las etapas de desarrollo terminaron sin inconveniencia alguna logrando que se implementaran todas las funcionalidades solicitadas por el cliente.

A continuación, se adjuntarán los registros de las reuniones en las que se especificarán la versión entregada, los inconvenientes que surgieron en la etapa anterior, la fecha de la reunión y la funcionalidad a implementar para la nueva etapa de desarrollo.

Fecha	Versión	Funcionalidad para la siguiente etapa	Inconvenientes de la etapa completada
19/Junio	-	Movimiento y animaciones personaje principal, configuración TilePalette, crear sistema de cámaras con Cinemachine	-
2/Julio	0.1	Crear interfaz de jugador, menús, cofres, pociones, funcionalidad de inventario y funcionalidad de vidas	Algunos controles de colisiones se quedaban atrapados entre ellos, se pudo arreglar para antes de la entrega
12/Julio	0.2	Crear punto de control, portal, interruptor, plataformas especiales y gameMaster	-
24/Julio	0.3	Completar implementación de gameMaster, implementar los dos ataques del jugador y crear enemigo diablillo	No se fue capaz de configurar bien el gameMaster
9/Agosto	0.4	Pulir efecto ataques, crear primer jefe, crear sistema de diálogos y añadir funcionalidad al interruptor de generar enemigo	El “efecto de mover el arma” creado en los ataques no está del todo pulido
21/Agosto	0.5	Crear sistema de cuestionarios, crear personajes secundarios e implementar enemigo Odei	-
28/Agosto	0.6	Implementar segundo jefe, configurar el guardado en ficheros y añadir misiones secundarias al hablar con ciertos personajes secundarios.	Pequeños fallos en las interacciones con los personajes secundarios
4/Septiembre	1	-	-

Tabla 4.2.1: Registro de reuniones con el cliente

Como se puede ver en el registro de la Tabla 4.2.1, esta planificación en etapas consiguió que se pudieran alcanzar los objetivos esperados sin problemas, demostrando que la aplicación de esta metodología ágil Scrum permite una organización eficaz de los recursos y arroja muy buenos resultados en proyectos de pequeña, mediana y gran envergadura.

Para finalizar este capítulo, solo queda destacar que, como se ha estado diciendo anteriormente, algunas de las ideas del diseño quedaron fuera del proyecto debido a la falta de tiempo para sus implementaciones, pero esto será explicado más detalladamente en el Capítulo 5.

Capítulo 5. Conclusiones y trabajo futuro

Del desarrollo de este proyecto se ha aprendido a usar una gran cantidad de herramientas propias de Unity que, de no haberse aprendido durante un trabajo como este, igual no se hubiera ahondado tanto en las funcionalidades que ofrece la herramienta de Unity2D.

Cabe destacar que antes de realizar este proyecto, no se había tenido el más mínimo contacto con Unity y el proceso de aprendizaje abarcó una parte bastante notable del tiempo para desarrollar el proyecto. La constante búsqueda de información por foros, vídeos de la página de Unity, consejos y ayudas del tutor y lectura de libros sobre Unity, ha conseguido crear una base de conocimientos fundamental para este proyecto, a costa de tener una fase de aprendizaje de la tecnología bastante laboriosa y prolongada.

Además, se ha aprendido a programa en C#, un lenguaje con el que tampoco se había tenido contacto previo a este proyecto. Aún siendo muy similar al lenguaje Java, se ha hecho entretenido el ver otros lenguajes nuevos para seguir investigando sobre ellos al finalizar este proyecto.

Otro detalle muy importante por destacar es la gran cantidad realizada de scripts, configuraciones de herramientas de Unity y comprobaciones de interacción entre todos los componentes ante una gran variedad de situaciones y contextos diferentes, con el fin de minimizar todo lo posible los errores del juego y conseguir así un producto mucho más trabajado y pulido.

Algo realmente curioso que ha ido pasando a medida que se iban implementando los nuevos componentes del juego era que siempre se iban ocurriendo ideas nuevas para hacer aún más únicos y particulares cada uno de los Prefabs. Por ejemplo, la idea de un interruptor con tres comportamientos diferentes surgió de improvisto nada más haber implementado la funcionalidad de abrir una puerta.

Esta capacidad de absorber y maravillar aún al propio programador del proyecto, es algo fabuloso de los videojuegos y unos de los motivos por los que son un arte aun siendo programas informáticos, porque el ver como cada uno de los componentes que se van creando va tomando vida y va interactuando con los que

ya se habían implementado en iteraciones anteriores dan una sed de conocimiento que solo hace que se quiera añadir más vida por así decirlo al pequeño mundo que se está creando frente a tus ojos debido además al esfuerzo propio.

Aún así, y debido a que no se tenía nada de conocimiento sobre Unity, surgieron problemas que estuvieron quebrando la cabeza durante bastantes días de desarrollo, además de temas como el del efecto de atacar para el que no se contaban con sprites necesarios, que enseñaron lo difícil que serían las implementaciones sin utilizar las herramientas adecuadas proporcionadas por la herramienta.

A estos problemas se le suma la falta de tiempo para dedicar al proyecto debido a que, durante todo el ciclo de vida del proyecto, se ha estado trabajando en una empresa a tiempo completo de programador, lo cual fatiga mucho las ganar de seguir programando al llegar a casa. Aún así se ha conseguido un resultado bastante completo y que está a pocos pasos de convertirse en el resultado final que se tenía pensado en la fase de diseño.

Dicho esto, se destaca que, en un futuro, el plan es terminar de implementar todos los enemigos que se tenían planeados, usar más efectos de sonido y música y hacer más animaciones para dar un poco mas de detalle al juego. También se quedaron atrás grandes partes del diseño de los diálogos y se espera poder desarrollar la trama por completo en una versión futura además de diseñar más niveles y plataformas especiales para hacer de esta versión un juego realmente difícil de completar.

Por último, decir que, aunque al principio la herramienta de Unity imponga bastante respeto debido a la grandísima cantidad de funcionalidades de las que dispone, el poder implementar comportamientos de forma encapsulada y usando los Prefabs, hace que investigando poco a poco en la amplia documentación de la que dispone y dividiendo el proyecto en las adecuadas subtareas, no sea imposible hacer la implementación de cualquier comportamiento que se nos ocurra.

Bibliografía

Albert Capdevila (2017). El patrón singleton en C#. (Accedido el 8 de septiembre de 2019:
https://albertcapdevila.net/patron-diseno-singleton-csharp)

AmericaEconomía (2016). Ingeniería en Informática es la carrera más demandada en Iberoamérica. (Accedido el 6 de septiembre de 2019:
https://mba.americaeconomia.com/articulos/notas/ingenieria-en-informatica-es-la-carrera-mas-demandada-en-iberoamerica)

Ana Isabel Ledo Rubio (2018). Potencial de los juegos serios. (Accedido el 5 de septiembre de 2019:
http://hyperbole.es/2015/11/videojuegos-buenos-o-malos-para-la-salud-mental/)

Antonio Moon (2019). Aprende C# con Unity - Corrutinas. (Accedido el 5 de septiembre de 2019:
https://moonantonio.github.io/post/2018/csharpunity/007/)

Asociación Española de Videojuegos (2019). Evolución del videojuego en el mundo. (Accedido el 4 de septiembre de 2019:
http://www.aevi.org.es/la-industria-del-videojuego/en-el-mundo/)

Dave Calabrese (2014) Unity 2D Game Development. Editorial: Packt Publishing Ltd

DevelopMania (2009). Serialización de datos en C#. (Accedido el 9 de septiembre de 2019:
https://developmania.wordpress.com/2009/07/02/serializacion-de-datos-en-c/)

Frida Díaz (2016). Potencial de los juegos serios. (Accedido el 5 de septiembre de 2019:

http://www.eduforics.com/es/los-juegos-serios-y-su-potencial-como-dispositivos-educativos/)

Informática I (2014). Causas y consecuencias de la adicción a los videojuegos. (Accedido el 7 de septiembre de 2019:
https://informatica-i332.webnode.es/news/causas-y-consecuencias-de-la-adiccion-a-los-videojuegos/)

Jeff Sutherland, J.J. Shutherland (2014). Scrum: The Art of Doing Twice the Work in Half the Time. Editorial: Crown Publishing Group.

Jorge Vallejo (2017). GDD (Game Design Document). (Accedido el 5 de septiembre de 2019:
http://www.jorgevallejo.es/2017/10/gdd-game-desing-document.html)

José Vicente Pons (2017). ¿Que son los juegos serios? (Accedido el 5 de septiembre de 2019:
http://www.exelweiss.com/blog/356/serious-games-juegos-serios/)

Leandro González (2016). How to Write a Game Design Document. (Accedido el 7 de septiembre de 2019:
https://www.gamasutra.com/blogs/LeandroGonzalez/20160726/277928/How_to_Write_a_Game_Design_Document.php)

Marble Station (2008). Metodologías ágiles de gestión de proyectos (Accedido el 5 de septiembre de 2019:
https://www.marblestation.com/?p=661)

Microsoft (2015). Introducción al lenguaje C# y .NET Framework (Accedido el 9 de septiembre de 2019:
https://docs.microsoft.com/es-es/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework)

Miguel Angel Álvarez (2001). ¿Qué es la programación orientada a objetos? Accedido el 6 de septiembre de 2019:
https://desarrolloweb.com/articulos/499.php)

OpenGameArt. Galería de recursos gratuitos para videojuegos. (Accedido el 6 de septiembre de 2019:
https://opengameart.org/)

Project Management Institute (2017). A guide to the Project Management Body of Knowledge (PMBOK guide). Editorial: Project Management Institute.

Raúl González (2018). Videojuegos Cultura. (Accedido el 5 de septiembre de 2019:
https://www.eldiario.es/cultura/videojuegos/Videojuegos-cultura-Jose_Guirao-ministerio_de_Cultura_y_Deporte-Gamescom_0_807019612.html

SEGAN (2019) Ejemplos de juegos serios. (Accedido el 6 de septiembre de 2019:
http://seriousgamesnet.eu/games)

Shutterstock (2017). Ingeniería es el área con mayor tasa de abandono escolar en España. (Accedido el 6 de septiembre de 2019:
https://noticias.universia.es/ciencia-tecnologia/noticia/2017/06/27/1153624/ingenieria-area-mayor-tasa-abandono-escolar-espana.html)

Unity (2019). Animator. (Accedido el 8 de septiembre de 2019:
https://docs.unity3d.com/ScriptReference/Animator.html)

Unity (2019). Canvas. (Accedido el 8 de septiembre de 2019:
https://docs.unity3d.com/es/current/Manual/UICanvas.html)

Unity (2019). Cinemachine. (Accedido el 8 de septiembre de 2019:
https://unity.com/es/unity/features/editor/art-and-design/cinemachine)

Unity (2019). Tile Palette. (Accedido el 8 de septiembre de 2019: https://docs.unity3d.com/Manual/Tilemap-Palette.html)

Unity (2019). Prefabs. (Accedido el 8 de septiembre de 2019: https://docs.unity3d.com/es/2019.1/Manual/Prefabs.html)

Unity (2019). The Inspector window. (Accedido el 11 de septiembre de 2019: https://docs.unity3d.com/Manual/UsingTheInspector.html)

Unity3D (2019). Unity User Manual. (Accedido el 5 de septiembre de 2019: https://docs.unity3d.com/Manual/)

Víctor Gómez (2015). Diagrama de Clases. (Accedido el 8 de septiembre de 2019: https://instintobinario.com/diagrama-de-clases/)

Wikilibros (2019). Programación Orientada a Objetos/Características de la POO. (Accedido el 8 de septiembre de 2019: https://es.wikibooks.org/wiki/Programaci%C3%B3n_Orientada_a_Objetos/Caracter%C3%ADsticas_de_la_POO)

Wikipedia (2019). Videojuego de plataformas. (Accedido el 5 de septiembre de 2019: https://es.wikipedia.org/wiki/Videojuego_de_plataformas)

Anexo I : versión final GDD

A continuación, se adjunta la última versión realizada del GDD que sentó todas las bases para el comienzo de la fase de implementación del proyecto. Esto, demuestra que el uso de una metodología ágil permite que se implanten cambios sobre los requisitos iniciales sin que los tiempos del proyecto ni los recursos de los que se disponen se vean comprometidos.

El GDD, al igual que el resto de la memoria de este proyecto se estructurará siguiendo un índice expuesto en la primera página de este para luego continuar inmediatamente con el desarrollo de todos los puntos siguiendo el orden correspondiente.

1. *Personajes*
2. *Historia*
    - *2.1. Tema*
3. *Progreso en la historia*
    - *3.1. Diagrama de flujo*
4. *Jugabilidad*
    - *4.1. Objetivo*
    - *4.2. Habilidades del jugador*
    - *4.3. Mecánicas de juego*
        - *4.3.1. Físicas*
        - *4.3.2. Movimiento del personaje*
        - *4.3.3. Acciones*
        - *4.3.4. Combate*
        - *4.3.5. Transición entre pantallas*
    - *4.4. Objetos y potenciadores*
    - *4.5. Progresión y desafío*
    - *4.6. Derrota*
5. *Estilo del diseño*
6. *Música y efectos de sonido*
7. *Descripción técnica*
8. *Mercado*
    - *8.1. Público objetivo*
    - *8.2. Monetización*
    - *8.3. Idioma*
9. *Otras ideas*

1. Personajes

- Nuestro personaje controlable es el protagonista del juego. Su nombre es **Carlos** y es un chico normal, estudioso y aficionado a los videojuegos y al anime, de ahí que su avatar en el juego de moda **BitPlanet** sea **Kirin** (proveniente de “Kirito”, un protagonista de un anime que le entusiasmaba especialmente).

 *Imagen de Kirin*

- **Yiem**, es dueño y Maestro de Juego de un planeta virtual (llamado **BitPlanet)** donde reina la paz y la armonía y sus habitantes hacen sus cosas de la misma forma día tras día.

Yiem posee gran conocimiento de la programación y del mundo digital, pero necesitará de alguien que pueda aprender sus conocimientos y ponerlos en práctica para acabar con nueva amenaza que asola su planeta, ya que el paso del tiempo machaca ya su anciano cuerpo.

 *Imagen de Yiem*

- Los **Akuma** son los malos de esta historia. Tienen una sed de destrucción increíble (de ahí su nombre ya que significa demonio en japonés) y viven en los videojuegos a modo de virus esperando que aparezca algún jugador para acabar con su partida.

Los **Akuma** surgieron a raíz de jugadores que llevaron su afición a los videojuegos por el mal camino y sus almas se consumieron convirtiéndoles en datos corruptos.

Hay muchos tipos de **Akuma,** desde pequeñitos, casi inofensivos y muy torpes hasta algunos de un tamaño superior a los de las personas y con una gran inteligencia y dotes del engaño. Aunque todos comparten el mismo afán por acabar con todo aquello que les rodea, se pueden diferenciar varios tipos:

- Los **Diablillos** son **Akuma** que simplemente vagan por los juegos sin rumbo fijo.

Representan a los jugadores que por su afición a los juegos simplemente dejaron de lado el mundo real y se aislaron de sus seres queridos y amigos. No suponen una amenaza demasiado grande, pero son relativamente veloces.

 *Imagen de un Diablillo*

- Los **Odei** son Akuma cuya forma es líquida y sólo ciertas armas y objetos permiten eliminarlos.

Representan a los jugadores que se dejaron llevar por el odio en los juegos y cometieron actos violentos imperdonables a sus familiares. No es posible eliminarlos de un pisotón lo que los hace una amenaza mucho mayor que los diablillos, aunque son más lentos.

 *Imagen de un Odei*

- Los **Hakka** son **Akuma** que pueden traer problemas si no se eliminan con rapidez, puesto que, debido a su inteligencia, han dominado la técnica de la generación espontánea y son capaces de invocar (mientras vivan) esbirros esqueléticos que atacan a los jugadores.

Representan a los jugadores que dedicaron su vida a robar cuentas de los juegos de otros jugadores para luego venderlas o aprovecharse de los objetos que estos habían obtenido. Son débiles y fáciles de matar, pero huyen lentamente de los peligros mientras invocan a sus sirvientes desde un lugar seguro.

Imagen de un Hakka

- Los **sirvientes de Hakka** son simples esqueletos sin inteligencia ni sentimientos que avanzan rápidamente en una sola dirección atacando a todo lo que se encuentren de paso. Son bastante débiles y técnicamente no son **Akuma** ya que no poseen el alma de ningún humano, pero aun así, en grandes cantidades suponen un serio problema.

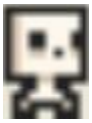


Imagen de un sirviente de Hakka

- Los **señores oscuros** son los dos líderes de los **Akuma** y jamás han sido derrotados. Son conocidos, aparte de por su poder y astucia, por cometer errores inimaginables mientras vivían en el mundo de los vivos. Coordinan las fuerzas de los **Akuma** para acabar con todos los planetas virtuales de los juegos y que así nadie más disfrute de los videojuegos.

  - El señor oscuro **Kyuzo,** conocido como el **Devorador de Esperanzas**, era un empresario de éxito y dueño de una multinacional comerciante de patatas.
    Un día su afición por los micro pagos en los juegos gratuitos llegó hasta tal punto que causó la bancarrota de su compañía, haciendo que miles de empleados perdieran su trabajo y sus familias pasaran muchísima hambre.

    Posee muchísima resistencia y unos ataques cuerpo a cuerpo muy potentes.

Su ***ataque especial*** es un proyectil que paraliza a cualquier enemigo que lo toque sin causar ningún tipo de daño.

Nadie ha conseguido derrotarlo, pero hay libros en los que se dice que su punto débil su pequeña cabecita.

Imagen de Kyuzo, Devorador de Esperanzas

- El señor oscuro **Taidana,** conocido como el **Aplastador de Mundos**, era un responsable de seguridad de una central nuclear de un país que ya no existe.
  Era aficionado a jugar al ordenador en horario de trabajo y un día, hubo una alerta de fusión de núcleo de la central, pero como él (único capaz de solucionar el problema antes de que pasara algo grave) estaba con los cascos puestos jugando, no escuchó la alerta, la central explotó y todo el país fue arrasado.

  Es capaz de correr a gran velocidad dañando a todo aquello que se interponga en su camino, además cuenta con un ***ataque especial*** que crea una onda sísmica muy potente.

  Es inmune a la magia debido a que su alma se llenó de radiación nuclear antes de morir y además no se le puede pisar encima de la cabeza puesto que tiene una hoja que le protege de todo el daño, por lo que la única opción podría ser armas cuerpo a cuerpo.

Imagen de Taidana, Aplastador de Mundos

- El Rey Demonio **Okui**, líder supremo de los **Akuma** y el señor oscuro más poderoso de todos, era antiguamente un universitario normal de primer año que se dejó llevar por los videojuegos.
  Su principal y única afición era jugar a juegos de violencia y disparos con mucha frecuencia. Llegó un momento en el que dejó de ir a clase para dedicarse al cien por cien a jugar y llegaba hasta el punto de jugar unas 20 horas en un solo día, parando solo para ir al baño, dormir y comer comida basura. Por ello el niño enfermó y murió de un infarto debido a sus malos hábitos de vida.

  Su resistencia es mucho más elevada que la de cualquiera de los señores oscuros y es capaz de hacerse inmune al daño durante breves periodos de tiempo. Su forma de atacar es mediante torpes ataques cuerpo a cuerpo, pero a medida que su salud decrementa, sus ataques se vuelven más rápidos y aumenta su velocidad de movimiento.

  Tiene tanta vida que puede recibir daño de cualquier tipo, pero cuidado cuando se enfurezca puesto que cientos de jugadores han sucumbido ante su forma final.

Imagen de Okui, Rey Demonio

2. Historia

Todo estaba tranquilo en el planeta virtual de Bitplanet. Los aldeanos iban de un lado para otro, los jugadores usaban los objetos que conseguían en las misiones y los cambiaban por otros objetos en los comercios, nada se salía de lo común.

Un personaje con apodo de **Kirin** acaba de llegar a la plaza principal después de haber completado una mazmorra con su gremio y está recordando con sus otros dos compañeros de equipo los momentos de tensión de la misión entre carcajadas y burlas entre ellos.

Pero algo raro pasó de repente, el cielo se empezó a oscurecer y todos los inventarios y menús de habilidades de los jugadores dejaron de funcionar. De pronto, de una nube negra del cielo caen unas especies de meteoritos color azabache por todo el planeta de BitPlanet.

Nadie sabía que estaba pasando hasta que varios pares de meteoritos cayeron abruptamente en la plaza principal. Ante la sorpresa de todos, los meteoritos no eran trozos de piedra, eran unas criaturas que se comportaban de forma extraña.

Un grupo de exploradores y guerreros atacaron a las criaturas tomándolas como un evento especial del juego que darían experiencia adicional si se eliminaban, pero, ante la sorpresa de todos, las armas se destruían al contactar con la criatura que acababa de aparecer y esta, de un solo ataque mato a uno de los jugadores.

Pero algo raro pasaba, al morir, el cadáver permanecía en el suelo en vez de desaparecer, el jugador había muerto en la vida real por causa de ese monstruo. Por mucho que los jugadores trataban de dejar de jugar, era inútil porque si algunos de esos monstruos derrotaban a su personaje, aunque este no estuviera siendo controlado en ese momento, el dueño del avatar moría. Decenas de jugadores murieron en ese momento, entre ellos los compañeros de **Kirin**.

Ante esto, apareció **Yiem**, el maestro del juego y dueño de BitPlanet, el cual alejó a todos los monstruos de la plaza y se llevó a **Kirin,** el cual seguía conmocionado por la muerte de sus amigos, con una teletransportación a su palacio lejos de allí.

Una vez llegaron al palacio de **Yiem**, este le explicó a **Kirin** lo que pasaba:

Los **Akuma**, una raza de demonios virtuales que vivían para asolar los mundos a los que iban, se había colado en el juego y estaba masacrando a los jugadores.

Las armas introducidas hasta ahora en el juego no valdrían contra estas criaturas y le pidió a **Kirin** que salvara el juego, ya que lo había estado observando y su inteligencia sería la única capaz de crear armas con los códigos del juego y poder así acabar con esas criaturas, ya que **Yiem** estaba muy anciano y solo podría ayudar a **Kirin** desde la distancia.

Dicho esto, **Kirin** accedió aun temblando a llevar a cabo la misión, por lo que **Yiem** lo teletransportó de nuevo la plaza principal para que comenzara así su lucha contra los demonios que mataron a sus compañeros de aventuras.

2.1. Tema

Este es un juego que enseñará al jugador cosas básicas sobre la programación orientada a objetos haciendo que añada así conocimiento en la vida real y habilidades en el juego para que **Kirin** pueda completar la limpieza de **Akuma** de **BitPlanet.**

Se mantendrá en el juego cierto matiz de superación personal del protagonista además de enseñar al jugador los peligros de llevar al extremo el uso de los videojuegos (simbolizado por la maldad y el origen de los **Akuma**).

3. Progreso en la historia

El juego comienza con el texto del punto 2 intercalado con imágenes de los personajes que la llevan a cabo (los primeros **Akuma**, **Kirin**, los dos amigos de **Kirin** y **Yiem**).

Tras esto, empieza el primer capítulo en la **Plaza Central** de **BitPlanet**. Este capítulo sirve un poco a modo de tutorial para que el jugador se asocie con los controles y la jugabilidad es presentada por **Yiem**, el cual hace de ayuda omnipresente durante todo el juego y le revela que necesitará 3 fragmentos de código corrupto para cerrar el portal de los demonios.

Una vez el jugador supera este primer nivel, podrá elegir cuál será el siguiente mundo:

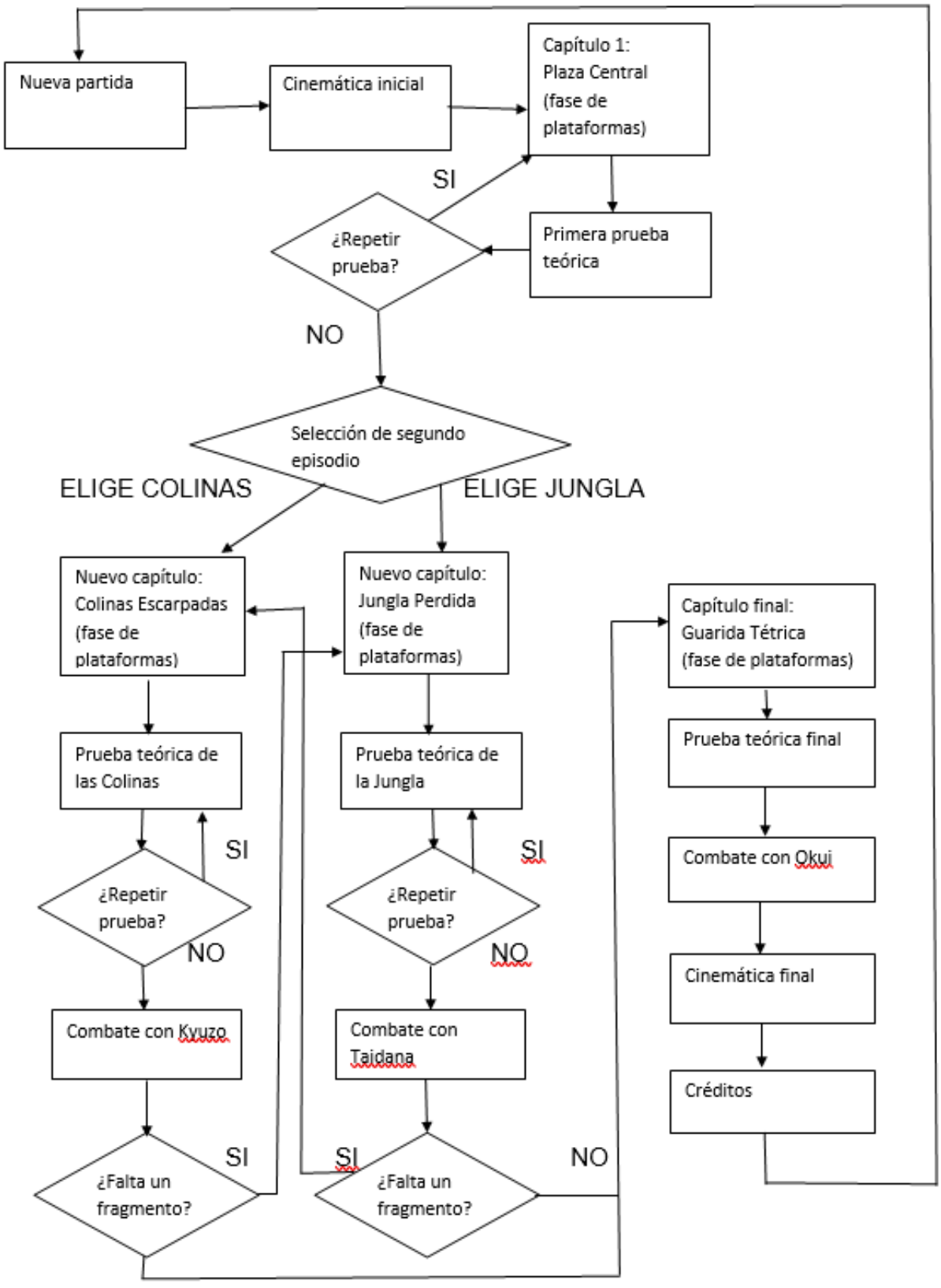
- **La Jungla Perdida**
- **Las Colinas Escarpadas**

Cuando se derrote al jefe de los **Akuma** asignado a la zona que haya elegido, recibirá uno de los fragmentos corruptos e irá al mundo que no eligió para conseguir otro de los fragmentos.

Tras este tercer capítulo, el jugador será transportado a la **Guarida Tétrica** donde está el tercer y más poderoso miembro de los **Señores Oscuros Akuma** y tras derrotarle, conseguirá el último fragmento que le permitirá cerrar el portal y acabar con la amenaza de **BitPlanet** y con la aventura de nuestro héroe.

3.1 Diagrama de flujo

El diagrama de flujo de la historia principal sería el siguiente:



4. Jugabilidad

En esta sección se explicará de forma detallada la forma que tendrá de jugarse el juego, pasando por habilidades, objetivos, mecánicas principales y potenciadores entre otras cosas.

4.1. Objetivos

Se diferencian dos tipos de objetivos en este juego:

- El *objetivo a corto plazo* es aprender cosas básicas de la programación a objetos mientras también se avanza a través de los niveles derrotando enemigos y haciendo entre plataformas que, más de una vez, precisaran de bastante habilidad por parte del jugador.

- El *objetivo a largo plazo* es conseguir las tres partes del código corrupto para así cerrar el portal que trae a los **Akuma** a **BitPlanet** y así salvar a los jugadores que están en él.

4.2. Habilidades del jugador

Para que el jugador sea capaz de jugar al juego sin problemas, necesitará tener nociones básicas de las siguientes habilidades:

1. ***Memoria*** para resolver los puzles de programación que le vayan surgiendo a lo largo de la historia.
2. ***Coordinación*** para poder superar las fases de plataformas más difíciles que se le planteen.
3. ***Intuición*** para poder encontrar el punto débil de cada enemigo y poder así derrotarlo.
4. ***Reflejos*** para poder esquivar ciertos ataques muy rápidos o muy potentes que penalizarán bastante al jugador si lo tocan.
5. ***Astucia*** para identificar en una pantalla llena de enemigos cual debe ser su objetivo prioritario por derrotar.

4.3. Mecánicas de juego

En este subapartado, se plantearán las reglas físicas del mundo, la forma con al que el jugador interactúa con el personaje y las acciones que este podrá realizar con su entorno.

4.3.1. Físicas

Al tratarse de un juego de plataformas 2D, las físicas serán principalmente una gravedad y un rozamiento suficientes como para que el jugador pueda realizar saltos entre plataforma con comodidad y de una forma lógica, nada de gravedad reducida o rozamiento que te haga pegarte al suelo.

4.3.2. Movimiento del personaje

El jugador moverá al personaje usando las flechas de dirección del teclado, saltará con la barra espaciadora y dispondrá de unas habilidades que serán explicadas en la ***Sección 4.3.3 Acciones.***

4.3.3. Acciones

El jugador dispondrá, aparte del salto y el movimiento básico, de hasta tres habilidades desbloqueables a lo largo de la aventura y estas son:

- **Golpe de espada:** Primera habilidad desbloqueable que permitirá al jugador hacer un golpe lateral con una espada mágica.

  Este golpe hace más daño que un salto simple y para usarlo se deberá presionar la ***tecla A.***

- **Lanzamiento de magia:** El jugador lanzará una esfera mágica en una dirección que recorrerá una distancia moderada hasta que golpee a un enemigo o llegue a cierta distancia.

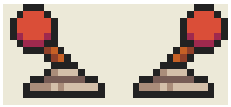
  Este ataque hace el mismo daño que un salto simple y para usarlo se deberá presionar la ***tecla S.***

- **<u>Golpe giratorio:</u>** Esta habilidad permitirá que el jugador haga un barrido lateral rápido usando una ligera pero letal hacha.

  Este ataque hace el mismo daño que un salto simple pero afecta a los enemigos cercanos por ambos lados del personaje. Para usarlo se deberá presionar la **tecla D**.

Además, el jugador podrá interactuar con cofres que soltarán objetos si se les ataca con cualquiera de las 3 habilidades mencionadas e interruptores que permitirán acceder a ciertas zonas si se les golpea con la espada.

 *Imagen de cofre cerrado*

 Imagen de interruptor desactivado/activado

También se podrán interactuar con puntos de guardado pasando sobre ellos para poder reaparecer en ciertos lugares en caso de morir en momentos posteriores.

 *Imagen de punto de control activado/desactivado*

4.3.4. Combate

Los combates contra los **Akuma** más débiles se realizarán de forma intuitiva y se deberá a prender los patrones de comportamientos que siguen estos enemigos y atacarles con las habilidades más efectiva según el tipo de **Akuma** al que nos enfrentemos.

Los combates contra los **señores oscuros,** sin embargo, serán enfrentamientos más duros, ya que tienen mucha más vida que los enemigos corrientes y poseen características físicas adicionales como saltos muy altos o velocidades de vértigo.

Además, cada uno cuenta con habilidades especiales que el jugador deberá evitar para que no se le penalice en exceso (ya que suelen o afectar al estado del jugador o causarle gran cantidad de daño). Aun así, cada **Señor Oscuro** cuenta con puntos débiles que el jugador deberá identificar si quiere ganar el combate.

4.3.5. Transición entre pantallas

Cada uno de los capítulos antes mencionado, cuentan con unos fondos y unos elementos de campo diferentes entre sí, aunque las propiedades del mapa serán las mismas en cada episodio.

Lo único que cambiará de episodio a episodio serán el tipo de enemigos, la cantidad de estos y la dificultad de los saltos a realizar.

La transición entre estos episodios se realizará a través de un portal que **Yiem** nos abrirá al final de cada capítulo, con la posibilidad de elegir cual queremos que sea el capítulo 2 y cual el 3.

La estética de las fases se mostrará en la ***Sección 5 Estilo de diseño.***

4.4. Objetos y potenciadores

En esta sección se explicarán los objetos y potenciadores que podrán ser encontrados a lo largo del juego.

En primer lugar, la mayoría de los objetos podrán ser encontrados en cofres que, como hemos dicho en la ***Sección 4.4.3 Acciones***, podrán ser abiertos usando cualquiera de las habilidades. La otra forma de obtenerlos será de algunos enemigos al morir.

Dicho esto, los objetos de este juego serán:

- **Poción de vida débil:** Restaura hasta 1 punto de vida al jugador.

Imagen de poción de vida débil

- **Poción de vida moderada:** Restaura hasta 2 puntos de vida al jugador.

 *Imagen de poción de vida moderada*

- **Poción fénix:** Restaura por completo la vida del jugador.

 *Imagen de poción fénix*

- **Poción de furia:** Aumenta el daño que aplica el jugador a los enemigos durante un breve periodo de tiempo y lo hace más resistente al daño.

 *Imagen de poción de furia*

- **Fragmento corrupto de código:** Son obtenidos como botín al derrotar a un señor oscuro y cuenta la leyenda que solo alguien que obtenga los tres fragmentos podrá acabar con la amenaza de los **Akuma** para siempre.

 *Imágenes de los fragmentos*

- **Espada corta de Atila:** Se obtiene al pasar la primera de las pruebas de codificación del juego, situada en la **Plaza Central**. Desbloquea la habilidad del *Golpe de Espada.*

 *Imagen de la Espada corta de Atila*

- **Bastón esmeralda de Merlín:** Se obtiene al pasar la prueba de codificación de la **Jungla Perdida**. Desbloquea la habilidad del *Lanzamiento de magia.*

Imagen del Bastón esmeralda de Merlín

- **Hacha de mano de Ullr:** Se obtiene al pasar la prueba de codificación de las **Colinas Escarpadas**. Desbloquea la habilidad del *Golpe Giratorio.*

Imagen del Hacha de mano de Ullr

4.5. Progresión y desafíos

En esta sección se explicará cómo aumentará la dificultad a lo largo del juego.

La primera fase (la que da lugar en la **Plaza Central**), tendrá una dificultad relativamente baja ya que sirve como tutorial del juego y está destinada a que el jugador se acostumbre a los controles. A lo largo de esta fase, el jugador será puesto a prueba por **Yiem**, el cual intercalará momentos de peleas con enemigos débiles con preguntas sobre programación orientada a objetos, para comprobar así si el jugador será capaz de completar su misión.

La segunda y tercera fase serán elegidas en el orden que quiera el juagador, ya que al final del primer episodio **Yiem** le preguntará al jugador que prefiere hacer, si escalar altas montañas para ver que tal esta la situación en **BitPlanet** o limpiar lugares infestados de **Akuma**. Ambas zonas contarán con nuevos enemigos no vistos hasta ahora por el jugador.

Con esta pregunta, se estará refiriendo a si el jugador quiere ir a una zona donde el plataformeo es más difícil de lo habitual (**Colinas Escarpadas**) o si prefiere ir a matar enemigos (**Jungla Perdida**).

En la zona de la guarida tétrica el jugador se encontrará con gran variedad de enemigos y con plataformeo difícil, e incluso con nuevos enemigos nunca vistos hasta ahora.

Cada una de estas zonas contará con otras teóricas que permitirán al jugador avanzar más fácilmente en la aventura.

Explicare más detalladamente el beneficio de las pruebas teóricas, que serán un total de 4 y constarán de unas 5 preguntas cada una sobre programación orientada a objetos básica:

- Si el jugador responde a las preguntas planteadas sin equivocarse en más de una pregunta por prueba, obtendrá 2 corazones de vida máximos adicionales.

- Si el jugador se equivoca 2 o 3 veces, recibirá solamente un corazón de vida máximo adicional.

- Si el jugador se equivoca más de 3 veces, no recibirá ningún corazón como recompensa y será regañado por **Yiem.**

Al final de todas las pruebas teóricas (excepto en la última ya que sirve a modo de prueba final), se puede repetir el desafío para así obtener las recompensas deseadas por el jugador y consiga aprender mejor los conceptos planteados.

Evidentemente el tener más corazones de vida, hará que el juego sea más sencillo puesto que el jugador tendrá un mayor margen de error en los capítulos, pero es posible superar el juego con los corazones máximos iniciales y mucha habilidad.

4.6. Derrota

La forma de perder en el juego es quedarte sin vidas en algún momento de la partida (inicialmente contaremos con 3 y se podran incrementar a lo largo de la historia).

Cuando esto pase, aparecerá **Yiem** y nos preguntará si queremos intentarlo de nuevo:

- Si decimos que sí, **Yiem** nos hará aparecer con su magia en el último punto de control que hayamos activado.

- En caso de decir que no y nos demos por vencidos, **Yiem** pondrá un mensaje indicando que está dececionado y se cerrará el juego, haciendo que el jugador tenga que empezar de cero la partida la próxima vez que abra el juego.

5. Estilo del diseño

La única parte del juego que aún no ha sido explicada cómo será gráficamente es la de las distintas fases, por ello he reservado esta sección del documento para mostrar ejemplos de tiles de cada fase y los fondos de paisaje que tendrán:

- **Plaza Central**

  Con ambientación de una ciudad arrasada por el ataque de los **Akuma**, pueden apreciarse edificios abandonados en el fondo.

  - *Background*

  - *Tiles*

- **Jungla Perdida**

  Ambientada en uno de los lugares más alejados para los jugadores de **BitPlanet**, se aprecian bastas vegetaciones y ningún rastro de edificios o modificaciones del entorno por parte del hombre.

Este aislamiento del resto de jugadores explica por qué tienen aquí establecida una de sus bases los **Akuma,** destinada al fortalecimiento de sus tropas. Al fondo del background, se pueden apreciar las **Colinas Escarpadas**.

- *Background*

- *Tiles*

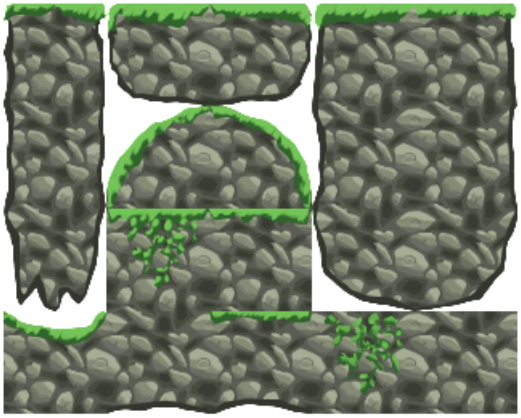
- **Colinas Escarpadas**

Este lugar se encuentra en el punto más elevado de **BitPlanet**, por lo que llegar aquí de una pieza es en sí un logro para los jugadores.

Desde aquí se puede observar cómo avanza la invasión de los **Akuma** ya que se ve todo el planeta perfectamente y por ello es utilizada como centro de inteligencia.

- *Background*

- *Tiles*



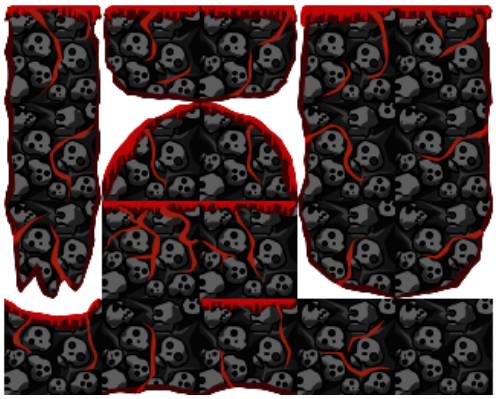
- **Guarida Tétrica**

  Fortaleza creada por los **Akuma** al llegar a **BitPlanet** levantada sobre una prospera ciudad en la que ahora los cadáveres de sus antiguos habitantes se apilan haciendo de suelo para los demonios que custodian este lugar.

  Se respira un ambiente tenebroso que disuadiría a cualquier insensato a adentrarse en este lugar y, por si fuera poco, se oyen ruidos muy peliagudos de su interior.

- *Background*

- *Tiles*



6. Música y efectos de sonido

En cada una de las fases, la música de fondo debe ser tensa ya que la amenaza de los Akuma hace sentir al protagonista de esa forma y se busca que el jugador comparta esa tensión.

En los combates contra los jefes, la música será electronica épica y enérgica para que el combate sea más espectacular.

Sin embargo, en las pruebas teóricas, habrá una música más calmada que ayudarán a pensar al jugador, acompañada de un sonido característica cuando acierte o falle la respuesta.

Los enemigos al atacar, moverse o ser derrotados también emitirán sonidos. El mismo protagonista al correr o hacerse daño también hará ruidos.

Una parte a destacar es que los jefes emitirán un sonido característico cuando vayan a hacer su ataque especial, ayudando al jugador a identificarlo con antelación y ponerse a salvo.

7. Descripción técnica

Inicialmente, el juego estará disponible únicamente para la plataforma de **Windows** y se usará para su desarrollo la plataforma de desarrollo **Unity 2D**.

Esto será descrito más a fondo en un **TDD** (Documento de Diseño Técnico).

8. Mercado

En esta sección, se aclaran puntos de cuál es la audiencia objetivo del juego, la lengua usada en los textos y cuál será el precio del mismo.

8.1. Público objetivo

El público al que se desea llegar con este juego es a cualquier persona que quiera introducirse en el mundo de la programación de una forma divertida y original, sin importer su edad o sexo.

8.2. Monetización

Inicialmente, el juego será gratuito y disponible para que quien quiera pueda descargarlo desde la página web principal del juego.

8.3. Idioma

El juego estará completamente en español pero, al estar orientado al mundo de la programación, algunos términos estarán en inglés.

9. Otras ideas

Se baraja también como idea que se puede añadir al juego la posibilidad de que el jugador tenga un **inventario** en el que podrá guardar pociones para usarlas cuando más las necesite.