UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA EN INFORMÁTICA

Kaladria: Bound By Fate. Diseño, desarrollo y comercialización de un videojuego multiplataforma

Mario Torres Martínez

Julio, 2019

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas De Información

GRADO EN INGENIERÍA EN INFORMÁTICA

Tecnologías de la Información

Kaladria: Bound By Fate. Diseño, desarrollo y comercialización de un videojuego multiplataforma

Autor: Mario Torres Martínez

Tutor académico: David Vallejo Fernández

Julio, 2019

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE **VOCAL** **SECRETARIO**

Resumen

En este documento se describen todas las fases necesarias para la creación de un proyecto de software multimedia enfocado al ocio digital; concretamente, en el formato de un juego interactivo o videojuego. El concepto es el de un juego de rol táctico con control de unidades y combate por turnos en el que el jugador debe avanzar a lo largo de múltiples niveles superando los desafíos que estos le presentan.

A la hora de afrontar el proyecto, se ha estimado conveniente dividirlo en tres fases: Diseño, fase en la cual se definirán los objetivos, diseños de personajes, diagramas, usabilidad de interfaces, escalabilidad del proyecto y cualquier otro paso secundario antes de avanzar a la fase siguiente de Desarrollo; en esta, se implementarán los distintos elementos arriba mencionados siguiendo una metodología de trabajo específica. Esto abarca el diseño de las interfaces, la programación del comportamiento de las unidades, el diseño de mapas, el almacenamiento de los datos, así como la revisión y optimización de todo lo implementado, todo ello con el fin de asegurarse de un rendimiento óptimo en el funcionamiento del juego. En la última fase, el juego será distribuido en una plataforma de venta, para lo cual se realiza un análisis de viabilidad de mercado con objeto de dilucidar la mejor opción de entre las que se adecúan al proyecto.

El documento concluye con la exposición de un análisis crítico del resultado obtenido, detallando de forma precisa qué objetivos han sido alcanzados y cómo estos se ven reflejados en el producto final.

Abstract

This document describes every needed phase in order to create a multimedia software project, focusing in digital leisure; specifically, in the format of an interactive game or videogame. The game concept is a tactical role-playing game with unit control and turn-based combat in which the player must progress along multiple levels, overcaming the challenges it presents.

Facing the project, it has been reckoned to divide it into three phases: Design, a phase in which objectives, diagrams, usability interface, project scalability and any other secondary step will be defined, prior to moving to the next Desing phase; in it, the varios aforementioned elements will be implemented, following a specific work methodology. This encompasses interface design, unit behaviour programming, map desing, data storage, as well as revision and optimization of everything implemented, all in order to to make sure that optimal game performance is achieved. In the last phase, the game will be distributed in a sales platform, for which a market viability analysis is done in order to elucidate the best option among the ones that fit the project characteristics.

The document concludes with a crytical analysis of the obtained result, precisely detailing what objectives have been achieved and how these are reflected in the final product.

Agradecimientos

A mis padres, por su apoyo cuando todo parecía negativo en mi vida.

A mi hermano, cuya capacidad para distraerme y hacerme reír es inigualable.

A Toñi, cuya comprensión y paciencia va más allá de cualquier límite.

A Irene, Jesús y Pablo, amigos desde hace tanto que ya son familia.

A cierto grupo de aventureros, cuya positividad y creatividad me asombra siempre.

A Guillermo, sin quien esto no hubiese sido posible.

A David, mi tutor, por estar siempre ahí cuando lo he necesitado, gracias de corazón.

Mario

Índice general

Índice de figuras

Capítulo 1

Introducción

A lo largo de este documento se describe la creación de un proyecto software multimedia desde cero. Previamente se realiza un análisis del marco social en el que se inscriben los videojuegos valorando su importancia en este. Esto dota al documento de un contexto y perspectiva que amplían la visión del desarrollo del proyecto.

1.1 Contexto socioeconómico de los videojuegos

1.1.2 Estado de los videojuegos en la sociedad moderna

Con la incorporación de nuevas tecnologías en la sociedad han aparecido nuevas formas de ocio en la última década, con una industria que ha empezado a superar hasta a la del cine, la de los videojuegos. El ocio digital supone un porcentaje bastante alto del tiempo que dedicamos al ocio a lo largo de la semana, con una media de hasta cinco horas semanales dedicadas exclusivamente a juegos. La industria de los videojuegos alcanza nuevas cotas año tras año, empezando en la actualidad a superar incluso a la del cine y ha cambiado tanto la forma en la que entendemos el ocio digital como en la que interactuamos con otras personas.

El uso masivo de los *smartphone* y tecnologías móviles ha abierto un nuevo mercado lleno de posibilidades para el ocio digital. Esto ha venido de la mano de la existencia de videojuegos que son capaces de interactuar con el entorno mediante el uso de realidad aumentada, de la creciente expansión de la realidad virtual, de juegos online que utilizan la gran infraestructura que provee internet y de que las formas de acceso a videojuegos sean más amplias que nunca. El impacto social que genera poder jugar con personas a miles de kilómetros de distancia, socializar con ellas a través de juegos llegando incluso la creación de comunidades, aumenta aún más dicho impacto por parte de los videojuegos en nuestro a día a día. Actualmente se pueden conseguir

puestos de trabajo relacionados simplemente con el hecho de jugar a videojuegos, y el mercado laboral para la producción de videojuegos es, sin duda, cada vez más deseado. Los videojuegos requieren grupos de profesionales numerosos y procedentes de extracciones de muy diferente naturaleza: artistas, músicos, ingenieros de software, hardware, etc. Esto hace que la creación de un videojuego sea un proyecto software que requiere de una capacidad multidisciplinar, lo cual lo hace extremadamente interesante desde un punto de vista profesional.

1.1.2 Videojuegos en España

Ahondando más en la cuestión de la relación existente entre los videojuegos y la sociedad, específicamente en la sociedad española, comprobamos que los videojuegos generan una gran fuente de beneficios para el Producto Interior Bruto del país, tanto por su consumo como por su creación y producción. Solo en 2017, el sector de videojuegos españoles facturó 713 millones de euros, un 15,6% más que en 2016, generando un incremento total de más del 16,5% de profesionales dedicados a este sector, cuyo número sobrepasa ya el de los 6.000. A pesar de esto, la gran mayoría de los videojuegos producidos en España son realizados por pequeños estudios, conocidos comúnmente como estudios *indies.* La mayoría de estos estudios en España están gestionados por PYMEs que, salvo en contados casos, cuentan con plantillas que

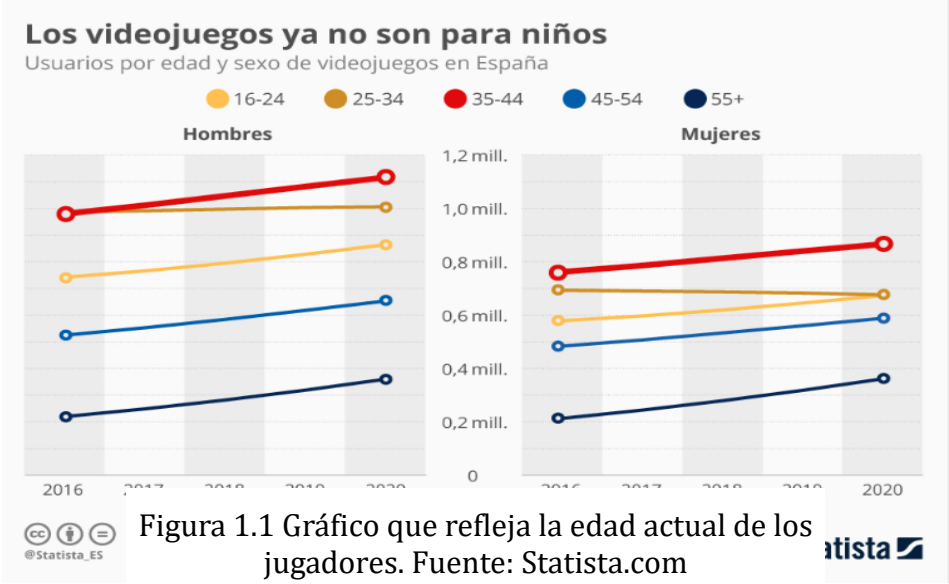


Figura 1.1 Gráfico que refleja la edad actual de los jugadores. Fuente: Statista.com

no superan los cincuenta miembros y que ni siquiera alcanzan los cinco en la mitad de las mismas [LBV18].

En España, la mayor parte de los consumidores de videojuegos tiene un perfil joven, fundamentalmente en la franja de edad que va de los 12 a los 35 años, y siendo como ya se ha mencionado un mercado creciente, el potencial grupo de jugadores de videojuegos va en aumento año tras año.

1.1.3 Género del videojuego

Existen infinidad de géneros dentro del marco de los videojuegos, desde *shooters* hasta puzles, pasando por juegos de plataforma, aventuras, educativos, etc. En este proyecto se ha decidido la realización de un juego del género *role-playing game* (juego de rol, en español). A continuación se procede a explicar brevemente el género y el subgénero al que pertenece este proyecto.

En los juegos de *role-playing game* («RPG» de aquí en adelante) los jugadores asumen el control de uno o más personajes a lo largo de una trama que permite al jugador progresar en el juego. Este tipo de género normalmente tiene una carga narrativa más elevada y mecánicas que suelen ser más complejas y que habitualmente no requieren tanta reacción en tiempo real como en otros videojuegos.

Figura 1.2: Ejemplo de un videojuego RPG. Fuente: Bethesda.com

Este género nació inspirado en los juegos de rol de mesa de los años 70, como Dungeons & Dragons, pero no fue hasta los años 90 cuando tuvo su verdadero boom, con múltiples juegos cuyo éxito se ha visto prolongado a lo largo de las últimas tres décadas, como son los casos de Dragon Quest, Final Fantasy, Might & Magic o Skyrim, y en la actualidad es un género que cuenta con un gran número de fans.

1.1.4 Plataformas móviles

Los últimos 10 años han cambiado el paradigma de los videojuegos y cómo jugarlos. Si hasta ahora los videojuegos eran un producto para jugar en los hogares o en salas recreativas, la llegada de las tecnologías móviles como los *smartphones* y las *tablets* permiten al usuario hacerlo en cualquier lugar. Un caso paradigmático lo constituye el fenómeno Pokémon GO, el cual alcanzó más de 100 millones de usuarios en su primer mes de publicación especialmente debido a su componente social, ya que al ser un juego de plataformas móviles permitía a los usuarios compartir, hablar y jugar con él en cualquier evento[Per16].

Figura 1.3: Jugadores de Pokémon GO en un evento

Con un mercado tan inmenso de potenciales usuarios, sería un error no enfocar la materialización del producto adaptando su formato y usabilidad a estas plataformas. Siendo los sistemas operativos más populares Android y iOS – aunque distanciado este en segundo lugar –, orientar la creación de un videojuego hacia estos sistemas operativos no solo es una buena decisión comercial, sino que es casi un requisito en el mercado actual.

1.2 Motivaciones

Tras todo lo expuesto anteriormente, queda evidenciada la existencia de un gran mercado con multitud de posibilidades en el campo de los videojuegos. Con todas las plataformas de desarrollo para estos, así como para las tecnologías móviles, cada día más presentes como el medio más prometedor para el que desarrollar aplicaciones.

Las tecnologías actuales de desarrollo en videojuegos son herramientas que se encuentran entre las más modernas y con los paradigmas de diseño y desarrollo más actuales, lo cuál hace que un proyecto de estas características sea una forma muy eficaz de adquirir conocimiento real en un campo que puede trasladarse a otros sin problema.

El desarrollo del proyecto requiere conocimiento multidisciplinar dentro de la industria del desarrollo multimedia: Decisiones de usabilidad, escalabilidad y ergonomía; optimización de código y algoritmia eficiente; desarrollo para hardware heterogéneo; capacidad de análisis de prueba y corrección de errores, entre otras.

Con estas premisas, el desarrollo de un videojuego se convierte en un proyecto deseable con múltiples opciones para su desarrollo que ofrece un conocimiento actual y sometido a altos estándares de calidad.

1.3 El proyecto

Teniendo en cuenta las motivaciones arriba mencionadas, se propone crear un videojuego donde el jugador tenga el control de múltiples unidades que están relacionadas entre sí por la trama, y que deben avanzar a lo largo de una serie de mapas en los que las unidades enemigas intentan derrotar a las unidades controladas por el jugador. El diseño de los mapas debe suponer un cierto desafío, generando situaciones

interesantes en las que el jugador deba pensar y plantearse sus opciones; además de esto, este debe poder personalizar en cierta medida las características de los personajes que controla, teniendo que tomar decisiones sobre qué habilidades poseerán los mismos. A esto hay que añadir que el juego debe dar facilidades para posibles modificaciones posteriores si se estimase oportuno.

A nivel técnico, esto supone la creación y definición de los niveles desde un punto de vista de la algoritmia. El proyecto debe resolver problemas como el desplazamiento de las unidades de forma automática por parte de la IA, detectando esta los obstáculos a sortear para alcanzar una posición correcta en el tablero. La eficiencia y consumo de animaciones, la asincronía entre la velocidad del código y su representación visual, la gestión de múltiples eventos en un momento dado en pantalla, así como en general el diseño de un código eficiente e interfaces responsivas, un sistema de ficheros que permita cambios y expansión en el código, así como el guardado de datos son otras de las características que deben ser tenidas en cuenta como partes importantes en la composición del proyecto.

1.4 Acerca de este documento

La estructura de los siguientes capítulos procede a describir en orden los puntos más importantes de cara al desarrollo del proyecto. Para esto, en el capítulo 2 se realiza un análisis de los objetivos, separándolos en el objetivo principal y los secundarios. El capítulo tres ahonda en el contexto de la situación del proyecto analizando el estado del arte respecto a los motores gráficos en su actualidad, y cuáles han sido las motivaciones que han llevado a elegir Unity. La metodología de trabajo es analizada en el capítulo 4, examinando su funcionamiento y viendo la aplicación de este al proyecto. Los capítulos 5, 6 y 7 describen respectivamente el diseño y desarrollo del juego, el ejecutable terminado y la posterior comercialización de este. Esto abarca diagramas, algoritmia, decisiones de diseño y elecciones de plataforma de venta, abarcando estos puntos la gran mayoría del documento. En último lugar se evalúa de forma crítica y concisa cómo han sido alcanzados los distintos objetivos propuestos y en qué grado.

Capítulo 2

Objetivos

Con los precedentes consignados en el capítulo anterior, se procede ahora a detallar los objetivos generales y específicos del proyecto. A continuación, se plantean los distintos puntos en orden de realización a lo largo de la vida del proyecto software.

2.1 Objetivo general

El objetivo del proyecto es la realización de un juego que pueda ser distribuido y puesto a la venta en múltiples plataformas. Para esto, se tendrá que realizar el diseño y desarrollo del mismo.

Con el fin de diseñar un juego adecuado a las prerrogativas actuales en el mercado, se realizará el diseño de personajes, mapas, distintas habilidades y clases que componen las unidades, qué narrativa va a seguir el juego durante todo su desarrollo y qué tipo de comportamiento tendrán las unidades enemigas a lo largo del juego. Asimismo, se definirán interfaces correctamente legibles y usables adecuadas al medio en el que se realiza el proyecto.

Con la estructura definida para el proyecto software, se comenzará la programación de las distintas clases que componen el proyecto, así como el diseño de las diversas pantallas, animaciones, elementos visuales e interfaces necesarias para el correcto funcionamiento de aquel. Una vez hecho esto se procederá a la revisión de errores y optimización del código realizado para un mejor funcionamiento.

Cuando se considere que el trabajo está completado, este será puesto a la venta mediante una plataforma correcta de distribución de aplicaciones, que permita una amplia difusión del producto.

2.2 Objetivos específicos

Ahora que se posee una idea más precisa del concepto que se pretende realizar, se procederá a especificar los distintos objetivos con más detalle, ahondando en estos para su correcta definición.

2.2.1 Diseño de sistema multiplataforma

Ya que el sistema debe ser compatible con múltiples sistemas operativos y diferentes dispositivos, es preciso analizar los elementos que permitan que lo sea:

- Elegir una plataforma de desarrollo que se adecúe a las necesidades multiplataforma del sistema.
- Desarrollar algoritmia que permita el funcionamiento de los distintos eventos del programa en todas las plataformas.
- Diseñar interfaces compatibles con múltiples resoluciones de pantalla, de manera que estas se adapten de forma automática.
- Analizar cuál sea la mejor forma posible de interacción del jugador con el juego, evaluando con pros y contras qué controles son los más adecuados para ello.
- Generar un sistema de almacenamiento de datos que cubra las distintas eventualidades posibles, ya que estos habrán de ser guardados en distintos sistemas operativos.

2.2.2 Diseño de niveles

Con el fin de poder crear un sistema que genere mapas que hagan posible la expansión del juego en caso de ser necesario, se realizarán los siguientes pasos:

- Diseñar el sistema de forma que este sea escalable, haciendo que este permita la inclusión de nuevas unidades y diseños.
- Definir con claridad los principios visuales de los elementos del mapa de modo que estos sean legibles y usables y expresen con claridad el estado del mismo en todo momento.

- Optimizar el espacio requerido en memoria por parte de los mapas para que estos puedan generarse con el menor número de recursos necesarios.

2.2.3 Desarrollo de los combates

Los combates son el pilar central en el que se apoya el juego, por lo que su mecánica es la que debe ser analizada con mayor detalle a fin de crear una experiencia de usuario satisfactoria y desafiante. El conjunto de objetivos que han de alcanzarse a este respecto es:

- Definir y precisar las reglas de combate, encontrando un sistema que permita al usuario superar los niveles mediante la estrategia y la toma de decisiones tácticas durante el combate.
- Incluir elementos visuales que enfaticen el impacto de las decisiones del usuario, como animaciones y efectos de sonido.
- Desarrollar un código que calcule de forma eficiente y rápida las acciones tomadas tanto por el jugador como por el computador.
- Crear patrones de combate para la IA de manera que esta reaccione de forma diferente a lo largo de los múltiples niveles, siendo sus decisiones evidentes para el jugador, haciendo posible que este pueda identificar los patrones y actuar en consecuencia.
- Diseñar opciones de combate intuitivas para el usuario, de forma que no necesite más de unos segundos para discernir qué elementos en pantalla le permiten interactuar con las unidades.
- Los combates deben mostrar de forma clara y precisa las condiciones de victoria y derrota, que serán variables en el transcurso del juego.

2.2.4 Distribución y comercialización del producto

Una vez producida la versión final para distintas plataformas, se procederá a la distribución de estas en una web comercial tras haber llevado a cabo un análisis previo de los pros y contras de las distintas plataformas disponibles. Para que esto sea posible se tendrá que:

- Realizar un estudio de mercado que permita tener una perspectiva clara de la situación en la que se encontrará el producto al ser distribuido.
- Analizar los posibles competidores que existen actualmente, y cómo diferenciar significativamente el juego de los demás en el mercado.

Capítulo 3

Estado del arte

En este capítulo se analizan las distintas herramientas utilizadas en la creación y diseño del proyecto software, justificándose en su contexto por qué han sido elegidas como las herramientas idóneas para tal fin. Además de esto se aporta el estado del conocimiento existente relativo a videojuegos similares al del presente proyecto, así como la situación actual de las distintas webs para su distribución.

3.1 Juegos RPG

Como ya ha sido mencionado anteriormente, los juegos RPG son un género bastante extendido dentro de los múltiples que configuran el mapa de los videojuegos, género que cuenta con una gran cantidad de ellos publicados en la última década. Aunque todos ellos tienen patrones comunes, sin embargo pueden encontrarse diferencias reseñables, principalmente cuando se trata de distintas plataformas. Teniendo esto en cuenta, al analizar los distintos juegos en los medios en los que han sido publicados y ver cómo operan actualmente, es posible obtener una información clara y precisa del estado actual de modelos semejantes.

3.1.1 Plataformas móviles

En los últimos años la cantidad de juegos distribuidos para plataformas móviles de corte RPG táctico ha aumentado masivamente. Esto se debe a la facilidad que proporcionan las pantallas táctiles para el control y manejo de unidades en el mapa, así como a un mejor entendimiento por parte de la industria de las necesidades de los jugadores en este género: nos referimos a elementos como claridad a la hora de visualizar el mapa o una buena narrativa (a diferencia de sus precursores, que prácticamente se han limitado a una mera implementación iterativa en los dispositivos

móviles de juegos antiguos diseñados para otras plataformas, provocando así problemas de interfaz que se generan por no haber sido creados pensando en esta plataforma).

Ejemplos actuales de juegos de este género que tienen una interfaz correcta y un sistema jugable adaptado a pantallas móviles y que tienen en cuenta los problemas que se pueden plantear en estos dispositivos, serían juegos como Banner Saga, Fire Emblem Heroes o Terra Battle.

Figura 3.1: Ejemplo de RPG táctico: Banner Saga. Fuente: IGN.com

En la actualidad, el hardware de los dispositivos móviles ha alcanzado niveles que permiten reproducir juegos de gran potencia sin generar problemas relevantes, mediante el uso de procesadores computacionales de hasta 8 núcleos y procesadores gráficos de gran velocidad – con modelos actuales como los Adreno 6XX – y con pantallas de alta resolución. Si a esto unimos el notable incremento en el uso de estas plataformas, todo ello hace que se nos muestren como las que cuentan con un mayor potencial de crecimiento y más posibilidades de estar al alcance de un número de usuarios más elevado. Estas son las razones que motivan que en este proyecto se haya optado por estos dispositivos a la hora de diseñar y desarrollar el juego.[1]

[1] https://www.apple.com/es/iphone-xs/specs/
https://store.google.com/product/pixel_3_specs
https://www.mi.com/global/mi9/specs/

3.1.2 Sistemas Windows

Tradicionalmente los PCs venían siendo una de las plataformas con un mayor número de usuarios, algo que se vio incrementado a comienzos del siglo XXI. Sin embargo, con la llegada de los dispositivos móviles esa cantidad de usuarios se ha visto afectada experimentando una progresiva reducción. No obstante, el número de usuarios que existe para esta plataforma sigue siendo extremadamente alto y no puede ser menospreciado en absoluto. Otros dispositivos de sobremesa con sistemas operativos diferentes, como los sistemas Unix (Linux y macOS), no han desarrollado herramientas capaces de adaptarse al nivel de producción masivo de videojuegos que existe hoy en día, razón por la cual el sistema operativo Windows sigue siendo la opción por excelencia por parte de los estudios a la hora de crear un videojuego para estos dispositivos.

Actualmente, los ordenadores de alta gama diseñados para el uso de videojuegos tienen una potencia nunca antes vista en este campo, con procesadores computacionales de hasta 16 núcleos y procesadores gráficos capaces de realizar cálculos de hasta 16 TFLOPs. Por este motivo, los PCs siguen siendo la opción más viable para el desarrollo de un videojuego, ya sea en sistemas operativos Windows o Unix, dada su increíble potencia.[2]

3.1.3 Consolas

Pese a no estar contempladas en este proyecto, las consolas merecen una mención aparte por ser sistemas cerrados de producción de juegos que siguen siendo una opción bastante popular entre los consumidores, aunque su número de usuarios sea inferior al de sus otras dos competidoras directas.

La tendencia actual de las consolas es la de acercarse a los sistemas de PC, como hace Sony mediante el uso de arquitecturas x86 o Nintendo desarrollando sistemas que presentan gran similitud con los de los móviles actuales.

[2] https://www.amd.com/es/products/cpu/amd-ryzen-9-3950x
https://www.amd.com/es/products/graphics/amd-radeon-rx-5700

3.2 Motores gráficos

Los motores gráficos son la herramienta principal en el uso de videojuegos ya que permiten la automatización de múltiples tareas y ocultan la complejidad de procesos que suceden a bajo nivel. A continuación se describen las características de los motores gráficos y se analizan algunos de los más importantes.

3.2.1 Características de los motores gráficos

Los motores gráficos están compuestos por un conjunto de herramientas que permiten el desarrollo visual y de software de forma reutilizable, lo cual supone una optimización de tiempo y recursos necesarios para desarrollar los distintos elementos que componen un videojuego, mediante la repetición de elementos software y sistemas de renderizado gráfico, el sistema de detección de colisiones o el sistema de audio. Esto permite a los desarrolladores centrar sus esfuerzos en los componentes artísticos y en la lógica inherente a los juegos que han de producir. Se procede a describir brevemente los componentes en los que se puede dividir un motor gráfico:

- Hardware, drivers y sistema operativo. Los elementos de hardware están vinculados a la plataforma prevista para el motor de juego, con el objetivo de mejorar la eficiencia del juego para sistemas específicos. Los drivers permiten realizar a bajo nivel la gestión de múltiples dispositivos, como las tarjetas gráficas y de sonido. La capa de sistema operativo permite comunicar los procesos del motor gráfico con los recursos hardware asociados a este.
- SDKs y middlewares. El uso de bibliotecas externas e internas permite al desarrollador acceder a funcionalidades específicas.
- Capa independiente de la plataforma. El software de esta capa está orientado al desarrollo de sistemas multiplataforma. De forma más específica, unifica procesos que no son exactamente estándar, como por ejemplo la diferencia entre C, C++ y C#, haciendo que el desarrollador pueda ignorar los problemas que implica el desarrollo en múltiples plataformas.

- Subsistemas principales. Existe un conjunto de herramientas y utilidades que permiten al motor gráfico realizar operaciones de una complejidad significativa. Entre otros, destacan la biblioteca matemática, las estructuras de datos y algoritmos, la gestión de memoria y la depuración y logging.
- Gestor de recursos. Este componente del motor gráfico proporciona una interfaz unificada para acceder a los distintos elementos software que componen el motor.
- Motor de rendering. Este componente se ocupa de la generación de gráficos con el fin de mostrar los elementos visuales que conforman el videojuego. El renderizado de los elementos se basa en múltiples API, tanto de bajo como de alto nivel, dotando de independencia al hardware del motor.
- Herramientas de depuración. Este módulo permite la revisión y monitorización del motor gráfico, permitiendo mejorarlo analizando métricas como velocidad de ejecución, rendimiento, uso de memoria, etc.
- Motor de física. Este sistema gestiona los elementos relacionados con la colisión de elementos en el juego, las interacciones entre objetos y las mecánicas destinadas a simular el realismo de elementos de forma dinámica.
- Interfaces de usuario. Con el fin de tratar los eventos introducidos por el usuario, el sistema de interfaces actúa de puente entre los elementos de bajo nivel y los más altos, asignando los eventos a la lógica de estos.
- Networking y multijugador. La mayoría de los motores gráficos actuales tienen un componente diseñado para manejar interacciones online entre usuarios de un juego, el cual hace posible el envío de información entre los mismos.
- Subsistema de juego. Este módulo agrupa los diversos elementos del juego que afectan al funcionamiento de este, siendo aquí donde se encuentran los sistemas de scriptings y de lenguaje que definen el comportamiento y patrones de los elementos.
- Audio. El componente de audio de los videojuegos es importante para la inmersión y feedback auditivo de los jugadores, ocupándose ese elemento de manejar los distintos sonidos y la música.

Puede hallarse información en la que se profundiza sobre estos elementos en la siguiente referencia [VM17].

3.2.2 Unreal Engine

Unreal Engine es un motor gráfico creado por el estudio norteamericano Epic Games, creador de juegos como Gears of wars, Fortnine o Unreal Tournament. Actualmente en su versión 4, este motor gráfico se centra en el fotorrealismo de su motor de renderizado, dotando a los elementos que aparecen en pantalla de un aspecto cinemático, gracias a su uso dinámico de la iluminación y a la capacidad de manejo de una enorme cantidad de partículas en pantalla en tiempo real. A continuación se mencionan algunos de los aspectos más significativos del motor, tras un análisis de la documentación provista en su web[3].

Una de las características más llamativas de Unreal con respecto a otros motores gráficos del mercado es que no necesita el uso de scripts ni conocimiento de código alguno para funcionar. Gracias a unos assets específicos para Unreal, denominados Blueprints, la dificultad de acceso a este motor gráfico por parte de los desarrolladores con pocos conocimientos de programación se ve enormemente reducida. Estos Blueprints permiten el control, implementación y modificación de cualquier elemento que pertenezca al proyecto, así como interacciones con el entorno, cámaras, menús, etc. En el caso de querer un nivel de control más preciso y personalizado, los scripts siguen siendo una opción, al utilizar estos el lenguaje de programación C++ para su funcionamiento. Esto dota a los scripts de flexibilidad para el control del lenguaje a alto y a bajo nivel, vital en proyectos de gran envergadura [Sev15].

Cabe destacar que Unreal engine posee un editor visual que permite la creación y previsualización de partículas en tiempo real, lo cual, combinado con su editor de secuencias, permite la realización del paradigma de escenas cinematográficas que busca generar. Es interesante observar que Unreal ha optado por crear herramientas específicas en su motor gráfico enfocadas a la creación de los múltiples elementos que componen los juegos de mundo abierto (un género de juegos donde todos los elementos se encuentran disponibles para el jugador en un entorno sin pantallas de carga), teniendo herramientas de apoyo para follaje, paisajes y terreno[Lee16].

[3] https://www.unrealengine.com/en-US/features

Además de lo anteriormente mencionado, este motor de juegos tiene un soporte directo de juego multijugador, con un Framework escalable con estructura de cliente/servidor ya creada que sirve de soporte para diseñar la estructura de juego online entre múltiples jugadores.

Unreal Engine 4 es completamente gratuito; no obstante, cualquier juego producido utilizando este motor tiene la condición de que el 5% de todos los beneficios producidos por este es para Epic Games, ya sea a través de compras de la aplicación o de pagos realizados directamente en el propio juego. La única excepción a esto es si el juego produce menos de 3000 dólares cada cuatro meses.

3.2.3 CryEngine

CryEngine es un motor gráfico creado por Crytek para el videojuego FarCry, que luego refinó y puso posteriormente a la venta. Las pretensiones de la compañía con respecto al motor son bastante ambiciosas: la creación del motor gráfico más potente del mercado, capaz de abarcar todas las opciones posibles en la creación de un videojuego[4].

Esto se refleja en un motor gráfico de muchísima potencia, elevado consumo de memoria y una complejidad mayor que la de otros de sus semejantes. A cambio, CryEngine provee a los desarrolladores de la posibilidad de crear proyectos de enorme complejidad, con gran capacidad de desarrollo a bajo y alto nivel en entornos 3D.

En el aspecto visual, el uso de los Lens Flare, destellos mediante iluminación similares al que provocaría una lente al dejar pasar la luz, permite estilizar el diseño de los juegos con una técnica similar a la que se utiliza en la industria del cine. Los efectos de esta iluminación pueden ser observados en tiempo real. Otra técnica utilizada para mejorar el aspecto gráfico de los videojuegos creados es la que permite crear reflejos en superficies reflectantes como agua o metal en tiempo real, mediante su tecnología Real-Time Local Reflections. Sumado a esto se incluye un efecto de Motion Blur de alto nivel unido a una gran capacidad de anti-Aliasing, aumentando el rendimiento y la calidad gráfica de los juegos hasta alcanzar un nivel extremadamente alto.

[4] https://www.cryengine.com/features

En el aspecto más técnico, el editor de CryEngine destaca por su capacidad para importar elementos no creados en el sistema e integrarlos sin problema. Es un editor de niveles diseñado para poder crear múltiples niveles en un tiempo record mediante sus herramientas de pincel, que crean objetos en las escenas, y un sistema de sandbox, que permite programar y depurar en tiempo real todos los elementos que se están desarrollando en pantalla; asimismo cuenta con un sistema de rebobinado y análisis, vistas previas y edición en tiempo de ejecución, por lo que el desarrollador tiene un total control sobre la situación en todo momento[Gun14].

Los scripts están disponibles en C++, Lua y C#, permitiendo la interacción entre ellos sin problema, dando así la posibilidad de utilizar el lenguaje de programación que resuelva de la forma más eficiente el problema planteado en una subsección del proyecto.

El acceso a CryEngine no es gratuito, requiriendo un pago mensual de 10 dólares. A cambio, todos los beneficios generados de la venta del juego van directamente a los desarrolladores, sin cobro de porcentaje alguno por parte del motor. Esto refleja una filosofía de venta enfocada para grandes proyectos que conllevan la inversión de una gran cantidad de tiempo y la intervención de un elevado número de personas a la hora de trabajar con el motor.

3.2.2 Unity Engine

Unity es un motor de videojuegos multiplataforma desarrollado por Unity Technologies en 2005. Como tal, Unity permite la creación de videojuegos para iOS, Android, Windows, Mac, Linux, PlayStation4, XboxOne, AndroidTV, PlaystationVR y Nintendo Switch, entre otros. Asimismo ofrece servicios 2D y 3D, con el objetivo de ser una herramienta multifuncional adaptada al mayor número de plataformas posibles. De entre todos los motores gráficos mencionados, es el que tiene el mayor número de plataformas en las que distribuir el proyecto sin realizar cambios relevantes en cómo programar las diferentes versiones[5].

[5] https://docs.unity3d.com/Manual

Un rasgo significativo de Unity es que no tiene pretensiones de crear la mejor calidad gráfica del mercado con su motor. Es un motor capaz de generar elementos gráficos en 2 y 3 dimensiones en plataformas móviles, las cuales no necesitan una potencia tan alta como para usar Unreal o CryEngine, puesto que son modelos menos potentes que las consolas o los PCs. A cambio, Unity se centra en un sistema de renderizado extremadamente ágil, con un Shader (sombreado) usando físicas reales y una iluminación global en tiempo real que permite simular efectos visuales creíbles, todo ello con un consumo de recursos bajo, optimizando así la velocidad de carga y la eficiencia del juego.

Unity referencia de forma separada su sistema 2D, ya que, pese a que muchas funciones del motor son comunes en 2 y 3 dimensiones, el sistema tiene herramientas específicas para optimizar el desarrollo de este tipo de juegos. Elementos que sobresalen en este punto son: el sistema de ordenación de capas, que permite organizar la visibilidad de los elementos de forma intuitiva; el sistema de sprites, que gestiona las texturas 2D y las imágenes que utilizan los objetos de la escena, junto a un sistema creado directamente en el editor que permite retocar y manipular las imágenes; un sistema de mapeado de casillas, que genera de forma visual un mapa con edición simple mediante el uso de pinceles y un sistema de físicas en 2D, que permite crear efectos como aceleración, desplazamiento e impactos[Jac14].

El editor de Unity está diseñado para poder pausar en cualquier momento el contenido que se está ejecutando, posibilitando modificar aquello que se desee y reanudar la ejecución sin guardar los cambios, permitiendo así al desarrollador experimentar con las diversas opciones que tiene a su alcance sin riesgo a perder la iteración inicial cuando se empezara a ejecutar.

Además de los sistemas tradicionales para producir juegos, Unity provee de herramientas para el desarrollo en todas las plataformas actuales de realidad virtual, como Oculus, GoogleVR, PlaystationVR y otras, convirtiéndose de este modo en una opción muy atractiva para este tipo de juegos.

A nivel de Scripting, Unity tiene un sistema estándar de scripts asociados a objetos del entorno, mediante los cuales cada uno de estos opera de forma independiente al resto en la escena. El lenguaje utilizado en estos scripts por defecto es C#; no obstante,

Unity provee compatibilidad .NET mediante el uso de bibliotecas DLLs, ampliando así las opciones de desarrollo de Scripts.

Unity tiene un modelo de licencia pensado especialmente para estudios pequeños consistente en un sistema de suscripción. Esto permite que pequeñas compañías o desarrolladores independientes que generen beneficios inferiores a 100.000 dólares anuales puedan trabajar de forma gratuita con este motor gráfico.

3.3 Webs de distribución de aplicaciones

Con todos los avances que están teniendo lugar en el campo del entretenimiento multimedia, y concretamente en el de los videojuegos, no es de extrañar que múltiples empresas importantes hayan concurrido para ejercer la función de distribuidoras de pequeñas compañías y desarrolladoras independientes que no tienen los medios para hacer llegar su producto a un amplio espectro de individuos. A continuación se nombran las más relevantes en relación con la realización de este proyecto.

3.3.1 Steam

Steam es una plataforma de distribución y venta de videojuegos que comenzó como una distribuidora privada de juegos, siendo a todos los efectos una tienda para juegos de PC, pero que con el paso del tiempo ha ido permitiendo que los usuarios puedan subir sus propios juegos a la web. Estos juegos no suben inmediatamente a esta, sino que deben pasar por un proceso de validación de dos fases: en la primera, el juego debe ser validado por otros usuarios para ser aceptado, lo cual conlleva el potencial peligro de que un juego, que podría ser valorado positivamente en otras situaciones, no salga adelante debido a las valoraciones negativas de los usuarios por motivos ajenos a la calidad del juego; en la segunda fase, el juego debe ser validado por Steam (cuyo propietario es Valve), permitiendo así la puesta en venta del juego a cambio de un porcentaje de los beneficios. El problema es que las políticas de Steam favorecen a los juegos que generan grandes beneficios, pues permite a estos obtener un porcentaje de beneficios mayor que el que permite obtener a juegos que generan un beneficio total

menor. Esto favorece a las empresas desarrolladoras más grandes, puesto que su potencial para generar juegos con beneficios elevados es superior. [Kuc18]

3.3.2 Google Play y App Store

Los creadores de dispositivos móviles de mayor prestigio (Google y Apple) tienen sus propias tiendas en las que se pueden subir juegos para sus sistemas operativos (Android y iOS). Ambas plataformas ofrecen un medio de distribución oficial que está integrado de forma nativa en sus sistemas operativos correspondientes. Gracias a esta difusión que aporta el hecho de que estén en todos los dispositivos, estas tiendas se han convertido en medios de difusión extremadamente amplios para aplicaciones específicas del dispositivo en el que se encuentran.

3.3.3 itch.io

La mayoría de web de distribución y venta de videojuegos han sido históricamente específicas para una plataforma, hasta el punto de que si se entra en una web de juegos de PC, no es posible obtener juegos de una plataforma Android o iOS, o viceversa. La web itch.io ofrece la opción de subir distintas versiones de un mismo juego, cada una de estas para una plataforma diferente, con una tarificación del 30% para cualquier punto, lo cual va en consonancia con la política de ser una web para pequeños desarrolladores indie.

Capítulo 4

Metodología de trabajo

El siguiente capítulo recoge la metodología seguida a la hora de realizar este proyecto con una explicación de cómo funciona, la justificación de por qué elegir dicha metodología y cómo se va a aplicar esta al diseño, creación y publicación comercial del proyecto.

4.1 Elección de la metodología de desarrollo

Dada la naturaleza de los videojuegos, es necesario analizar con sumo cuidado qué metodología de trabajo se debe seguir. Todo desarrollo de un juego requiere de incrementos iterativos a lo largo del tiempo, en los cuales al principio se desarrollan mecánicas base que son fundamentales para el juego, añadiendo complejidad al proyecto en sucesivos ciclos de vida. A esto se suma la posibilidad de cambios en los requisitos del proyecto a lo largo de dichos ciclos, por lo que es necesaria una metodología de trabajo que permita adaptabilidad y evolución a lo largo de todo el ciclo de vida del proyecto. De entre las opciones que permiten afrontar un desarrollo en estas condiciones, Extreme Programming (XP en adelante) parece ser la opción más adecuada, ya que se centra en el desarrollo del software por encima de la gestión del proyecto; no obstante se ahondará en esto en los siguientes puntos.

4.1.1 Metodología de trabajo ágil XP

XP es una metodología de trabajo que proporciona un marco para el desarrollo de software ágil, cuyo objetivo principal es la creación de software de alta calidad con la mayor calidad de vida del grupo, o en este caso del individuo, que conforma el proyecto software. Gracias a esta flexibilidad que permite que el número de personas relacionadas con el proyecto pueda ser variable, es una metodología perfecta para

parejas o desarrolladores en solitario. Más información acerca de XP puede ser localizada en la siguiente referencia[BA04].

Existen una serie de condiciones para que XP pueda ser aplicable, listándose y evaluándose a continuación[6]:

- Requisitos software que pueden cambiar dinámicamente a lo largo de todo el proyecto: Dado que algunos de los componentes que integran un juego pueden no cumplir las expectativas deseadas (no resultan divertido, no son claros, no encajan narrativamente, etc), es necesario aplicar cambios que modifiquen lo previamente definido.
- Riesgos provocados por el uso de nuevas tecnologías en proyectos de tiempo limitado: Los motores gráficos se actualizan a un ritmo rápido en el desarrollo del software, llegando a cambiar radicalmente en un año en algunos casos. Una metodología capaz de prever dichos riesgos y adaptarse a las actualizaciones permite crear un juego más eficiente.
- Grupos muy reducidos de desarrollo: Al tratarse de un proyecto individual, es necesario un sistema de acercamiento a la creación del software pensado para uno o más individuos.

4.2 Aplicación de la metodología de trabajo

En el siguiente punto se procede a explicar la aplicación al proyecto de los distintos puntos que componen el proceso XP, explicando cuáles de estos se ven reflejados directamente en el ciclo de vida del desarrollo software a lo largo de todo el proyecto, explicando para ello las cuatro actividades básicas en que se divide la metodología XP y como esta se ha aplicado, señalando qué diferencias hay a la hora de aplicarlo a un modelo individual.

[6] https://www.agilealliance.org/glossary/xp/

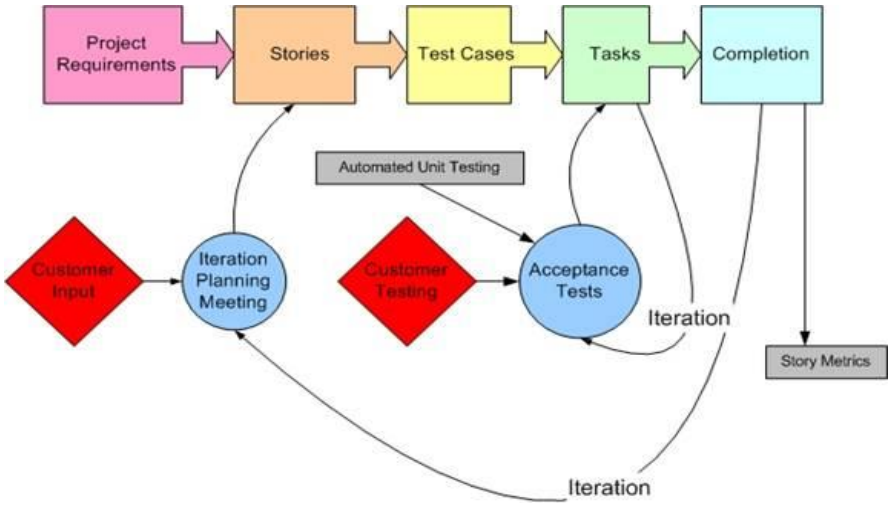


Figura 4.1 Proceso XP. Fuente: usml.edu

1. Programación: La prioridad máxima del Extreme Programming, alrededor del cual pivotan el resto de reglas, por la cual a lo largo de un ciclo debe generarse un programa funcional que será incrementado en sucesivas iteraciones. Esto ha sido aplicado mediante la concepción de los distintos mapas y unidades, que han empezado con un modelo simple y una única unidad y han ido incrementando su complejidad a medida que el desarrollo ha avanzado.
2. Pruebas: Todo código debe ser probado al final de un ciclo, intentando realizar todos los casos posibles en los que este pueda ser comprometido o fallar, refactorizando esta en caso de fallar. Toda prueba tiene que tener como finalidad la mejora del software. En el caso del proyecto, analizar qué elementos podían generar fallos y probar el comportamiento de la máquina en sus distintas variantes lógicas, intentando encontrar errores en la lógica, así como fallos de interfaz que pudieran provocar errores en la visibilidad del resultado final.
3. Escuchar: Esta fase se basa en la comunicación y conversación entre los miembros del grupo que conforman el proyecto. En este caso no ha sido posible realizar un análisis en grupo durante la creación de este por ser un proyecto individual. No obstante, en la fase de beta (en la que el producto funciona correctamente pero no ha sido lanzado aún al mercado), el videojuego se somete a la prueba de usuarios, que dan feedback sobre qué elementos del juego deberían ser modificados o mejorados.

4. Diseño: El diseño tanto de las interfaces como de la estructura básica en la que se asienta el proyecto debe ser simple pero eficiente, capaz de afrontar cambios de la forma más rápida posible. El diseño inicial de diagramas de clase, de secuencia y de casos de uso han permitido al programa adaptarse a los sucesivos ciclos, permitiendo cambiar y modificar elementos de forma dinámica.

Capítulo 5

Arquitectura del videojuego

En este capítulo se abarcan los distintos elementos que han compuesto el proyecto, concretando y definiendo su estructura, y realizando un análisis del funcionamiento de las distintas pantallas. Para esto se realiza un pequeño análisis previo de diseño del juego, para luego definir de forma técnica mediante diagramas la estructura general.

5.1 Desarrollo del concepto del videojuego

Conforme a lo ya explicado en los capítulos precedentes, se procede a definir con detalle los elementos que componen este videojuego RPG táctico en sus diseños iniciales.

5.1.1 Diseño de mapas

Se define, por las dimensiones en las que se realiza el proyecto, que los mapas tengan una dimensión de 6 casillas de ancho por 8 de largo, en las cuales las unidades aliadas y enemigas, así como obstáculos no caminables por estas, puedan ser colocadas previamente, intentando buscar el equilibrio entre la dificultad y que los mapas sean visualmente interesantes. Ambas características se ven incrementadas en los mapas posteriores a los iniciales, ya que los primeros se diseñan con intención de que el jugador aprenda las mecánicas básicas del videojuego. A continuación se muestra la imagen conceptual del diseño de un mapa.

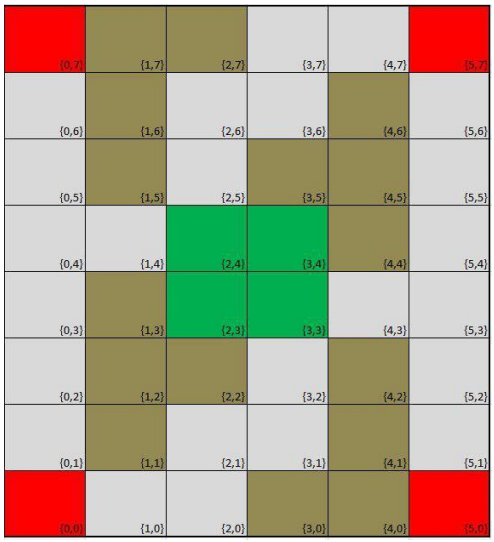


Figura 5.1: Esquema básico de un mapa a implementar

5.1.2 Interfaces

Los primeros diseños de interfaces intentan aplicar una filosofía WYSWYG (What you see is what you get) en la que todas las opciones disponibles para el jugador sean evidentes en el mismo momento en el que este interactúe con la pantalla. Debido a esto, el texto en las pantallas iniciales y de diálogo se colocan en la zona central de la pantalla, primera zona que un usuario mira cuando recibe nueva información en esta. En los niveles de combate la información debe quedar distribuida uniformemente sin afectar a la visibilidad del mapa de combate, puesto que el usuario necesita toda la información posible para tomar una decisión informada. De ahí que el mapa se coloque en la zona central de la pantalla, dejando en la zona superior la información obtenible de cada unidad y en la zona inferior las acciones posibles para esa unidad.

5.1.3 Mecánicas

Al ser un juego diseñado originalmente para plataformas móviles con su posterior adaptación a PC, la interfaz del juego no podía ser especialmente compleja, ya que la cantidad de opciones del usuario está limitada a una pantalla de tamaño inferior al habitual sin las opciones clásicas de juego, como son un teclado, mandos, etc. Por tanto, las opciones debían mantenerse claras y precisas a lo largo del juego, razón por la cual

se implementa un sistema por turnos, en el que en cada turno el propietario de uno de los dos grupos de unidades (jugador o computadora) realiza hasta dos tareas por turno: realizar una acción y moverse.

- Realizar una acción. Consiste en una interacción con alguna de las otras unidades del mapa, ya sea mediante una habilidad, que puede ser positiva o negativa para otra unidad, o mediante un ataque, que es la opción básica y primaria del juego para afectar a las unidades enemigas, algo que, no obstante, puede llevarse a cabo no solo contra los enemigos.
- Moverse. Permite desplazar a una unidad por el mapa, y se determina conceptualmente que las unidades pueden desplazarse cantidades variables según el tipo de estas.

5.1.4 Miscelánea

Se decide la trama de la historia. Debido a que el objetivo es dirigirse a un público joven, se toma la decisión de hacer una trama desenfadada sin más pretensiones que las de hacer pasar un buen rato al usuario.

Además de esto se escoge el estilo de música y sonidos adecuados para el tipo de juego, así como se toma la decisión de realizar animaciones para los movimientos de ataque y algunas habilidades específicas de los personajes.

5.2 Diagramas

En este apartado se muestran algunos de los diagramas más importantes del proyecto software, así como una breve descripción de su significado, y se detalla el esquema de clases que ha servido para estructurar el proyecto.

5.2.1 Diagramas de casos de uso

A continuación se muestran los diagramas de casos de uso de las escenas de inicio, diálogo y combate. Se descarta realizar diagramas de uso de las escenas de fin de partida y créditos debido a su simpleza.

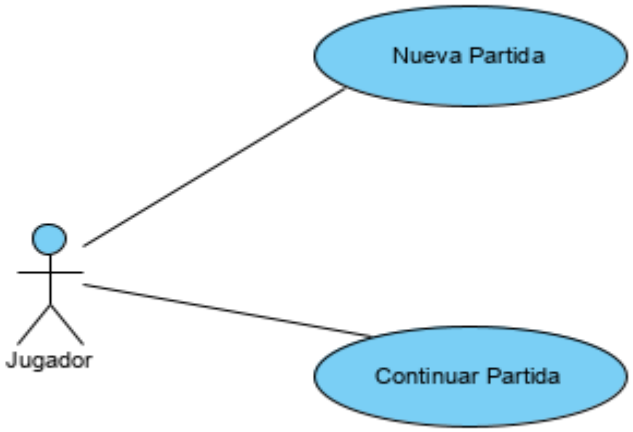


Figura 5.2: Caso de uso de la escena inicial

En la pantalla inicial se dictaminan los dos casos principales que tiene el jugador: Comenzar una nueva partida, iniciando el juego, y segundo lugar, reanudarla, cargando la partida que ya posee.

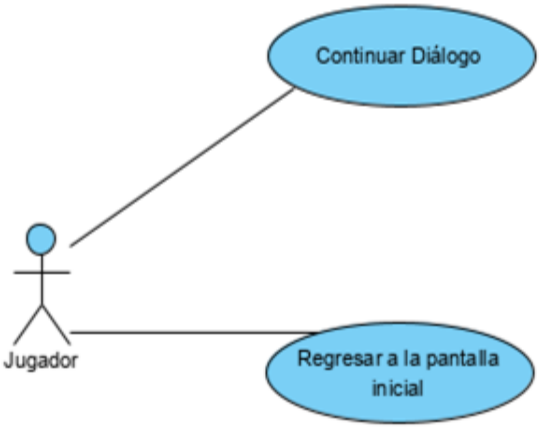


Figura 5.3: Caso de uso de la escena de diálogo

En la figura 5.3 se pueden observar las posibilidades que tiene el usuario en la escena de diálogo. Como es un elemento introductorio con trama, no existen muchas opciones. Avanzar en la narrativa o volver a la escena de inicio, ya sea porque no quiere seguir avanzando o porque ha cometido un error.

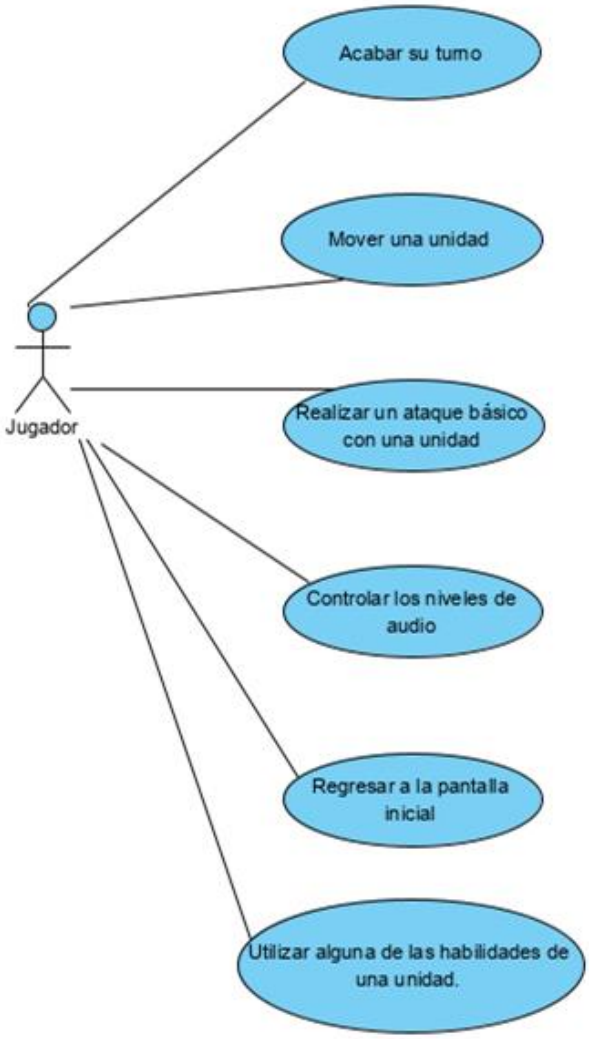


Figura 5.4: Caso de uso de la escena de batalla

El combate tiene la mayor cantidad de opciones del juego. El jugador puede realizar varias acciones en su turno, tales como acabarlo, mover unidades, hacer que estas ataquen o usen habilidades, así como ajustes técnicos de audio.

5.2.2 Diagramas de secuencia

Debido a la inmensa cantidad de posibilidades de los diagramas de secuencia de este proyecto, no es práctico plasmarlas todas en el documento, por lo que se exponen a continuación solo los diagramas más complejos y relevantes del proyecto. Se adjuntan en el Anexo B los diagramas de secuencia de movimiento válido y de ataque del jugador.

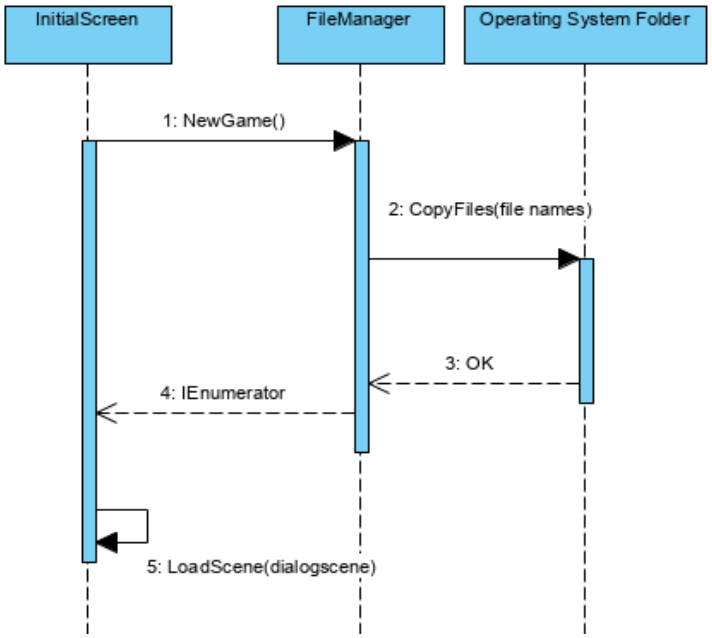


Figura 5.5: Diagrama de secuencia de comienzo de partida

En la figura superior se muestra el diagrama de secuencia asociado con el proceso de iniciar una nueva partida. Cuando el jugador pulsa el botón de partida nueva, se genera un evento que invoca a FileManager, el cuál copia todos los ficheros relacionados con una partida nueva desde el proyecto a una carpeta de guardado en el sistema operativo. Tras eso, la pantalla cambia a la nueva escena, la primera escena de diálogo del documento.

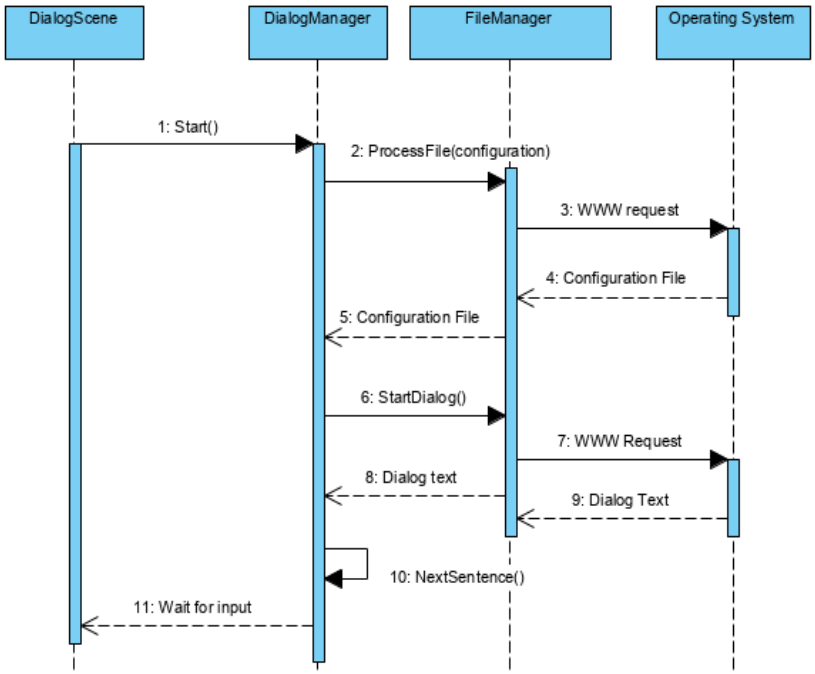


Figura 5.6: Diagrama de secuencia de inicio de diálogo

Cuando comienza un diálogo, se debe generar la primera línea sin que el usuario introduzca ningún tipo de input, para realizar este proceso, se debe invocar al gestor DialogManager, que interactúa con el sistema de ficheros y obtiene el archivo en cuestión de la carpeta externa situada en el sistema operativo. Tras esto, el diálogo comenzará con una animación de diálogo y, al acabar la línea asignada, quedará esperando a que el jugador pulse un botón para continuar.

5.2.3 Diagrama de clases

A continuación, se procede a mostrar el diagrama de relaciones entre clases, explicando las relaciones entre estos. Esta figura muestra las relaciones entre clases. Debido a su dimensión y con el fin de que el texto quede legible y ordenado, el contenido de todas las clases del proyecto se encuentra en el Anexo B bajo el título de

Contenido de las clases del proyecto. Además de esto, se ha dividido la relación entre clases en dos partes para que los componentes sean legibles en este documento.

La estructura general del proyecto es la de una arquitectura de tres capas, dividiendo las funciones de las clases en tres áreas bien diferenciadas: Persistencia, encargada del almacenamiento de datos; Dominio, responsable de la lógica del programa; y Presentación, que gestiona las interfaces y elementos gráficos[GHJ+94].

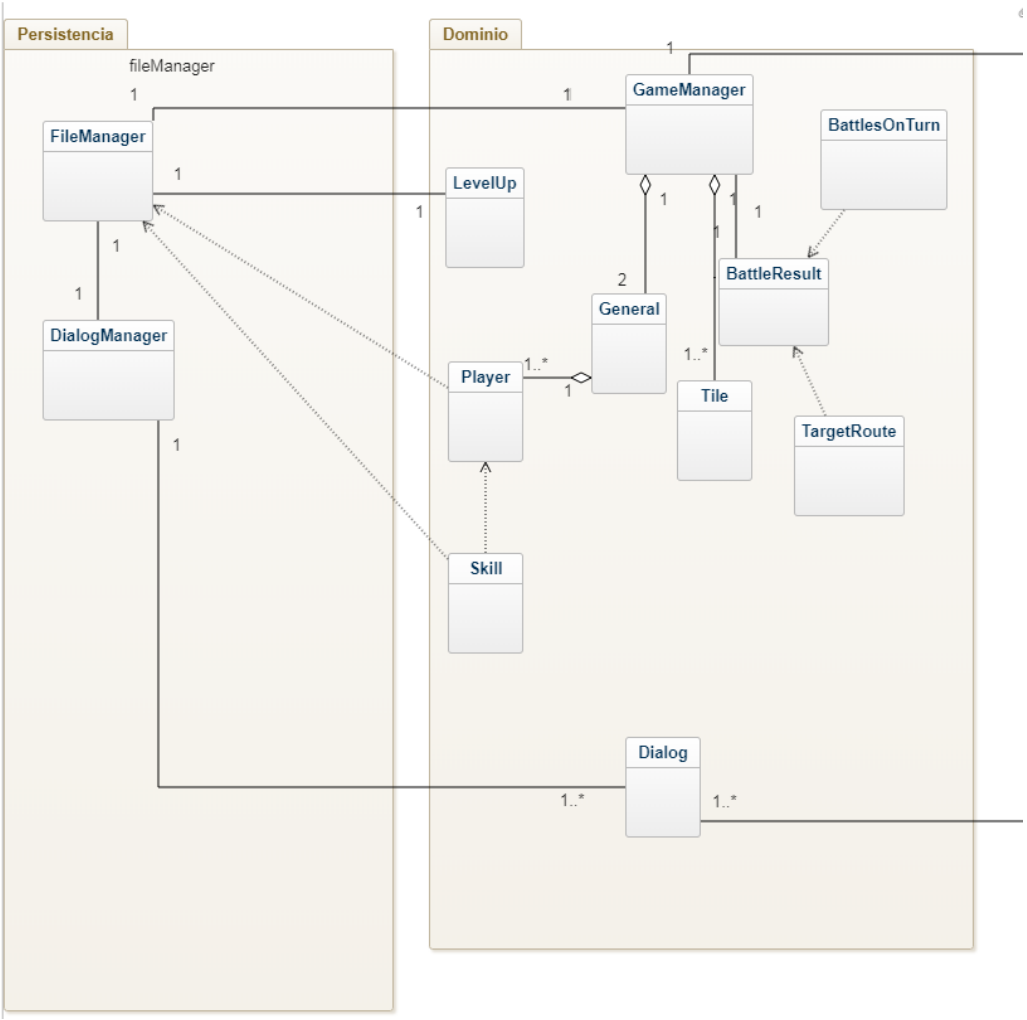


Figura 5.7 Esquema de relaciones de Persistencia y Dominio

En la capa de persistencia (Figura 5.7) del proyecto se encuentran las clases FileManager y DialogManager. Filemanager realiza la función de gestor de todos los ficheros del programa, con la excepción de algunos de diálogo, los cuales realiza en colaboración con DialogManager. La comunicación de esta capa con el resto del proyecto se realiza a través de GameManager, Dialog y LevelUp, gestores de Dominio de sus respectivas escenas.

Es importante hacer notar que estos gestores, así como BoardManager, utilizan una estructura de patrón Observer, existiendo una única copia de cada manager, que comunica y maneja los datos relacionados con todas las clases de su propia capa. Este patrón se usa en consonancia con el patrón Singleton para crear una única instancia de todos estos gestores. Los gestores son las únicas clases que se pueden comunicar con los gestores de otras capas, y nunca con otros elementos sin conocimiento de sus responsables. Ambos patrones son explicados con más detalle en [REF00].

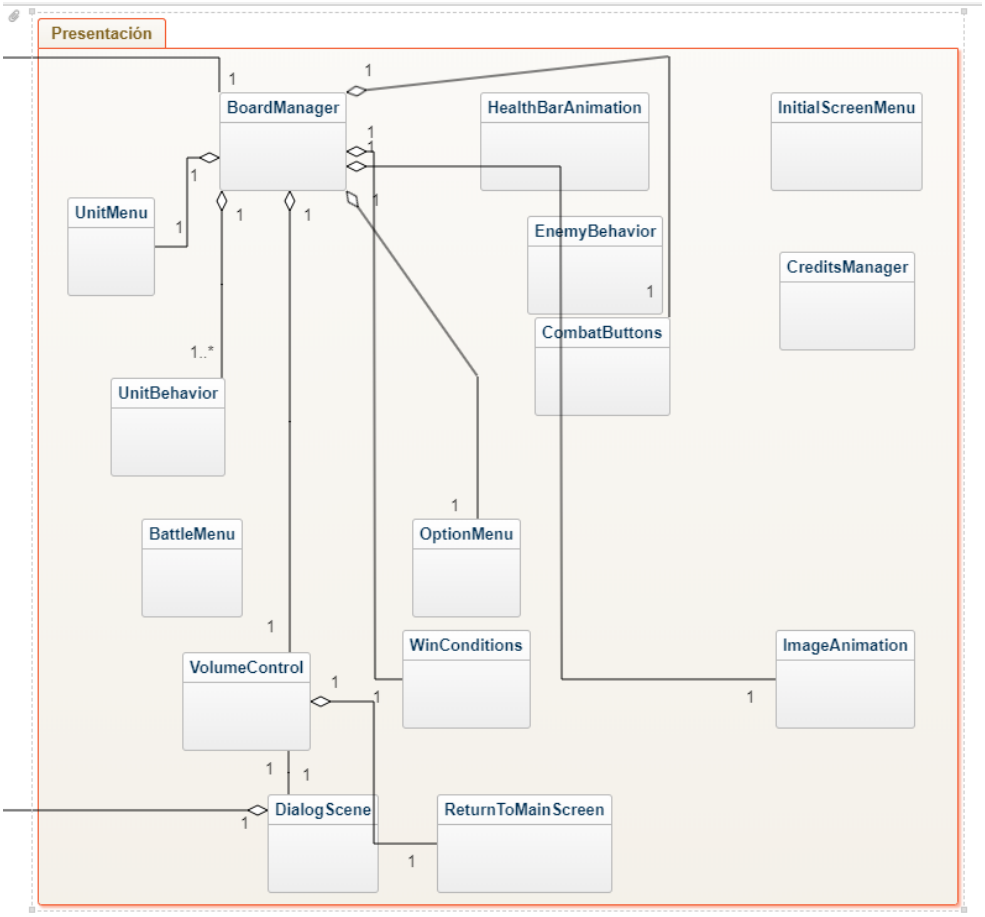


Figura 5.8 Esquema de relaciones de Presentación

En la capa de Dominio que se ve reflejada en la figura 5.7 se localizan 3 gestores. Dialog sincroniza los elementos de diálogo con la capa de presentación en la escena de diálogo, procesando los strings recibidos y creando los objetos necesarios. LevelUp recoge y analiza los datos relacionados con las unidades, decidiendo qué subida de parámetros tienen estos y comunicándolo a la interfaz de su escena. En último lugar tenemos GameManager, encargado de toda la gestión de lógica de combate. Este

manager tiene multitud de clases únicas. Este manager tiene conocimiento de los ejércitos de ambos bandos, la lógica y condiciones de victoria, la función de las distintas habilidades de las unidades y la lógica encargada de la IA del enemigo, gestionando todo el juego y creando todos los equivalentes visuales mediante llamadas a BoardManager para que los gestione.

En la capa de Presentación (figura 5.8), BoardManager se encarga de manejar, acceder y almacenar todas las instancias de elementos que solo van a aparecer una vez en la pantalla, tales como menús, y luego gestionar su visibilidad acorde a la lógica que le dicte GameManager. Además de esto, mantiene de forma ordenada el tablero visual del juego, transmitiendo las coordenadas desde su lado hasta la lógica donde estas coordenadas se convierten en una unidad lógica.

5.3 Escenas del proyecto

En los siguientes apartados se analiza el concepto, función e implementación de cada una de las escenas del proyecto, analizando cuales son los retos que estas plantean y cómo se implementan las soluciones para resolverlo.

5.3.1 Escena inicial

Esta escena es la primera que ve el usuario y es la que permite tanto iniciar el juego como continuar una partida que esté ya empezada. Está compuesta por los siguientes elementos: una imagen de fondo, un logo y dos botones, elegidos de forma que permitan que el texto sea legible y que ambos hagan posible una interacción visual y auditiva cuando sean clicados. A esto se suma una música de fondo apropiada para la temática del juego. El mayor punto de desafío con esta escena es la carga rápida de los ficheros.

Respecto al código asociado a la escena, se dispone de un script ligado a esta que recoge un evento OnClick para cada botón, convirtiéndolos en un evento de tipo OnPointerDown que posibilite que sean reactivos en plataformas móviles. El tratamiento de los ficheros se divide en dos partes: El acceso a ellos en una carpeta

especial mediante una corrutina perteneciente a la clase FileManager(encargada del manejo de ficheros) que copia los ficheros a la carpeta correcta para el funcionamiento del programa y el formato de estos, manteniendo la estructura y codificación necesaria para soportar caracteres latinos propios del castellano.

Las corrutinas en Unity son un tipo de función que permiten la gestión en paralelo de código. Tienen la capacidad de detenerse, pausarse o ser lanzadas con retardo. Este código realiza la carga de una imagen cuando se interactúa con alguna de las unidades del mapa. Para localizar dicha imagen existen dos formas de realizar el proceso: una en iOS y Windows, donde simplemente hay que cargar la imagen mediante el acceso al archivo en la ruta correspondiente; hay que tener en cuenta que iOS requiere que los archivos a cargar se encuentren en la carpeta StreamingAssets del proyecto, ya que es la que tienen asignada para la carga. Y otra forma en Android, en donde, pese a compartir este requerimiento de acceso a StreamingAssets para poder cargar imágenes, estas vienen codificadas en un archivo JAR. Por esta razón debe utilizarse la clase UnityWebRequest para la descompresión del archivo. En casos donde se requiera un control sobre el momento en que termina la función, se tendrá que utilizar WWW, ya que obligatoriamente UnityWebRequest requiere el uso de Yield en la función.

Figura 5.9 Estructura de formato StreamingAssets. Fuente: Unity.com

FileManager es el encargado de convertir los archivos de texto con los datos de juego en objetos funcionales del sistema. Como ya se ha mencionado, múltiples archivos del sistema se encuentran en la carpeta StreamingAssets que se utilizan en las plataformas Android y iOS para localizar archivos que deben ser accesibles en tiempo de ejecución. No obstante, Unity establece que para el acceso de archivos en modo de lectura y escritura no se debe usar StreamingAssets, que solo permite lectura, y que puede

generar problemas con los permisos de acceso del sistema operativo. De ahí que se utilice una carpeta de persistencia llamada PersistentDataPath, en la que se copian los archivos para poder acceder a ellos a lo largo del juego. El sistema de acceso a cualquiera de los archivos funciona de igual manera que la descrita anteriormente. Para acceder a un archivo concreto este puede ser localizado de forma normal mediante File.ReadAllText o métodos nativos a C# ordinarios, mientras que en Android los archivos se encuentran comprimidos en archivos JAR. Por este motivo se utilizan las clases UnityWebRequest y WWW según la necesidad, como ha sido comentado anteriormente.

Una vez se ha accedido al fichero, el string obtenido carece de formato, por lo que debe ser procesado, y teniendo en cuenta que el desarrollo de estos documentos ha sido realizado en Windows, los saltos de línea tendrán el formato \r\n, que no funciona en otros sistemas operativos.

5.3.2 Escena de diálogo

La escena de diálogo muestra los diálogos de la narrativa del juego. El contenido de estos se muestra en la zona central de la pantalla captando la atención del usuario. La escena está compuesta por una imagen de fondo, un contenedor del diálogo, así como imágenes que muestran qué personajes están hablando en esa línea de diálogo; estas se gestionan mediante GameObjects que poseen un componente denominado SpriteRenderer, que cambia dinámicamente la imagen a la correspondiente en los archivos de diálogo. Existe un botón que permite avanzar a la siguiente línea de diálogo y que está situado en la esquina inferior derecha de la ventana de diálogo, el lugar donde dirigirá la vista el usuario al ir acabando una línea.

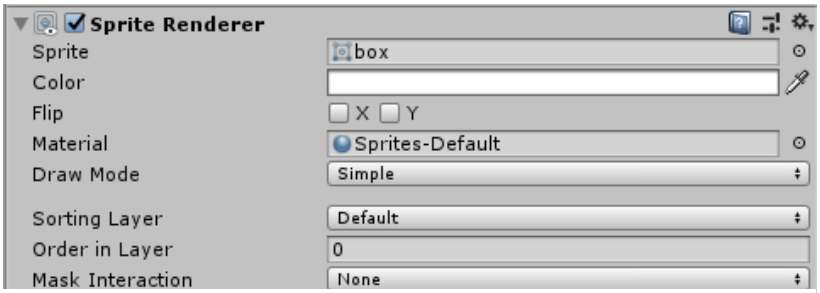


Figura 5.10 Elementos de Sprite Renderer

El código relacionado con la gestión de los diálogos se realiza mediante el uso de una cola de la clase Queue en la que se insertan todos los diálogos mediante una clase Dialog creada especialmente para contener todas las líneas de texto asociadas a un bloque de diálogo de un personaje.

El método DisplayNextSentence mostrado en la figura 2 del Anexo B realiza la función de coger los múltiples objetos Dialog creados al iniciarse la escena y procesar cada uno de ellos. Un objeto Dialog contiene múltiples líneas de texto y un nombre que identifica al personaje que habla. Siempre y cuando queden bloques de diálogo en el código, se asignará una línea al GameObject encargado de mostrar texto mediante una rutina llamada TypeSentence, que va mostrando letra a letra.

5.3.3 Escena de combate

Se parte de la base de que hay que construir un tablero de 6x8 que contenga todas las casillas del tablero. El encargado de realizar esta acción es la clase GameManager, gestora de la clase de lógica y que, al igual que BoardManager y FileManager, funciona mediante un patrón Singleton y aplica el patrón Observer a su capa correspondiente. Para crear el tablero, se utiliza la clase Tile, que contiene una clase generada en persistencia y que actúa de contenedor de las unidades: la clase Unit. Además de esto, Tile tiene un parámetro para saber si la casilla es caminable o no. Cuando FileManager, el gestor de la capa de persistencia, suministra mediante una petición de GameManager los parámetros de inicio del nivel, GameManager invoca a BoardManager, generando el componente visual del tablero.

Una vez obtenido el nivel del mapa general, se asigna un comportamiento a la lógica del enemigo, que actuará de tres formas diferentes. A continuación se asignan las unidades al ejército que les corresponda, el cual está estructurado como una lista de unidades.

Los niveles del juego tienen distintas condiciones de victoria y derrota que son mostradas al comienzo de cada combate con objeto de que el jugador tenga la información necesaria para saber cómo superar el nivel. Cuando la lógica dictamina que existe una condición de victoria o derrota, se encarga de llamar a una clase nativa de Unity, SceneManager, que actúa de gestor de las escenas existentes en el proyecto.

LoadScene carga una escena, ya sea con valores numéricos asignados a una escena o un string que representa el nombre de la escena.

A continuación se describen con más detalle los elementos que componen esta escena, describiendo la lógica que permite su correcto funcionamiento.

5.3.3.1 Mapa de combate

BoardManager genera el campo de batalla mediante un método CreateBoard que asigna GameObjects al mapa recorriendo un array bidimensional de GameObjects y utilizando el componente Transform de estos para asignar unas posiciones (x, y) en el mapa; los GameObjects tienen correspondencia con sus representaciones en la capa de lógica. Si bien este código es extenso, no supone una complejidad elevada ni posee un interés especial. Sin embargo, se va a analizar una rutina que es un componente de la asignación de imágenes por ser un código más relevante en esta parte.

La clase Unit almacena dos tipos de información. Por un lado, contiene la información de los parámetros de combate necesarios para el correcto funcionamiento de las batallas; por el otro, almacena el contenido de las habilidades asociadas a una unidad. Esta clase de almacenamiento Skill contiene información sobre el rango efectivo de ataque de la habilidad, así como el tipo de habilidad correspondiente. Un caso singular dentro de la clase Unit es la creación de una instancia denominada Mock, que permite a la lógica del juego partir de una unidad inválida en todos los casos de combate a la hora de comparar unidades.

Cuando dos unidades han sido seleccionadas para un combate, GameManager asigna, dadas las posiciones en el mapa por parte de BoardManager, la unidad atacante a la variable originUnit y la unidad que recibe el daño como targetUnit. Una vez realizado ese proceso, se realiza la lógica de combate, que depende de la habilidad seleccionada para la interacción entre dos unidades. En los casos de un combate con ataques básicos se realizarán los cálculos necesarios mientras que en el uso de habilidades targetUnit no tiene capacidad para responder. El resultado del combate, ya que el usuario puede elegir verlo y decidir cancelar el combate, es almacenado en una clase contenedora llamada BattleResult y que GameManager utiliza para ejecutar el combate en caso de que el jugador acepte. Debido a su excesiva extensión, el código referente a la lógica de

combate figura en el anexo adjunto a este documento en la sección listado de código como «Listado de código 1».

Ya que las unidades enemigas están controladas por la IA, esta debe ser capaz de realizar movimientos con las unidades cuando encuentren una unidad del jugador en su rango de alcance.

El algoritmo de movimiento de una unidad utiliza una técnica de backtracking para evaluar el árbol de decisiones posibles para el movimiento de una unidad en un turno. En cada llamada recursiva, la lógica del juego evalúa si la unidad está en rango de alguna otra rival, y decide si esta es un objetivo válido o no en base a la lógica definida en los tres estilos de combate que tiene asignados. En caso de encontrar un enemigo válido, este es añadido como miembro de la clase TargetRoute, una clase contenedora que tiene el listado de coordenadas de la ruta óptima hasta ese momento y el objetivo prioritario de esa ruta, de forma que al retornar de las llamadas recursivas se comparan las rutas y se obtiene la ruta de menor distancia para alcanzar el objetivo deseado. En caso de no encontrarse ninguna unidad, las unidades se desplazarán o no por el mapa según sea el estilo de combate.

5.3.3.2 Animaciones en combate

Cuando una unidad es seleccionada como enemiga tras pulsar algún botón de ataque (como ataque básico o alguna habilidad), se muestra un cuadro de diálogo en el que sale el posible resultado del combate. Estos cálculos los gestiona el manager de la capa de lógica y este diálogo simplemente tiene los valores.

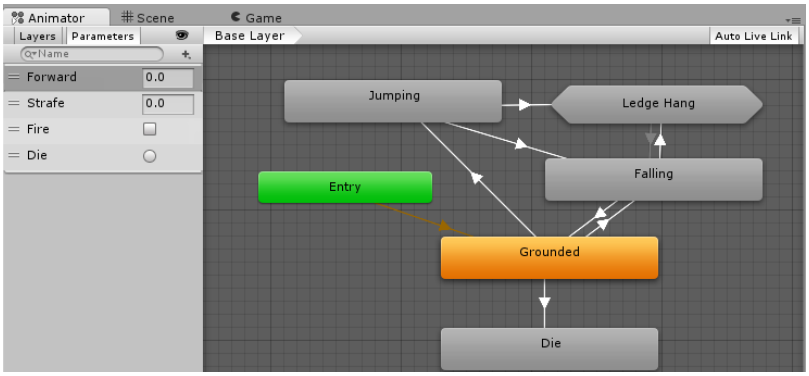


Figura 5.11 Ejemplo de estructura de un Animator en Unity.

Una vez confirmada la decisión de combate se realiza la carga de una animación, en la que en la parte izquierda se sitúa la animación de la unidad atacante y en la zona derecha la animación de la unidad atacada. Como un mismo intercambio entre unidades puede tener más de un ataque, las posiciones se cambian cuando tiene lugar más de un ataque.

La animación es un GameObject que es invocado únicamente cuando se le necesita y que pertenece al GameObject principal de la escena, el cual tiene una instancia de aquel asignada cuando la animación está desactivada por defecto.

SetupBattleAnimation es una corrutina que prepara todos los elementos de la animación mediante la asignación de valores en las barras de vida, las cuales son invocadas mediante la función InvokeRepeating, que a su vez permite llamar a una función de forma cíclica con un tiempo definido. Asimismo, realiza una asignación de las unidades correspondientes a cada sección del combate mediante su nombre. Una espera entre una llamada a la función y la siguiente, garantiza que la primera animación sea realizada antes de que una segunda pueda empezar a ser animada, encolando de forma visual los ataques.

5.3.3.3 Unidades

Las unidades son GameObjects que están compuestos por un componente SpriteRenderer, un Animator, un BoxCollider2D, un RigidBody2D y un script de comportamiento. A continuación se muestra la configuración de este en Unity. La única diferencia existente entre unidades del jugador y unidades controladas por la computadora es el script de comportamiento.

El script se compone de tres eventos, OnMouseDown, OnMouseDrag y OnMouseUp, y permite, conforme a las condiciones establecidas por las reglas del juego y expresadas mediante el control de BoardManager, gestionar el movimiento de la unidad en los momentos adecuados, pudiendo esta ser agarrada y desplazada a una nueva posición mediante Drag & Drop.

5.3.3.4 Interfaz de unidad

Cuando una unidad es seleccionada se muestran los valores de sus parámetros de batalla, así como los botones de las acciones que puede realizar en su turno. Este GameObject es único a lo largo de todo el combate, y sus valores cambian dinámicamente al clicar sobre una nueva unidad. Los valores de esta son consultados por BoardManager, que recibe la información de GameManager.

5.3.4 Escena de subida de nivel

La escena de subida de nivel muestra el aumento de parámetros de combate por parte de las unidades involucradas en la anterior escena de combate. Las unidades adquieren una cantidad de valores aleatorios para cada parámetro, y en determinados niveles se mostrarán dos botones que dan a elegir al usuario qué habilidad quiere para ese personaje. El menú solo avanza si se clica en la pantalla, y en caso de haber habilidades, si además de clicar se ha seleccionado una de las dos. Una vez terminado este proceso se carga la siguiente escena de diálogo, guardando previamente el estado de la partida para permitir continuarla cuando el jugador inicie otra vez el programa.

Para realizar estas funciones, hay que diferenciar entre los dos módulos principales: Uno encargado de la gestión de actualizar los parámetros de las unidades, mostrando el incremento de estos usando la invocación a la función nativa de Unity Update; el segundo elemento es el guardado de elementos mediante

Update es una función nativa de Unity que se invoca en cada frame y es utilizada frecuentemente para hacer comprobaciones en tiempo real de elementos de la escena. Durante esta función, los valores indicados de inicio y objetivo de cada parámetro se inician de forma secuencial, hasta que alcanzan el valor deseado. Una vez se gestionan todos los parámetros, esta animación deja de ejecutarse.

5.3.5 Escena de derrota

Esta escena simplemente muestra una imagen de FIN DE PARTIDA cuando el jugador es derrotado en combate. Una música acorde a la situación suena de fondo y el usuario solo tiene que pulsar la pantalla para volver al menú inicial. El script relacionado con la escena se asegura de que la pantalla de inicio es creada y la carga.

5.3.6 Escena de créditos

Esta escena se muestra al final de la partida cuando el jugador ha completado todos los niveles. La escena está compuesta por un elemento de texto en el que se muestran los distintos responsables de cada uno de los diferentes campos del desarrollo del juego. Este texto aparece en la zona inferior de la pantalla y asciende hasta desaparecer por la parte superior, tras lo cual el texto de «FIN» permanece en mitad de la pantalla, y a partir de ese momento el jugador puede pulsarla para volver a la escena inicial.

Para realizar este efecto, se genera un bloque de texto con todo el contenido a mostrar, y mediante la anteriormente mencionada función Update, se actualiza frame a frame el estado del Transform que contiene el texto, subiendo hasta alcanzar la posición deseada.

5.4 Componentes de Unity

En esta sección se procede a describir aquellos elementos que han sido una base en el desarrollo del juego y que son propios de Unity, tanto clases de código como componentes que existen en el editor de desarrollo.

5.4.1 GameObject

Unity utiliza como pilar central de su desarrollo una clase fundamental llamada GameObject, de la que heredan el resto de objetos que pueden existir en una pantalla (referidas cada una de las pantallas a partir de este punto a lo largo del documento como «escena»). Todo elemento que existe en estas escenas hereda de GameObject:

cualquier elemento, desde la cámara hasta los sonidos, pasando por las imágenes, personajes, texto, etc., son herencia de GameObject.

5.4.2 Frame Renderer

Este componente que permite cargar imágenes en la escena en diversos formatos, como jpg, png, etc.,. Además, este componente interactúa con un componente Animator, que permite animar una colección de imágenes y cargarlas frame por frame en el Sprite Renderer. Esto es especialmente útil a la hora de realizar animaciones.

5.4.3 Collider2D

Collider2D es un componente que permite realizar interacciones del usuario con eventos mediante la asignación de una zona de la escena a este GameObject, un concepto que en videojuegos se conoce como «hitbox».

5.4.4 Transform

Transform es un elemento esencial de los GameObject, puesto que es un componente que permite asignar posiciones vectoriales en el espacio para los objetos de la escena; en muchos casos los scripts modifican estos valores para desplazar elementos en ella.

Capítulo 6

Resultados

Este capítulo sirve para mostrar los resultados obtenidos al acabar el ciclo de desarrollo, mostrando las distintas pantallas del juego, así como un breve análisis de lo que se ha logrado en cada una de ellas. Además de esto, se incluyen imágenes adicionales del combate, al ser el elemento principal del videojuego. El contenido de este proyecto puede ser consultado y descargado en el enlace a pie de página[7]

6.1 Escena inicial

En esta escena se puede observar cómo los botones están diseñados de forma que no impiden la visión del fondo. Una música ambiental suena de fondo mientras el jugador decide si realizar una partida nueva o continuar la ya empezada. A nivel técnico, el rendimiento final de esta pantalla tras varias pruebas muestra un tiempo total de un segundo entre pulsar el botón de inicio y el comienzo de la partida.

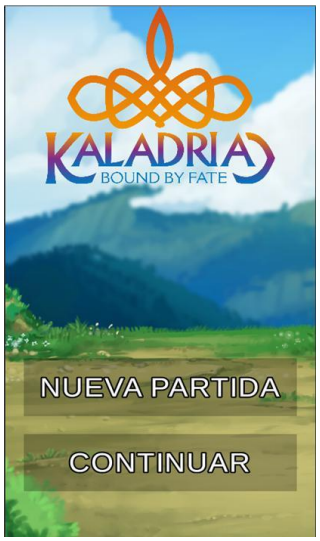


Figura 6.1 Escena Inicial

[7] https://bitbucket.org/Sublieme/kaladriaproject/

6.2 Escena de diálogo

Los diálogos presentan un sistema de respuesta rápida en el cual el usuario puede pasar frases sin necesidad de leerlas pulsando el botón de avanzar a la siguiente línea, lo que agiliza el proceso de rejugabilidad cuando el jugador se sabe los diálogos. Una opción de volver al menú principal y de control de música y sonidos se encuentra disponible en este nivel, y en términos generales, se considera satisfactoria la animación con la que se muestran los textos, generando una experiencia de usuario positiva. El fondo y las imágenes que representan a los distintos personajes que hablan se intercambian en tiempo de ejecución, ofreciendo una experiencia dinámica en la cual quedan reflejadas con claridad las interacciones de los personajes.

Figura 6.2 Escena de diálogo

6.3 Escena de combate

El combate se divide en dos elementos que son visibles en pantalla: En primer lugar, La interfaz y stats, que proporciona al usuario la información necesaria para tomar decisiones relacionadas con la gestión de sus unidades en el mapa. En segundo, los combates, que tienen una animación que representa el ataque de una unidad a otra, tras un cálculo previo del resultado para que el jugador pueda decidir si ese ataque es conveniente o no.

Figura 6.3 Interfaz en el combate

La parte superior de la pantalla se encarga de mostrar los parámetros relevantes al combate, mientras que la parte inferior muestra las opciones que tiene el jugador con la unidad seleccionada.

Figura 6.4 Selección de combate

Los botones responden generando el radio correspondiente a la habilidad seleccionada, y cambian de color según el tipo de habilidad para ayudar a diferenciar elementos. Los tipos de habilidades también tienen un icono distintivo entre sí, permitiendo agrupar las habilidades por sus efectos de forma intuitiva con solo un vistazo a la pantalla.

Seleccionar una unidad para realizar una acción genera un radio de acción para la unidad, que permite visualizar con claridad qué casillas tiene disponibles para realizar la acción. Mientras que los obstáculos impiden el movimiento, estos no impiden la realización de habilidades, que pueden ser utilizadas más allá.

Figura 6.5 Animación de combate

Las animaciones de combate se generan en tiempo real, utilizando la clase Animator para cargar la animación correspondiente en cada gameObject, utilizando los nombres de las unidades como referencia. Cuando múltiples animaciones quieren tener acceso simultáneamente al animador, estas quedan esperando durante un periodo de tiempo suficiente para que la animación que está reproduciéndose se ejecute, para acto seguido mostrar el siguiente bloque de animación.

6.4 Escena de subida de nivel

La subida de nivel es una pantalla simple visualmente pero que tiene una gran importancia debido a dos factores principales: En esta escena se realiza el avance de los personajes, actualizando sus valores y permitiéndoles adquirir habilidades, lo que ayuda a la progresión del juego y da control al usuario mediante un cierto grado de personalización. En segundo lugar, esta pantalla realiza la función de guardado del juego, permitiendo a este volver al punto donde lo dejó cuando jugó la última vez.

Figura 6.6 Subida de nivel con habilidades

6.5 Escena de partida finalizada

Cuando el jugador no consigue completar los objetivos propuestos para el nivel, es derrotado. Cuando esto sucede, la escena de combate genera una escena de final de partida o game over, que viene acompañada de una música acorde a la situación. Cualquier interacción con la pantalla devuelve al usuario al menú principal, donde puede volver a intentarlo.

Figura 6.7 Fin de partida

6.6 Escena de créditos

Si el jugador supera todos los niveles propuestos, la historia se termina. En ese momento, una serie de textos nombrando a todas las personas involucradas en la creación del proyecto aparece en la pantalla, ascendiendo desde la parte inferior hasta desaparecer en la zona superior. Una imagen de Fin muestra que el juego ha sido completado, y cualquier interacción tras este punto con la pantalla devolverá al usuario al menú inicial.

Figura 6.8 Créditos

6.7 Estimación de costes

En este apartado se va a proceder a la evaluación de los costes derivados de la adquisición de los distintos recursos necesarios para crear el juego, además de una estimación del coste por tiempo invertido en el desarrollo del proyecto.

6.7.1 Costes de recursos

Una de las ventajas de haber trabajado utilizando la herramienta de desarrollo de Unity es que esta posee una tienda propia de recursos, donde los distintos creadores pueden ofrecer imágenes, música, escenas ya completas y mucho más. La siguiente lista desglosa los distintos elementos comprados y su coste:

- Interfaces: El coste total del diseño de la interfaz de combate, diálogos y menú ha sido de 35 euros, adaptando las interfaces y su escalabilidad a la mayor cantidad de resoluciones posibles, y han sido obtenidos de la tienda de recursos de Unity.
- Arte genérico: Los fondos de pantalla usados en diálogos han sido obtenidos también de la ya mencionada tienda por un valor de 20 euros.
- Personajes, animaciones y casillas: Estos elementos han sido creados específicamente para el juego, con lo que no han sido obtenidos mediante la tienda de Unity y han sido creados en consonancia con una artista que los ha diseñado, con un coste total de 650 euros.
- Música: La música ha sido obtenida con una licencia free to use de la tienda de Unity, con un coste completamente gratuito bajo la condición de la aparición del nombre del creador en el producto final, figurando este en los créditos.

6.7.1 Coste de desarrollo

Es complejo estimar el precio por hora de desarrollo cuando se trata de un proyecto individual sin un sueldo ni un horario de oficina regular. No obstante, es posible hacerse una idea razonable del precio por hora de trabajo de media en el sector mirando el

sueldo medio que se oferta en webs de empleo en el campo del desarrollo software. Según Indeed, un motor de búsqueda de empleo, y con actualización del 23 de Julio de 2019 en el momento de realizar este documento, el sueldo medio por hora es de 11, 60 euros la hora[8]. Esto entra en consonancia con otras webs de empleo, como InfoJobs, que estima la hora de trabajo en 11, 90 euros, con una media de sueldo anual de más de 23.000 euros[9].

Con estas premisas, y con una estimación aproximada de 4 meses de trabajo, trabajando 8 horas diarias 5 días a la semana, se obtiene un coste total de tiempo de trabajo de 7665 euros. Esto sumado al coste total de los recursos mencionados en el anterior punto, fija el coste aproximado de diseño y desarrollo del proyecto en 8370 euros.

[8] https://www.indeed.es/salaries/Desarrollador/a-de-software-Salaries?period=hourly
[9] https://orientacion-laboral.infojobs.net/guia-trabajar-sector-it

Capítulo 7

Distribución y comercialización del producto

En este capítulo se realiza la parte final del ciclo de vida del proyecto, dividido en dos partes: análisis del producto obtenido y elección de la plataforma de distribución. Asimismo se describe el análisis previo de creación del producto comparándolo con otros juegos del género para las plataformas en las que se ha desarrollado.

7.1 Análisis de mercado previo

El mercado de dispositivos móviles es un mercado extremadamente saturado y rápido, con juegos cuya vida media es tremendamente volátil dado que pueden perder a la mayor parte de los jugadores en un periodo de tiempo muy corto. Por ello un videojuego debe ser diferente en algún aspecto a otros con los que ha de competir en el mercado. Existen varias formas de conseguir esto: poseer mecánicas muy distintas a las del resto de competidores, que su arte destaque sobre el de los demás o que la adquisición del mismo tenga un fácil acceso (en muchos juegos hay acceso gratuito y el beneficio se genera mediante el gasto de dinero en el juego o, en el caso de muchos desarrolladores indies, a través de donaciones efectuadas por los jugadores satisfechos con el juego).

Los RPG tácticos, como se ha mencionado anteriormente, tienen pocos competidores en plataformas móviles, pues la mayoría de estos son adaptaciones a estas plataformas, por lo que, en general, muchos usuarios no sienten un gran interés por jugar a juegos que ya estaban disponibles en otras plataformas. De ahí que el lanzamiento de un nuevo juego más enfocado a plataformas móviles tenga más opciones de captar la atención del jugador.

En la actualidad el perfil del jugador medio es el de un individuo que ronda los 33 años y dedica al juego entre 4 y 5 horas. En la inmensa mayoría de los casos de niños y

jóvenes con interés en los videojuegos, los padres demuestran interés por compartir este tipo de ocio con ellos. Con esta perspectiva de un público con una edad media relativamente joven que alberga la posibilidad de tener hijos, la propuesta de un juego para todas las edades amplía enormemente el espectro de opciones a la hora de captar potenciales jugadores. [ESA19]

En coherencia con todo lo anteriormente mencionado, en la fase inicial de concepto se decide que el arte del juego sea el punto fuerte sobre el que se sustente la venta del mismo, lo que conlleva que se descarten diseños más adultos de los personajes en favor de una perspectiva más infantil y de colores más llamativos, lo que permitirá atraer al público que se ha definido para el producto (jugadores jóvenes comprendidos entre los 9 y los 16 años), algo que a su vez puede atraer a familiares que muestren interés en compartir tiempo de ocio con ellos.

7.1.1 Comparativa con principales competidores

Una de las partes vitales del proceso de comercialización es la comparativa con otros videojuegos. Estudiar cuáles son las virtudes y los fallos de juegos similares en concepto, permite tener una idea más acertada de cuál es el nicho de mercado posible para el producto. A continuación se detallan algunos de los RPG tácticos más destacados para Android, iOS y PC.

- Fire Emblem Heroes. Este juego, realizado por Nintendo, es uno de los juegos especializados en plataformas móviles más importantes del sector, con unos beneficios generados en los dos últimos años de más de 500 millones de dólares americanos. El principal atractivo de este juego es su pertenencia a una franquicia que lleva siendo exitosa desde hace más de 20 años, con un sistema de juego simple pero efectivo. Los mayores problemas que este juego plantea son: un sistema de obtención de unidades basado en el azar, en el que se paga para poder aumentar las probabilidades de éxito, y una historia narrativamente pobre, que se hace repetitiva conforme se avanza en ella.
- Final Fantasy Tactics. Se trata de un juego producido por Square Enix, originalmente creado para PlayStation1 y que ha experimentado múltiples adaptaciones a lo largo del tiempo. Este juego es un buen ejemplo de cómo no analizar

y adaptar correctamente un videojuego, ya que, pese a su genial sistema de combate y una gran narrativa, carece de unos controles adecuados a las plataformas móviles.

Estos ejemplos demuestran la importancia de mantener un equilibrio entre la simplicidad de la interfaz en plataformas móviles y la calidad del producto que se vende.

7.2 Elección de la plataforma de distribución

Tal y como se ha mencionado anteriormente en el estado del arte, existen múltiples opciones a la hora de distribuir un juego, aunque muchas de ellas tengan verdaderos inconvenientes. La mayoría de ellos están relacionados con el hecho de que las tiendas distribuidoras estén asociadas con un único tipo de aplicación: por ejemplo, Google Play solo permite la distribución de aplicaciones para Android, del mismo modo que Apple hace lo propio para su entorno. La mayoría de los sitios web que permiten la subida de videojuegos tienen un sistema de evaluación por parte de los propios usuarios, lo cual restringe y limita la cantidad de opciones que tiene un pequeño desarrollador para ganar visibilidad.

El aumento exponencial de la cantidad de juegos indie en el mercado, surgido en los últimos diez años, ha creado la necesidad en los pequeños desarrolladores de ver cómo su producto puede ser lanzado en múltiples plataformas en un sitio web unificado. Es aquí donde entra en juego itch.io, una plataforma web creada en 2013, que para febrero de 2018 contenía 100.000 juegos y artículos de carácter indie. [10]

[10]https://es.wikipedia.org/wiki/Itch.io

7.2.1 Proceso de distribución en itch.io

El proceso de distribución de una aplicación en itch.io es extremadamente sencillo. El usuario no tiene nada más que crear una cuenta en la web y subir sus ejecutables en un nuevo proyecto en la misma, tal y cómo se muestra en la siguiente figura.

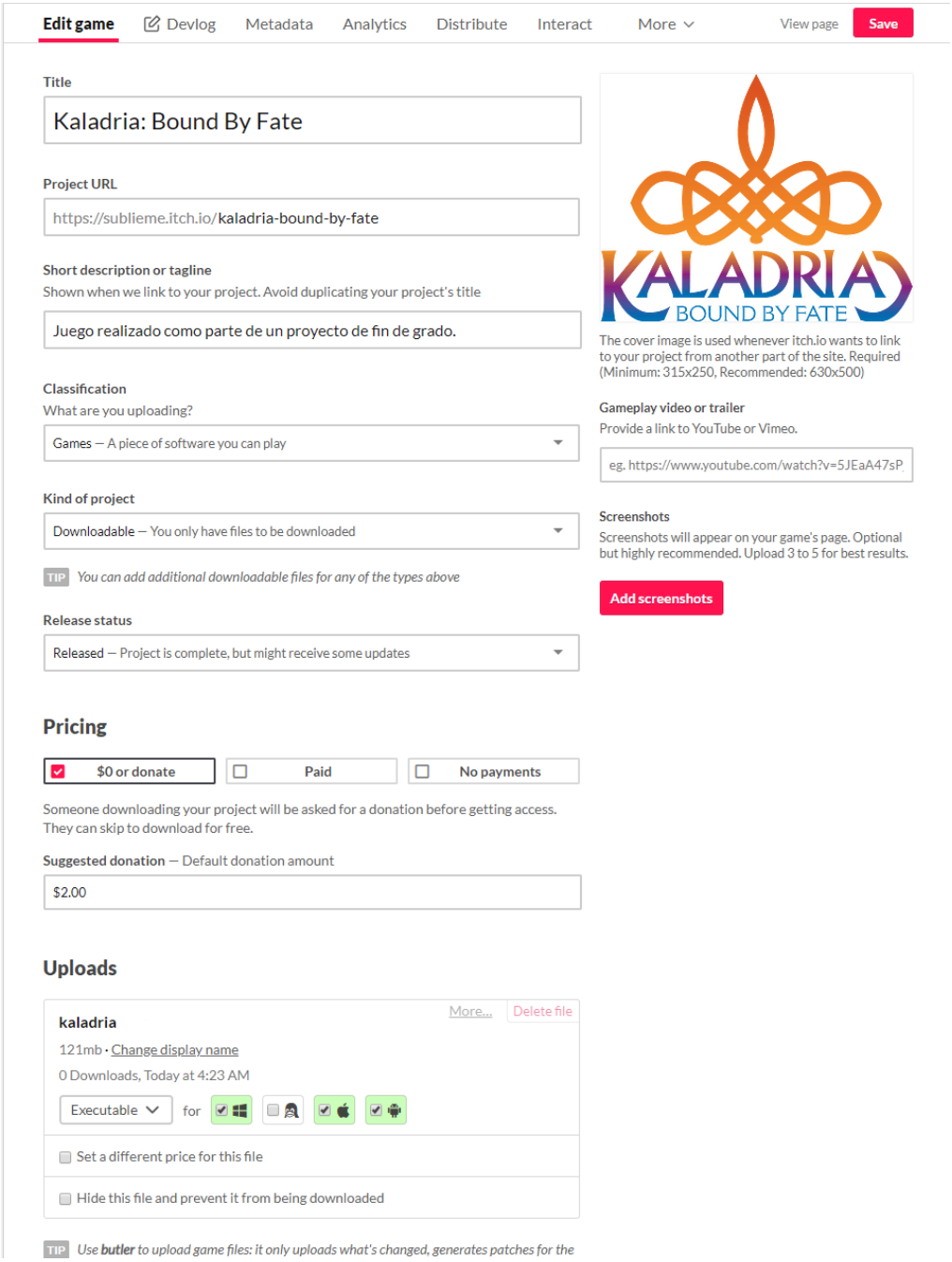


Figura 7.1: Web de publicación del videojuego

Es importante indicar la flexibilidad de itch.io a la hora de recibir dinero: permite que el juego sea gratuito y que los beneficios sean directamente por donaciones, y también pagar a cambio del producto. No solo esto, sino que también posibilita indicar el estado de este: como finalizado, de acceso temprano, fase beta, finalización de soporte del producto, etc. Asimismo, itch.io permite personalizar la página del videojuego de la forma que prefiera el desarrollador, incluyendo la opción (si así lo desea este) de subir su propio CSS a la web y aumentar el tamaño máximo de subida tras una breve charla con los administradores de la página web.

Por estas razones, itch.io se desmarca del resto de sus competidores en el mercado como una opción rápida y fiable, siendo por ello la opción elegida para este proyecto.

Capítulo 8

Conclusiones

Para finalizar este documento se realiza un análisis de los resultados obtenidos tras el proceso de diseño, desarrollo y comercialización del proyecto, observando con atención en qué grado se han cumplido los distintos objetivos fijados al comienzo de este.

8.1 Cumplimiento de los objetivos

8.1.1 Objetivo principal

El objetivo inicial de este proyecto era la creación desde cero de un proyecto multimedia multiplataforma: un videojuego. Si bien no ha estado exento de problemas durante el desarrollo, algo que ya asumía la metodología de trabajo, se ha conseguido un ejecutable final capaz de funcionar en múltiples plataformas, con un total de 10 niveles jugables, 5 personajes disponibles y 11 escenas de diálogo. Como juego indie realizado por un solo desarrollador, el producto final cumple las especificaciones pedidas y solo queda ver si el mercado responde favorablemente ante el producto o si este se hundirá en un océano de juegos indies.

8.1.2 Objetivos específicos

A continuación se procede a evaluar cada uno de los puntos que componen los objetivos secundarios mencionados en el capítulo 2, precisando de qué forma estos se han cumplido.

Diseño del sistema multiplataforma:

- El sistema conseguido es multiplataforma, habiéndose elegido Unity como entorno de desarrollo y motor gráfico, permitiendo la creación del sistema multiplataformas.
- Todos los algoritmos creados para el juego han demostrado ser compatibles con 3 sistemas operativos (Windows, iOS, Android) y el código está implementado de forma que puede expandirse a otros sistemas.
- Las pantallas utilizan un sistema dinámico de overlay que se ajusta en la medida de todo lo posible a la inmensa cantidad de resoluciones que existen en el mercado. Cabe destacar que no se han encontrado casos en los que no se pudiera hacer funcionar el juego, si bien en algunos dispositivos la interfaz tenía algún fallo menor.
- Las decisiones de controles se programaron decidiendo que el control táctil era una base excelente que permitía adaptar el comportamiento a sistemas con ratón sin apenas carga.
- El código relacionado con la gestión de ficheros procesa y guarda estos con la codificación correcta, y da formato a los diversos archivos, generando los objetos correspondientes.

Diseño de niveles:

- El sistema está diseñado para ser capaz de aguantar con un rendimiento óptimo múltiples unidades nuevas y mapas más complejos, ya que estos son generados mediante un sistema de ficheros y una carga de los recursos asociados.
- El mapa presenta una interfaz clara que en ningún momento se ve afectada por el menú y los datos necesarios para funcionar, al estar el mapa comprendido entre los dos bloques de interfaz existentes.
- Al usarse ficheros y recursos que están compuestos por sprites, con pesos ínfimos, el sistema permite la expansión y la creación de múltiples mapas sin requerir apenas recursos, exigiendo solo al desarrollador el acceso a nuevas casillas para añadir variedad.

Desarrollo de los combates:

- La estrategia en combate se ha decidido mediante un sistema de habilidades y colocación cuidadosa de las unidades del mapa. En este punto se podrían realizar mejoras ampliando la variedad de habilidades y añadiendo más mapas con más detalle.
- Existen animaciones visuales con sonidos que muestran el resultado de las acciones realizadas.
- El cálculo de los combates se realiza unidad por unidad, haciendo que las unidades enemigas se muevan de forma independiente al cálculo iterativo mediante corrutinas, generando un sistema eficiente y funcional.
- Los tres patrones de combate que posee la IA enemiga permiten al usuario identificar patrones que se repiten a lo largo del mapa y que permite aprender comportamientos incluso en caso de ser derrotado.
- Las ayudas visuales de iconos, junto a descripciones cuando se consiguen las habilidades permiten que los jugadores puedan tener una idea intuitiva de los conceptos básicos del combate, si bien algún sistema de descripción se podría implementar como ayuda adicional.
- Al comienzo de todos los combates se especifica con claridad las condiciones de derrota y victoria.

Distribución y comercialización del producto:

Con la versión final producida para distintas plataformas, se procederá a la distribución de estas en una web comercial tras haber llevado a cabo un análisis previo de los pros y contras de las distintas plataformas disponibles. Para que esto sea posible se tendrá que:

- Se ha efectuado un análisis de las opciones de venta del producto, decidiendo un público objetivo y orientando el estilo visual del juego de acuerdo a las premisas definidas.
- Un análisis general ha encontrado ciertos problemas en los competidores, pero debido al desarrollo individual del proyecto, es extremadamente difícil conseguir aplicar estos conocimientos, al ser los competidores juegos de gran calado con un equipo masivo detrás.

- Se han evaluado las plataformas de distribución más importantes y se ha decidido conforme a sus políticas de distribución que itch.io era la opción más adecuada al proyecto.
- Se continúa evaluando el feedback y los comentarios, si bien en el momento de este escrito no existen suficientes datos para generar un análisis sólido al respecto.

8.2 Justificación de las competencias adquiridas

Competencias	Justificaciones
Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas.	A lo largo de todas las decisiones de diseño del juego, especialmente en la decisión de dónde colocar botones, diálogos y otros elementos de la interfaz, aplicando una filosofía WYSWYG, haciendo que la experiencia del jugador sea fluida e intuitiva.
Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil.	Se ha diseñado un sistema multimedia interactivo multiplataforma, incluyendo plataformas móviles. Este proyecto se encuentra claramente situado dentro del marco de las tecnologías de la información.

Anexo A

Listado de código referenciado

En este anexo se localizan las referencias al código mencionado en el apartado de arquitectura y que sirven de guía visual para el texto asociado.

```
   public void DisplayNextSentence() {
        if (_sentences.Count > 0) {
            TextMeshProUGUI dialogText =
GameObject.Find("DialogText").GetComponent<TextMeshProUGUI>();
            string sentence = _sentences.Dequeue();
            StopAllCoroutines();
            StartCoroutine(TypeSentence(sentence, dialogText));
        } else {
            if (_dialogCounter < _dialogs.Count) {
                EnqueueDialog(_dialogs[_dialogCounter]);
                Image avatar;
                TextMeshProUGUI name;
                if (_dialogCounter % 2 == 0) {
                    _avatarBoxLeft.SetActive(true);
                    _nameBoxLeft.SetActive(true);
                    _avatarBoxRight.SetActive(false);
                    _nameBoxRight.SetActive(false);
                    avatar =
GameObject.Find("AvatarLeft").GetComponent<Image>();
                    name =
GameObject.Find("NameTextLeft").GetComponent<TextMeshProUGUI>();
                } else {
                    _avatarBoxLeft.SetActive(false);
                    _nameBoxLeft.SetActive(false);
                    _avatarBoxRight.SetActive(true);
                    _nameBoxRight.SetActive(true);
                    avatar =
GameObject.Find("AvatarRight").GetComponent<Image>();
                    name =
GameObject.Find("NameTextRight").GetComponent<TextMeshProUGUI>();
                }
                ChargeImage(avatar, _dialogs[_dialogCounter].Name);

                name.text = _dialogs[_dialogCounter].Name;
                _dialogCounter++;
                DisplayNextSentence();
            } else {
                SceneManager.LoadScene("BattleScene");
            }
        }
    }
```

```
    private IEnumerator ChargeImage(Image image, string name) {
        string path = path = Application.streamingAssetsPath + "/" +
name + ".png";
        byte[] imgData;
        Texture2D tex = new Texture2D(2, 2);


        if (path.Contains("://") || path.Contains(":///")) {
            UnityWebRequest www = UnityWebRequest.Get(path);
            yield return www.SendWebRequest();
            imgData = www.downloadHandler.data;
        } else {
            imgData = System.IO.File.ReadAllBytes(path);
        }


        tex.LoadImage(imgData);


        Vector2 pivot = new Vector2(0.5f, 0.5f);
        Sprite sprite = Sprite.Create(tex, new Rect(0.0f, 0.0f,
tex.width, tex.height), pivot, 100.0f);

        image.sprite = sprite;

    }
```

```
    public IEnumerator SetupBattleAnimation( BattlesOnTurn battles, int i) {
        if (i < battles.ONames.Count) {
            if(battles.NewTHealthbars[i] < 0) {
                battles.NewTHealthbars[i] = 0;
            }
            if (battles.NewOHealthbars[i] < 0) {
                battles.NewOHealthbars[i] = 0;
            }
            _checkEnd = false;
            //foreach (AnimationState anim in oUnitAnim) { }
            HealthBarAnimation oHealthBarScript =
GameObject.Find("OriginHealthBar").GetComponent<HealthBarAnimation>();

oHealthBarScript.gameObject.transform.Find("Bar").gameObject.transform.localScale
= new Vector2(battles.CurrentOHealthbars[i], 1f);
            oHealthBarScript.ActualHealth = battles.CurrentOHealthbars[i];
            oHealthBarScript.TargetHealth = battles.NewOHealthbars[i];

            HealthBarAnimation tHealthBarScript =
GameObject.Find("TargetHealthBar").GetComponent<HealthBarAnimation>();

tHealthBarScript.gameObject.transform.Find("Bar").gameObject.transform.localScale
= new Vector2(battles.CurrentTHealthbars[i], 1f);
            tHealthBarScript.ActualHealth = battles.CurrentTHealthbars[i];
            tHealthBarScript.TargetHealth = battles.NewTHealthbars[i];

            oHealthBarScript.InvokeRepeating("UpdateHealth", 0, 0.1f);
            tHealthBarScript.InvokeRepeating("UpdateHealth", 0, 0.1f);



GameObject.Find("OriginUnitAnim").GetComponent<ImageAnimation>().LoadAnimation(ba
ttles.ONames[i] + "Attack");

GameObject.Find("TargetUnitAnim").GetComponent<ImageAnimation>().LoadAnimation(ba
ttles.TNames[i] + "Damage");
            GameObject.Find("TargetUnitAnim").GetComponent<Animator>().speed =
0f;
            yield return new WaitForSeconds(0.8f);
            GameObject.Find("TargetUnitAnim").GetComponent<Animator>().speed =
1f;
            yield return new WaitForSeconds(0.3f);
            StartCoroutine(SetupBattleAnimation(battles, i+1));
        } else{
            _checkEnd = true;
        }

    }
```

```
    public void Update() {
        if (!Screen.fullScreen) {
            Screen.fullScreen = true;
        }
        if (animationPlaying) {
            if (currentNumber != desiredNumber) {
                if (initialNumber < desiredNumber) {
                    currentNumber += (2.0f * Time.deltaTime) *
(desiredNumber - initialNumber);
                    if (currentNumber >= desiredNumber) {
                        currentNumber = desiredNumber;
                    }
                }
            }

            if (nextStat < 6) {
                unitTexts[nextStat].text =
unitTexts[nextStat].text.Split(':')[0] + ": ";
                unitTexts[nextStat].text = unitTexts[nextStat].text +
currentNumber.ToString("0");
                if (currentNumber == desiredNumber) {
                    nextStat += 1;
                    switch (nextStat) {
                        case 1:
                            initialNumber = units[nextUnit].Hp;
                            currentNumber = initialNumber;
                            desiredNumber = updatedUnit.Hp;
                            break;
                        case 2:
                            initialNumber = units[nextUnit].Atk;
                            currentNumber = initialNumber;
                            desiredNumber = updatedUnit.Atk;
                            break;
                        case 3:
                            initialNumber = units[nextUnit].Def;
                            currentNumber = initialNumber;
                            desiredNumber = updatedUnit.Def;
                            break;
                        case 4:
                            initialNumber = units[nextUnit].Res;
                            currentNumber = initialNumber;
                            desiredNumber = updatedUnit.Res;
                            break;
                        case 5:
                            initialNumber = units[nextUnit].Spd;
                            currentNumber = initialNumber;
                            desiredNumber = updatedUnit.Spd;
                            break;

                    }
                }
            } else {
                animationPlaying = false;
                nextStat = 1;
            }
        }

    }
```

```
    private TargetRoute AttackRoute(Unit unit, int mov, TargetRoute targetRoute,
List<Coordinate> route) {



        if (mov < 0 || !IsCoordValid(route)) {
            return targetRoute;
        }

        Unit tempTarget = CalculateMaxPriorityTarget(unit, new Unit(),
route[route.Count - 1]);
        if (!tempTarget.Name.Equals("mock")) {
            if (targetRoute.Target.Equals(tempTarget)) {
                if (targetRoute.Route.Count > route.Count) {
                    targetRoute.Route = new List<Coordinate>();
                    foreach (Coordinate routeCoord in route) {
                        targetRoute.Route.Add(routeCoord);
                    }
                }
            } else {
                tempTarget = targetRoute.Target;
                targetRoute.Target = CalculateMaxPriorityTarget(unit,
targetRoute.Target, route[route.Count - 1]);

                if (!targetRoute.Target.Equals(tempTarget)) {
                    targetRoute.Route = new List<Coordinate>();
                    foreach (Coordinate routeCoord in route) {
                        targetRoute.Route.Add(routeCoord);
                    }
                }
            }
        }


        if (targetRoute.Route.Count == 0) {
            targetRoute.Route.Add(route[0]);
        } else if (targetRoute.Route[targetRoute.Route.Count - 1].Y >
route[route.Count - 1].Y) {
            targetRoute.Route = new List<Coordinate>();
            foreach (Coordinate routeCoord in route) {
                targetRoute.Route.Add(routeCoord);
            }
        }



        route.Add(new Coordinate(route[route.Count - 1].X + 1, route[route.Count
- 1].Y));
        targetRoute = AttackRoute(unit, mov - 1, targetRoute, route);
        route.RemoveAt(route.Count - 1);

        route.Add(new Coordinate(route[route.Count - 1].X, route[route.Count -
1].Y + 1));
        targetRoute = AttackRoute(unit, mov - 1, targetRoute, route);
        route.RemoveAt(route.Count - 1);

        route.Add(new Coordinate(route[route.Count - 1].X - 1, route[route.Count
- 1].Y));
        targetRoute = AttackRoute(unit, mov - 1, targetRoute, route);
        route.RemoveAt(route.Count - 1);
```

Anexo B

Listado de diagramas

El anexo contiene los diagramas de secuencia de otros escenarios posibles del juego, que quedan como material de referencia para el lector.

Diagrama de secuencia de movimiento válido:

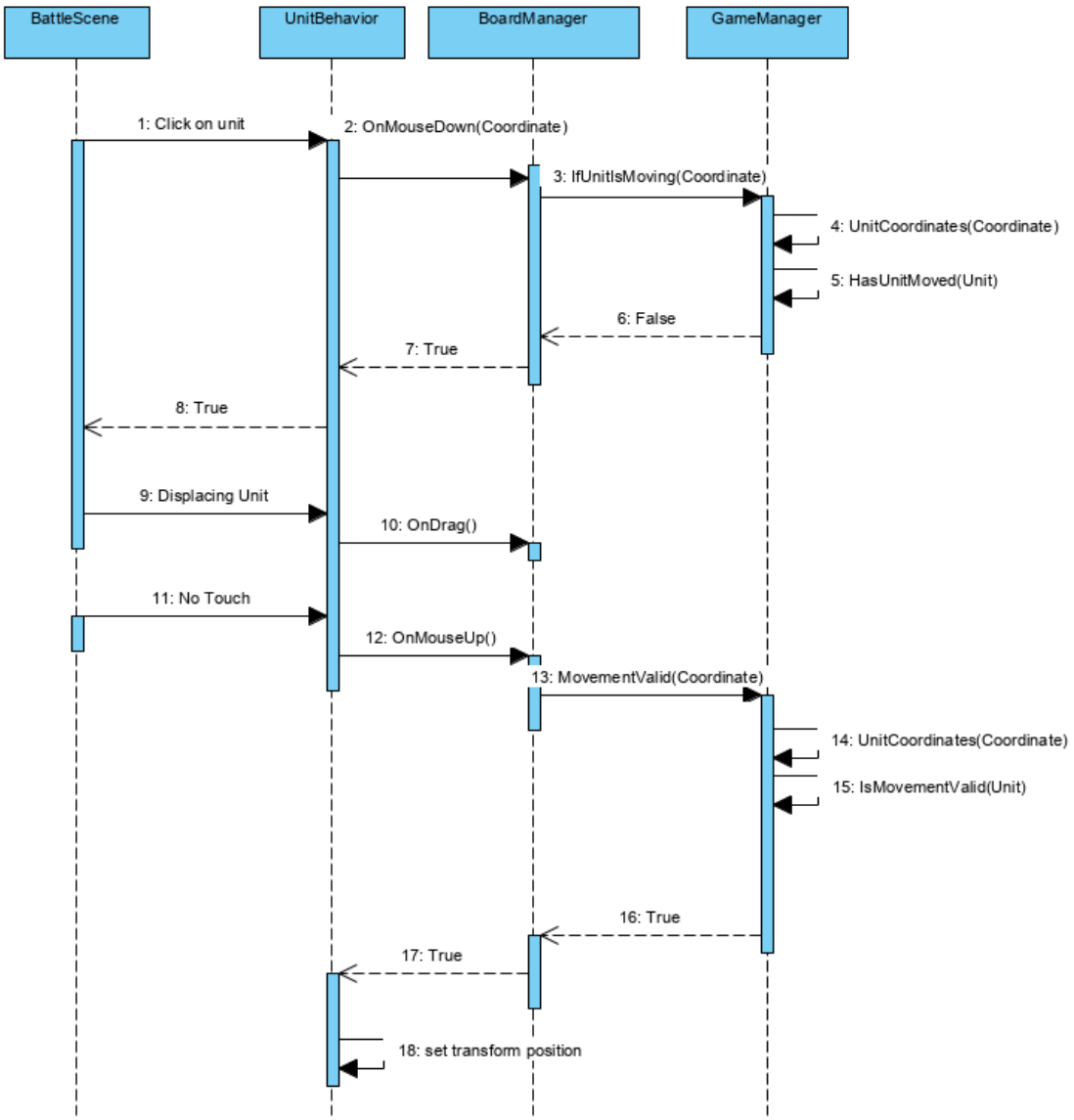
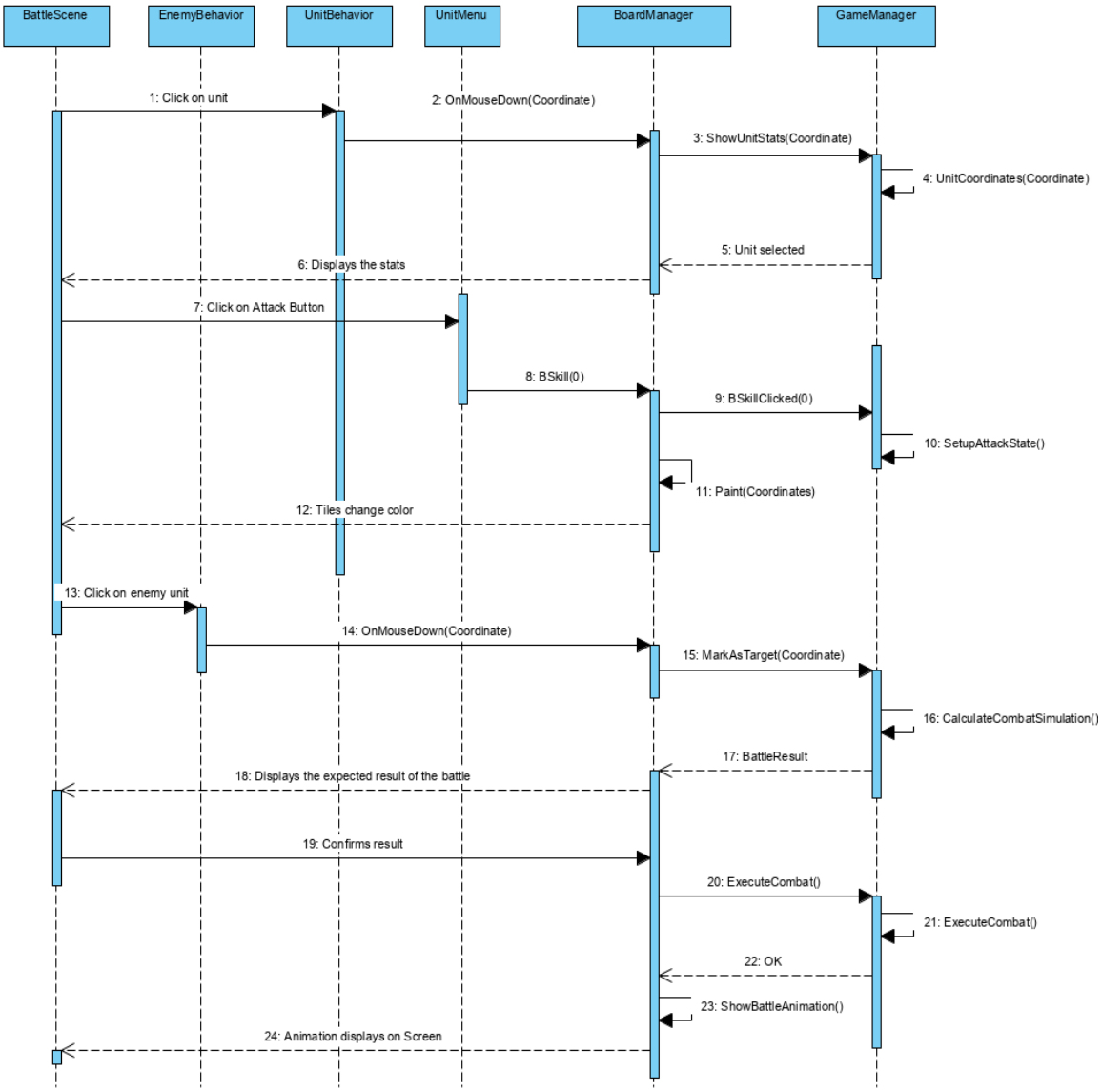
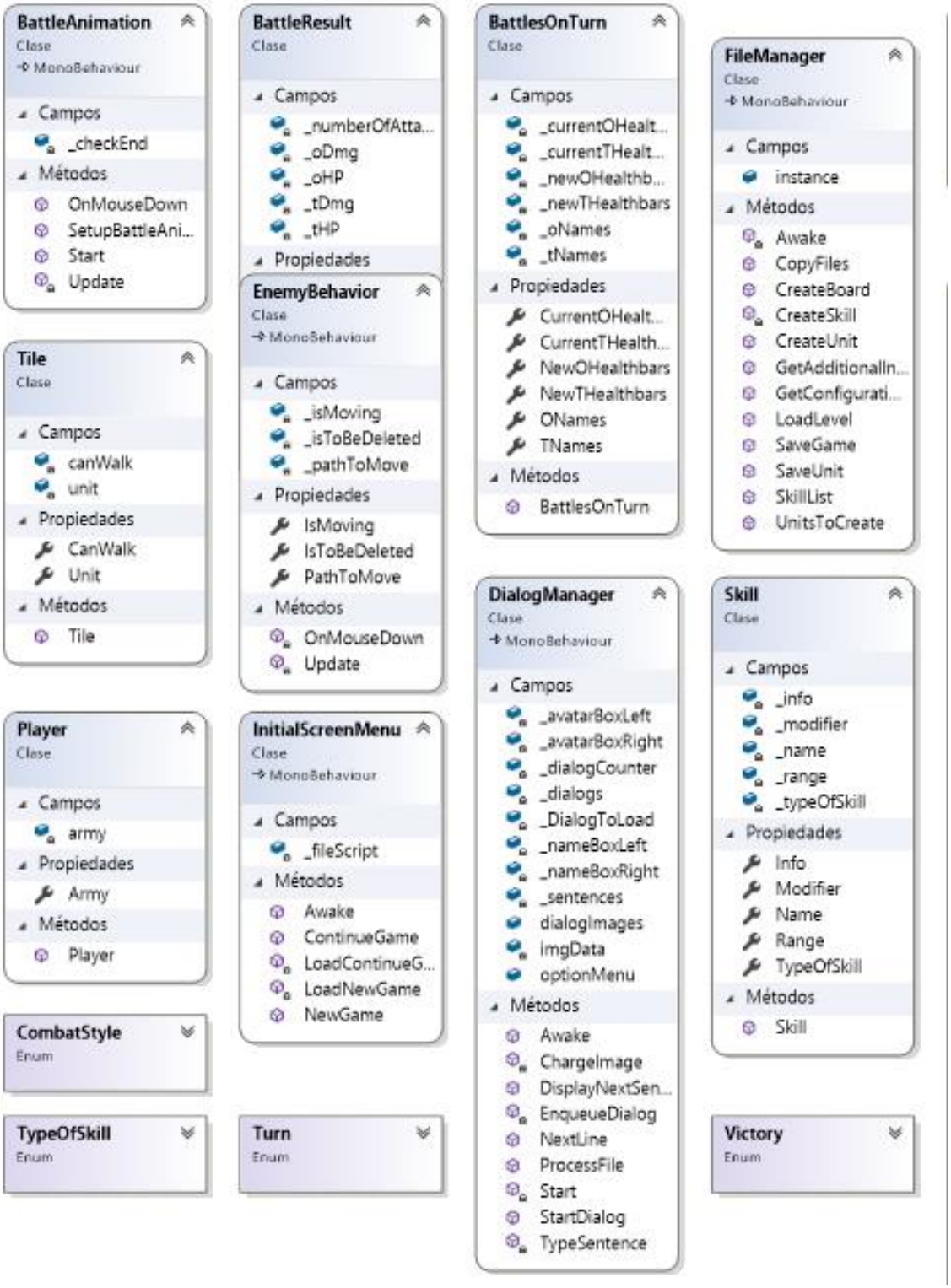
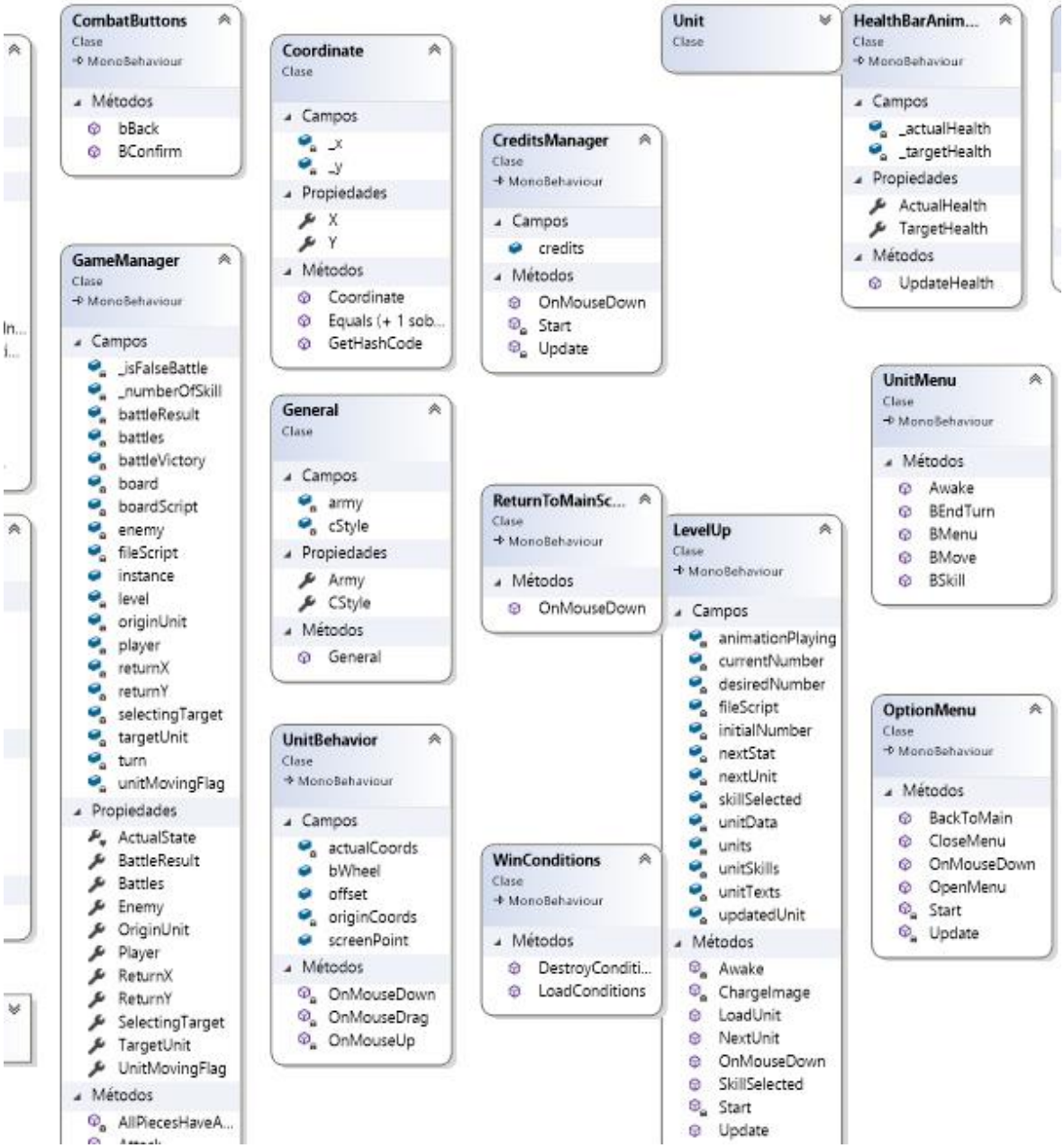


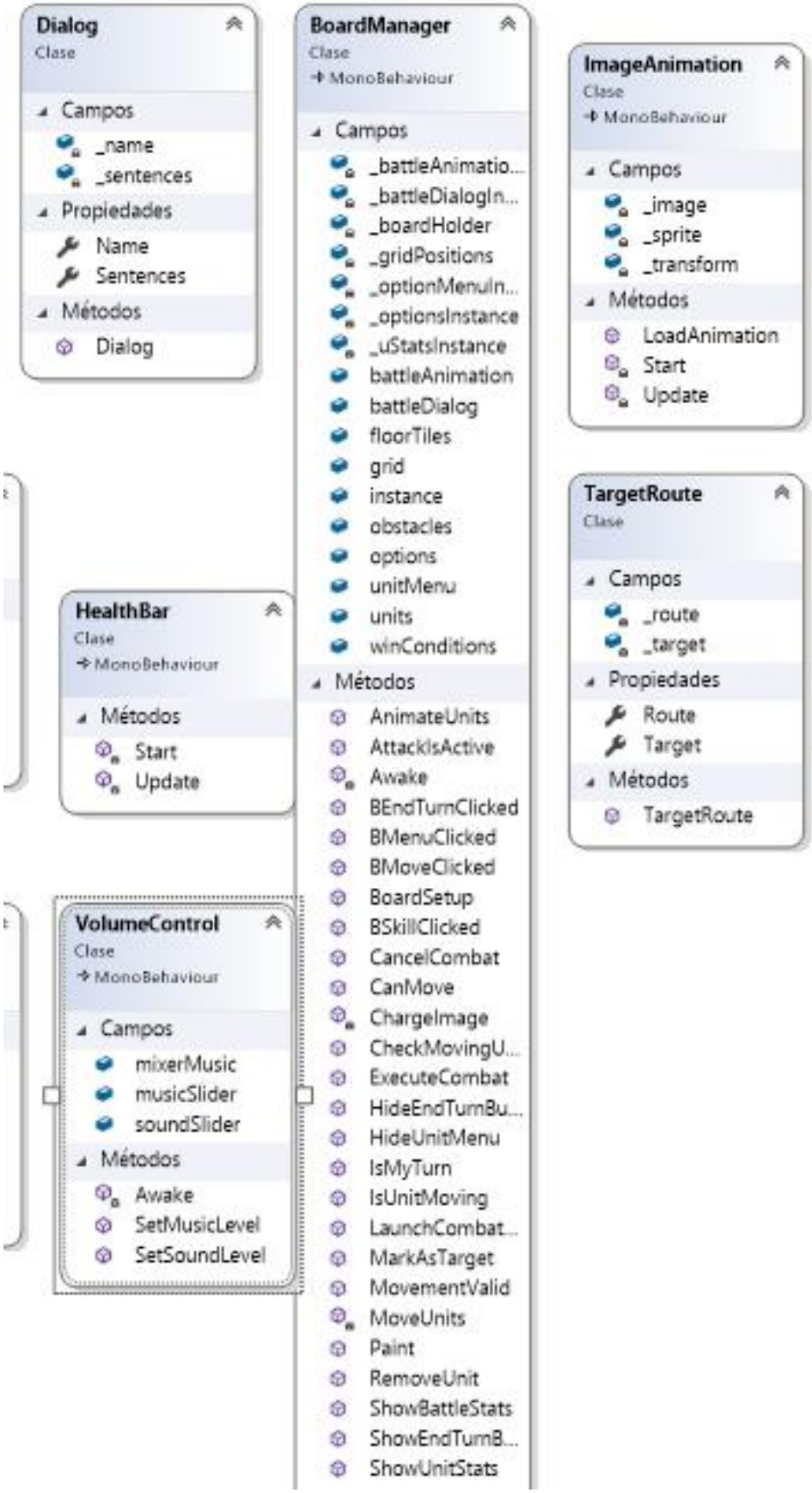
Diagrama de secuencia de ataque del jugador:



Contenido de la clases del proyecto:




CombatButtons
Clase
MonoBehaviour
Métodos
bBack
BConfirm
Coordinate
Clase
Campos
_x
_y
Propiedades
X
Y
Métodos
Coordinate
Equals (+ 1 sob...
GetHashCode
Unit
Clase
HealthBarAnim...
Clase
MonoBehaviour
Campos
_actualHealth
_targetHealth
Propiedades
ActualHealth
TargetHealth
Métodos
UpdateHealth
CreditsManager
Clase
MonoBehaviour
Campos
credits
Métodos
OnMouseDown
Start
Update
GameManager
Clase
MonoBehaviour
Campos
_isFalseBattle
_numberOfSkill
battleResult
battles
battleVictory
board
boardScript
enemy
fileScript
instance
level
originUnit
player
returnX
returnY
selectingTarget
targetUnit
turn
unitMovingFlag
Propiedades
ActualState
BattleResult
Battles
Enemy
OriginUnit
Player
ReturnX
ReturnY
SelectingTarget
TargetUnit
UnitMovingFlag
Métodos
AllPiecesHaveA...
General
Clase
Campos
army
cStyle
Propiedades
Army
CStyle
Métodos
General
UnitMenu
Clase
MonoBehaviour
Métodos
Awake
BEndTurn
BMenu
BMove
BSkill
ReturnToMainSc...
Clase
MonoBehaviour
Métodos
OnMouseDown
LevelUp
Clase
MonoBehaviour
Campos
animationPlaying
currentNumber
desiredNumber
fileScript
initialNumber
nextStat
nextUnit
skillSelected
unitData
units
unitSkills
unitTexts
updatedUnit
Métodos
Awake
ChargeImage
LoadUnit
NextUnit
OnMouseDown
SkillSelected
Start
Update
UnitBehavior
Clase
MonoBehaviour
Campos
actualCoords
bWheel
offset
originCoords
screenPoint
Métodos
OnMouseDown
OnMouseDrag
OnMouseUp
OptionMenu
Clase
MonoBehaviour
Métodos
BackToMain
CloseMenu
OnMouseDown
OpenMenu
Start
Update
WinConditions
Clase
MonoBehaviour
Métodos
DestroyConditi...
LoadConditions


Dialog
Clase
Campos
_name
_sentences
Propiedades
Name
Sentences
Métodos
Dialog
BoardManager
Clase
MonoBehaviour
Campos
_battleAnimatio...
_battleDialogIn...
_boardHolder
_gridPositions
_optionMenuIn...
_optionsInstance
_uStatsInstance
battleAnimation
battleDialog
floorTiles
grid
instance
obstacles
options
unitMenu
units
winConditions
Métodos
AnimateUnits
AttackIsActive
Awake
BEndTurnClicked
BMenuClicked
BMoveClicked
BoardSetup
BSkillClicked
CancelCombat
CanMove
ChargeImage
CheckMovingU...
ExecuteCombat
HideEndTurnBu...
HideUnitMenu
IsMyTurn
IsUnitMoving
LaunchCombat...
MarkAsTarget
MovementValid
MoveUnits
Paint
RemoveUnit
ShowBattleStats
ShowEndTurnB...
ShowUnitStats
ImageAnimation
Clase
MonoBehaviour
Campos
_image
_sprite
_transform
Métodos
LoadAnimation
Start
Update
TargetRoute
Clase
Campos
_route
_target
Propiedades
Route
Target
Métodos
TargetRoute
HealthBar
Clase
MonoBehaviour
Métodos
Start
Update
VolumeControl
Clase
MonoBehaviour
Campos
mixerMusic
musicSlider
soundSlider
Métodos
Awake
SetMusicLevel
SetSoundLevel

Referencias

[LBV18] Libro Blanco del Desarrollo Español de Videojuegos. Disponible online: http://www.dev.org.es/es/publicaciones/libro-blanco-dev-2018, Asociación Española de Empresas Productoras Y Desarrolladoras de Videojuegos y Software de Entretenimiento, 2018.

[Per16] PEREZ, Sarah, 2016. Pokémon GO passed 100 million install over the weekend. En: *TechCrunch* [en línea]. Disponible en: https://techcrunch.com/2016/08/01/pokemon-go-passed-100-million-installs-over-the-weekend/ [consulta: 16 julio 2019].

[Kuc18] KUCHERA, Ben, 2018. Valve now rewards successful games with a larger cut of Steam revenue. En: *Polygon* [en línea]. Disponible en: https://www.polygon.com/2018/12/3/18123649/valve-steam-revenue-sharing [consulta: 17 julio 2019].

[ESA19] ESA Essential Facts 2019. Disponible online: https://www.theesa.com/wp-content/uploads/2019/05/ESA_Essential_facts_2019_final.pdf, Entertainment Software Association, 2019.

[VM17] VALLEJO, David, MARTÍN, Carlos, 2017, Desarrollo de videojuegos, un enfoque práctico, volumen 1 [en línea]. Amazon: Amazon. [consulta: 26 Julio 2019]. Disponible en http://cedv.uclm.es/libro2015/M1.pdf

[GHJ+94] GAMMA, Erich, HELM, Richard, JOHNSON, Ralph y Vlissides, John, 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. USA: Addison-Wesley. ISBN 0-201-63361-2.

[Lee16] LEE, Joanna, 2016, *Learning Unreal Engine Game Development*. USA: Packt Publishing. ISBN 978-1784398156.

[Sev15] SEWELL, Brenden, 2015, *Blueprints Visual Scripting for Unreal Engine*. USA: Packt Publishing. ISBN 978-1785286018.

[Gun14] GUNDLACH, Sascha, 2014, *Mastering CryEngine*. USA: Packt Publishing. ISBN 978-1783550258.

[Jac14] JACKSON, Simon, 2014, *Mastering Unity 2D Game Development*. USA: Packt Publishing. ISBN 978-1849697347.