
**Motor y editor de videojuegos en 2D enfocado al
desarrollo de juegos RPG**

**2D video game engine and editor focused on
RPG game development**



**Trabajo de Fin de Grado
Curso 2024–2025**

Autores

Miguel Curros García
Alejandro González Sánchez
Alejandro Massó Martínez

Director

Pedro Pablo Gómez Martín

Grado en Desarrollo de Videojuegos

Facultad de Informática

Universidad Complutense de Madrid

Motor y editor de videojuegos en 2D
enfocado al desarrollo de juegos RPG
2D video game engine and editor focused
on RPG game development

Trabajo de Fin de Grado en Desarrollo de Videojuegos

Autores

Miguel Curros García
Alejandro González Sánchez
Alejandro Massó Martínez

Director

Pedro Pablo Gómez Martín

Convocatoria: *Junio 2025*

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

26 de mayo de 2025

Dedicatoria

Dedicatoria de Miguel. - Miguel

Dedicatoria de Alex. - Alex G.

Dedicatoria de Alex. - Alex M.

Agradecimientos

A Guillermo, por el tiempo empleado en hacer estas plantillas. A Adrián, Enrique y Nacho, por sus comentarios para mejorar lo que hicimos. Y a Narciso, a quien no le ha hecho falta el Anillo Único para coordinarnos a todos.

Resumen

Motor y editor de videojuegos en 2D enfocado al desarrollo de juegos RPG

Pese a que los videojuegos de rol (RPG) son uno de los géneros más demandados actualmente en la industria, hay una gran escasez de motores y editores específicos para este tipo de juegos. Los motores más utilizados tienden a ser muy genéricos y no están centrados específicamente en los RPG.

A este problema, se le añade que los motores y editores para RPG de código abierto disponibles no suelen ser muy amigables con los nuevos usuarios, ya que suelen utilizar mecanismos y estructuras pensados para gente ducha en la materia. Los que sí que tienen una interfaz más sencilla y más práctica suelen estar bloqueados bajo un muro de pago, por lo que muchas veces, diseñadores aficionados se ven gastando el precio de un juego en la propia licencia de un *software* que van a utilizar en contadas ocasiones.

Para dar respuesta a estos problemas, en este trabajo se presenta el desarrollo de un motor y editor de videojuegos 2D orientado específicamente a los RPG, con soporte multiplataforma, cuyo objetivo es facilitar la creación de este tipo de juegos a usuarios sin experiencia en el desarrollo o la programación, manteniendo también algunos elementos más complejos para que usuarios más experimentados puedan generar juegos más completos, todo ello utilizando herramientas de libre acceso.

Palabras clave

Videojuegos de Rol, Desarrollo de Videojuegos, Editor de Videojuegos, Herramienta de Desarrollo, Motor de Videojuegos.

Abstract

2D video game engine and editor focused on RPG game development

Although role-playing video games (RPGs) are among the most demanded genres in today's industry, specific engines and editors designed for these type of games are scarce. The most used ones tend to be too generic and are not focused on RPGs specifically.

In addition to this, the available open source RPG engines and editors are often not very user-friendly for newcomers, as they use mechanisms and structures designed for users with advanced knowledge. Those that do offer a more straightforward and practical interface are often locked behind a paywall, so amateur designers find themselves spending the price of a game on the license for a software they will rarely use.

To address these problems, this project presents the development of a 2D-video game engine and editor specifically oriented towards RPGs, with cross-platform support, whose objective is to ease the creation of this type of games for users without experience in development or programming, while maintaining more advanced features so that experienced users can generate more complete games, all using open-source and freely accessible tools.

Keywords

Role-playing Video Games, Video Game Development, Video Game Editor, Development Tool, Video Game Engine.

Índice

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Plan de trabajo	2
1.4. Herramientas y Metodología	3
1.5. Estructura de la memoria	4
2. Juegos de Rol	5
2.1. ¿Qué es un juego de rol?	5
2.1.1. Tipos de juegos de rol	6
2.1.1.1. Juegos de rol de mesa o tablero	6
2.1.1.2. Juegos de rol en vivo	7
2.1.1.3. Juegos de rol por foro o texto	8
2.1.1.4. Juegos de rol digitales	8
2.1.1.5. Comparativa	8
2.2. Historia de los juegos de rol	8
2.3. Análisis de los principales juegos de rol	10
2.3.1. <i>Dungeons & Dragons</i>	10
2.3.2. <i>Pathfinder</i>	11
2.3.3. <i>Call of Cthulhu</i>	12
2.3.4. <i>Warhammer Fantasy Roleplay</i>	12
3. Videojuegos de Rol	15
3.1. ¿Qué es un videojuego?	15
3.1.1. El problema de los géneros de videojuegos	15
3.1.2. Historia de los videojuegos de rol	15
3.2. Sobre el desarrollo de videojuegos	15
3.2.1. Motor de videojuegos	15
3.2.1.1. Componentes de un motor de videojuegos	15
3.2.1.2. Separación entre motor y <i>gameplay</i>	15
3.2.1.3. Programación dirigida por datos (DDP) en videojuegos	15
3.2.1.4. Programación multiplataforma en videojuegos	15
3.2.2. Editor de videojuegos	15

3.2.2.1. Editores específicos de videojuegos para desarrollo de RPG	15
4. Planteamiento del Proyecto	17
4.1. Objetivos principales del Proyecto	17
4.2. Toma de decisiones	18
4.2.1. Diseño del motor	19
4.2.1.1. Base con el sistema entidad-componente	19
4.2.1.2. Componentes genéricos de juego	19
4.2.1.3. Componentes específicos de RPG	20
4.2.1.4. Sistema de eventos	21
4.2.2. Diseño del editor	22
4.2.2.1. Funcionalidad del editor	22
4.2.2.2. Modularidad y arquitectura	23
4.2.2.3. Tecnologías utilizadas	24
5. Desarrollo del Proyecto	25
5.1. Puesta en marcha del <i>toolchain</i>	25
5.2. Desarrollo del motor	27
5.3. Desarrollo del editor	27
5.3.0.1. Estructura del código y organización de módulos	27
6. Evaluación y Conclusiones	29
6.1. Objetivo de la evaluación	29
6.2. Metodología	29
6.3. Resultados	29
6.4. Análisis de los resultados y conclusiones	29
7. Trabajo Futuro	31
Introduction	33
Conclusions and Future Work	37
8. Contribuciones Personales	39
Contribuciones Personales	39
Bibliografía	45

Índice de figuras

1.1.	Diagrama de Gantt mostrando la planificación temporal del trabajo.	3
2.1.	Partida de LARP nórdico, extraída de Pohjola (2021).	7
2.2.	Tablero de juego moderno de <i>Kriegsspiel</i> (von Reisswitz y von Reisswitz, 1812), extraída de Heinemann (2022).	9
2.3.	Imágenes de distintas variedades de partidas de <i>Dungeons & Dragons</i>	11
2.4.	Partida de <i>Warhammer Fantasy Battle</i> , extraída de Paddington (2023).	13
4.1.	Editor de mapas de <i>RPGBaker</i>	22
5.1.	Estructura de carpetas del Proyecto.	26
7.1.	Gantt diagram showing the temporal planning of the work.	34

Capítulo 1

Introducción

“Una vez tuve una conversación bastante rara con un par de abogados y estaban hablando sobre: «¿Cómo elegís a vuestro público objetivo? ¿Hacéis "focus groups", encuestáis a gente y todo eso?» Y es como: «No, simplemente hacemos juegos que creemos que molan.»”
— John Carmack

1.1. Motivación

Los videojuegos de rol han sido uno de los géneros más influyentes de la industria, desde sus orígenes en la década de los años 80 hasta la actualidad, donde concentran una gran parte de la cuota de mercado.

El desarrollo de este tipo de juegos ha sufrido cambios mayúsculos con el paso de los años y con las consecuentes mejoras *hardware* y *software*, que nos han permitido evolucionar desde máquinas diseñadas exclusivamente para poder ejecutar un único juego a la amplia gama de dispositivos multimedia de los que disponemos actualmente.

Numerosos programas de edición y desarrollo de videojuegos han aparecido en las últimas dos décadas, pero aquellos que son más comerciales están pensados para juegos de todo tipo, por lo que suelen tener características más generales y no tan estrechamente relacionadas con el desarrollo de juegos de rol, y muchas veces restringen algunas de sus funcionalidades, lo que dificulta el desarrollo.

Aquellos programas que sí que están pensados para el desarrollo específico de videojuegos de rol tienen dos problemas fundamentales:

- Los que ofrecen una interfaz intuitiva, sencilla de utilizar, y bastante amigable con nuevos usuarios que están introduciéndose en el mundo del desarrollo de videojuegos están bajo un muro de pago, que pese a no ser muy elevado, hace que usuarios aficionados paguen por un *software* que rara vez utilizarán si no se afianzan finalmente en el mundo del desarrollo.
- Los que no están bajo un muro de pago, es decir, son *software* de código libre, suelen tener interfaces y sistemas complicados de entender para noveles,

quienes debido a la complejidad de estas herramientas deciden abandonar por completo el desarrollo.

La motivación principal de este Proyecto surge de la necesidad de contar con herramientas accesibles y flexibles, tanto para desarrolladores independientes experimentados que quieran crear juegos sin las limitaciones impuestas por los motores de uso general, como para personas sin amplio conocimiento en el desarrollo de videojuegos o en la programación de estos.

1.2. Objetivos

Este Proyecto tiene marcados como objetivos el desarrollo de un motor de videojuegos 2D, pensado específicamente para videojuegos de rol, acompañado de un editor que permita un desarrollo sencillo de videojuegos para este motor. El editor podrá generar ejecutables que el usuario solamente necesite distribuir sin la necesidad de hacer ningún paso extra posterior al desarrollo.

La interfaz del editor estará pensada para usuarios primerizos en el desarrollo, sin eliminar la posibilidad a usuarios más experimentados que quieran hacer juegos de mayor envergadura.

Para ello:

- Se desarrollará un motor multiplataforma que permita la implementación de videojuegos de rol.
- Se desarrollará un editor multiplataforma que facilite la implementación de los videojuegos de rol en el motor desarrollado.
- Se probarán ambas herramientas con usuarios para demostrar el funcionamiento del Proyecto y se extraerán las conclusiones necesarias, así como posibles mejoras de cara al futuro.

1.3. Plan de trabajo

Para cumplir con los objetivos anteriores, se dividirá el plan de trabajo en tres fases:

- Investigación del estado actual sobre los motores y editores específicos para videojuegos de rol, así como de los propios videojuegos de rol. En esta parte se intentará abstraer las características comunes entre todos los motores, editores y videojuegos, tanto los de código libre como los que están bajo una capa de pago; y se intentará proponer mejoras a los problemas que estos puedan tener de cara a nuevos usuarios poco experimentados.
- Diseño del Proyecto. Con las características abstraídas en la fase anterior, se planteará un diseño inicial que servirá como base durante el desarrollo del Proyecto. Este diseño, si bien no será inmutable, debería ser lo más «final» posible para evitar problemas durante la fase de desarrollo.

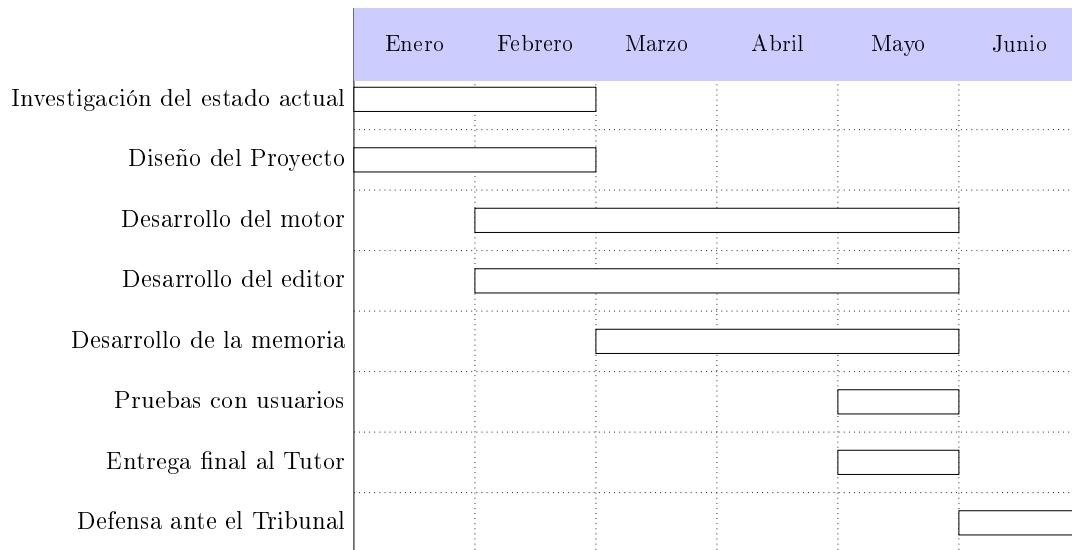


Figura 1.1: Diagrama de Gantt mostrando la planificación temporal del trabajo.

- Desarrollo del Proyecto. Una vez finalizado el diseño del Proyecto, comenzará el desarrollo. Esta fase, a su vez, se dividirá en varias fases:
 - Puesta en marcha del entorno de desarrollo. Se elegirá un entorno de desarrollo dependiendo de las necesidades del Proyecto, y se configurará todo para que sea lo menos trabajoso posible durante el desarrollo de ambas herramientas.
 - Desarrollo del motor. Se desarrollará un motor de acuerdo a los planes diseñados anteriormente, con soporte multiplataforma tanto para ordenador como para dispositivos móviles Android.
 - Desarrollo del editor. Al igual que con el motor, se desarrollará el editor de acuerdo al diseño preestablecido.
- Pruebas con usuarios. Para garantizar el correcto funcionamiento del Proyecto, se probarán las herramientas finales con usuarios de diversa índole ajenos al desarrollo de estas. Al finalizar las pruebas, se extraerán las conclusiones necesarias, y se retocarán en el Proyecto todas aquellas funcionalidades críticas que requieran de un arreglo antes de poder publicar la versión pública, dejando como trabajo futuro aquellas que supongan una excesiva carga como para poder desarrollarlas en el tiempo restante.

1.4. Herramientas y Metodología

En cuanto a las herramientas, se usará Git como sistema de control de versiones, usando un repositorio alojado en GitHub, con la ayuda de GitHub Desktop como herramienta de manejo del este. La gestión de tareas se llevará a cabo a través de los proyectos que GitHub incorpora en su página web.

El acceso al repositorio con el código puede hacerse a través de esta dirección: <https://github.com/almasso/rpgbaker>.

Con respecto a las herramientas de desarrollo del Proyecto, se utilizarán como entornos de desarrollo CLion, para programación en C++, y Android Studio, para la programación en Java de Android; se usará CMake como herramienta de generación de librerías externas; y MinGW como compilador en Windows, Clang como compilador en MacOS, y GCC como compilador en Linux . Las razones detalladas del uso de estas herramientas se encuentran en la sección 4.2**¹.

Por otra parte, la generación del PDF de la memoria se llevará a cabo mediante LATEX, utilizando la plantilla de TEXIS, y utilizando como entorno de edición de esta Texmaker.

En cuanto a la metodología de trabajo, se propondrán reuniones cada dos semanas con el tutor, pudiendo variar el número de semanas dependiendo del progreso realizado. En estas reuniones se expondrá el estado del Proyecto, así como se realizarán consultas referidas al diseño o al desarrollo de este.

La comunicación con el equipo se establecerá, tanto mediante reuniones presenciales cuando se tengan que abordar problemas importantes, como mediante *software* que permita mensajería y chat de voz, como Discord. Es mediante esta herramienta que también se tratará de hacer las pruebas finales con los usuarios.

1.5. Estructura de la memoria

En el capítulo ??, *Estado de la Cuestión*, se abordará la situación actual de los videojuegos de rol, del desarrollo de videojuegos, así como un poco de contexto histórico de ambos.

En el capítulo 4, *Planteamiento del Proyecto*, se tratará en profundidad el diseño planteado y las decisiones tomadas previas al desarrollo de las aplicaciones.

En el capítulo 5, *Desarrollo del Proyecto*, se hablará de la estructura interna del motor y del editor en cuanto a código, abordando también los problemas que se hayan tenido durante esta fase y las soluciones propuestas.

En el capítulo 6, *Evaluación y Conclusiones*, se expondrán las preguntas y objetivos de investigación, el desarrollo de las pruebas con los usuarios, y un análisis acerca de las pruebas, del que se extraerán conclusiones.

Finalmente, en el capítulo 7, *Trabajo Futuro*, se detallarán posibles mejoras para una futura actualización de las herramientas que podrían ser interesantes.

¹ANOT: No estoy del todo de si poner las razones del uso aquí o referenciar la sección posterior donde se desarrolla.

Capítulo 2

Juegos de Rol

RESUMEN: En este capítulo se explicará qué son los juegos de rol, se introducirá su historia, y se examinarán algunos de los juegos de rol más importantes.

2.1. ¿Qué es un juego de rol?

Los juegos de rol (RPG, *role-playing game*) son, en palabras de Lortz (1979, como se cita en Fine (1983)), «todos aquellos juegos que permiten a un determinado número de jugadores asumir los roles de personajes imaginarios y operar con cierto grado de libertad en un entorno imaginario».

Este tipo de juegos se caracteriza por una base muy sólida, que Tychsen et al. (2006) resumen en los siguientes elementos:

- **Narrativa estructurada por reglas:** los juegos de rol se basan en contar una historia dentro de un marco de reglas específicas. Tanto la narrativa como las reglas son únicas para cada juego, proporcionando una estructura que guía la progresión de este y las decisiones de los jugadores.
- **Participación múltiple y mundo ficticio compartido:** los juegos de rol requieren de la participación de múltiples jugadores (al menos dos), quienes interactúan dentro de un mundo ficticio común. Es esencial que todos los participantes comprendan la ambientación, el entorno y las reglas antes de comenzar la partida.
- **Control de personajes por parte de los jugadores:** la mayoría de los participantes asumen el control de al menos un personaje durante toda la partida. A través de estos personajes, los jugadores interactúan con el entorno y con otros personajes, desarrollando la narrativa del juego.
- **Dirección del juego mediante un instructor:** por lo general, existe la figura del director de juego (GM, *gamemaster*) que administra los aspectos del juego no controlados directamente por los jugadores. El *gamemaster* facilita el flujo del juego, proporciona contenido ambiental del mundo ficticio y arbitra las reglas y los conflictos que puedan surgir.

Sumado a estos elementos estructurales, se encuentran otras características igualmente esenciales para definir la experiencia de un juego de rol:

- **Progresión del personaje:** los personajes evolucionan mecánica, narrativa y emocionalmente a lo largo del tiempo mediante la obtención de puntos de experiencia que se pueden «gastar» en un sistema de niveles (Barton, 2008).
- **Inmersión:** resulta fundamental para comprender el atractivo de este género, permitiendo a los jugadores experimentar emociones y tomar decisiones desde la perspectiva de sus personajes (Montola y Stenros, 2010).
- **Narrativa emergente:** es decir, la construcción dinámica y colaborativa de la historia a partir de las acciones de los participantes (Fine, 1983).
- **Decisiones significativas:** las decisiones de los jugadores deben tener consecuencias reales en el desarrollo de la partida (Tekinbas y Zimmerman, 2003).
- **Alto grado de flexibilidad y adaptabilidad:** las reglas, ambientaciones o mecánicas se pueden modificar dependiendo de las necesidades del grupo, lo que evidencia un carácter modular y abierto (Edwards, 2001).

Por otra parte, existen algunos aspectos menos formalizados pero igualmente importantes para el funcionamiento y disfrute de los juegos de rol, como por ejemplo, un contrato social, en el que los jugadores acuerden las temáticas permitidas, el tono de la partida y las dinámicas sociales; una coherencia tonal y de género, que permita mantener una ambientación y tonos coherentes para la inmersión en la partida; fomento de la expresividad de los jugadores, es decir, beneficiarse del uso de la voz, gestos y una narración detallada para enriquecer la interpretación de los personajes; libertad de improvisación frente a lo inesperado; y la exploración emocional y empática con realidades ajenas.

En conjunto, todos estos elementos conforman una experiencia lúdica rica, dinámica y profundamente humana, donde la colaboración, interpretación y narrativa se entrelazan de forma única. Lejos de limitarse a un solo modelo, los juegos de rol han dado lugar a una amplia variedad de formas y enfoques, cada uno con sus propias mecánicas, estilos narrativos y niveles de complejidad.

2.1.1. Tipos de juegos de rol

A lo largo del tiempo, los juegos de rol han evolucionado en múltiples direcciones, dando lugar a distintas topologías con enfoques, estructuras y estilos de juego muy variados. Atendiendo a estas características, se presenta una clasificación general de los tipos más representativos:

2.1.1.1. Juegos de rol de mesa o tablero

Los juegos de rol de mesa (TTRPG, *tabletop role-playing games*), constituyen la forma más tradicional de juegos de rol. En este tipo de juegos, los jugadores se reúnen en torno a una mesa física o virtual y cada uno asume el papel de un



Figura 2.1: Partida de LARP nórdico, extraída de Pohjola (2021).

personaje ficticio. Uno de los participantes actúa como GM, responsable de narrar la historia, interpretar a los NPC (*non-player character*, personaje no jugador) y arbitrar las reglas.

Uno de los elementos más representativos de los TTRPG es el uso de dados poliédricos para resolver acciones inciertas. Por ejemplo, en *Dungeons & Dragons* (Gygax y Arneson, 1974), se utiliza un dado de veinte caras (popularmente conocido como «d20») para realizar la mayoría de las pruebas. Si un personaje intenta realizar una acción desafiante, como escalar una pared o atacar a un enemigo, el jugador tira el dado y suma ciertos modificadores; si el resultado iguala o supera una dificultad determinada por el GM, la acción tendrá éxito.

Estos juegos también incluyen hojas de personaje con estadísticas numéricas (como fuerza, inteligencia o destreza), inventarios, puntos de vida y habilidades especiales. Además, el progreso de los personajes se da mediante la adquisición de experiencia, lo que permite mejorar atributos, aprender nuevas habilidades o subir de nivel.

2.1.1.2. Juegos de rol en vivo

Los juegos de rol en vivo (LARP, *live-action role-playing game*) son una variante en la que los participantes interpretan físicamente a sus personajes en un espacio real, vistiéndose y actuando conforme a su rol. No se juega en torno a una mesa, sino que se representa la historia a través de la acción directa, muchas veces con elementos de escenografía, vestuario y una mayor carga teatral.

Las reglas suelen ser simplificadas o adaptadas para no interrumpir el flujo de la interpretación, y en muchos casos se utilizan sistemas de señales o puntos para resolver conflictos, en lugar de dados. En ciertos LARP de combate, se emplean armas acolchadas para simular enfrentamientos físicos.

Este formato se presta especialmente para la exploración emocional, la inmersión profunda y la representación de tramas políticas, sociales o dramáticas.

Dentro del género LARP, encontramos el subgénero de LARP nórdico, inspirado en la cultura nórdica en la que el fracaso es parte de la trama y es mucho más artístico que los LARP tradicionales, como se puede apreciar en la figura 2.1, con la recreación de un dragón a tamaño real y el involucramiento de numerosos jugadores.

2.1.1.3. Juegos de rol por foro o texto

Este tipo de juegos se desarrollan completamente por escrito, ya sea en foros de internet, chats o correos electrónicos. Los participantes escriben mensajes en los que narran las acciones, pensamientos y diálogos de sus personajes, construyendo la historia de manera colaborativa.

A menudo carecen de un sistema rígido de reglas o tiradas de dados, y dependen más de la narración consensuada y de la calidad interpretativa. En algunos casos se incorporan sistemas de puntos, turnos o moderados para mantener la coherencia y el equilibrio del juego.

Este tipo de juego permite un desarrollo más introspectivo de los personajes y tramas más complejas, ya que los jugadores disponen de tiempo para redactar sus intervenciones.

2.1.1.4. Juegos de rol digitales

Los juegos de rol digitales (CRPG, *computer role-playing games*, juegos de rol de ordenador) son videojuegos que adoptan las mecánicas y estructuras de los juegos de rol tradicionales.

Sobre este tema se ampliará más en el capítulo 3, donde se tratará en profundidad todo lo relacionado con este género de videojuegos.

2.1.1.5. Comparativa

A modo de resumen, se incluye una tabla comparativa entre los distintos tipos anteriormente vistos.

Tipo de juego	Medio	Resolución de acciones	Nivel de inmersión
Rol de mesa (TTRPG)	Presencial / Virtual	Dados y reglas estructuradas	Alto
Rol en vivo (LARP)	Presencial	Señales, puntos o actuación directa	Alto
Rol por foro / texto	Digital (escrito)	Narración compartida y consenso	Alto
Rol digital (CRPG)	Videojuegos	Algoritmos computacionales	Variable

2.2. Historia de los juegos de rol

La historia de los juegos de rol está íntimamente ligada a la evolución de los juegos de estrategia, la literatura de fantasía y la experimentación lúdica del siglo XX. Su origen puede rastrearse a las décadas de 1960 y 1970, con influencias que abarcan desde los *wargames* (juegos de guerra) hasta las obras de J.R.R. Tolkien.



Figura 2.2: Tablero de juego moderno de *Kriegsspiel* (von Reisswitz y von Reisswitz, 1812), extraída de Heinemann (2022).

Los juegos de rol tienen sus raíces en los *wargames*, juegos de estrategia militar que simulan conflictos bélicos con miniaturas y mapas. Entre ellos se destaca el pionero *Kriegsspiel* (von Reisswitz y von Reisswitz, 1812), diseñado por un noble prusiano que quería dotar al ejército de un entrenamiento táctico en caso de guerra. En la figura 2.2 podemos ver el tablero de juego, consistente en una representación cartográfica de un territorio real, y, encima, las unidades militares de ambos bandos (cada una de un color, y cada rango dependiendo del patrón dibujado).

Este tipo de juegos de mesa se fueron desarrollando con el paso del tiempo, apareciendo también libros que contenían una serie de reglas para desarrollar partidas más interesantes, como es el caso de *Little Wars* (Wells, 1913), convirtiendo a este tipo de juegos en simulaciones mucho más narrativas.

No es hasta la década de los 70, con la aparición de juegos como *Chainmail* (Gygax y Perren, 1971), que la comunidad comienza a incorporar elementos de personajes individuales y no colecciones de unidades militares. Con este suceso, se comienza a hablar propiamente de «juegos de rol», y se empiezan a ver los primeros juegos comerciales de este tipo.

Se considera que *Dungeons & Dragons* (Gygax y Arneson, 1974) es el primer juego de rol comercial. Este juego combinaba reglas derivadas de los *wargames* con elementos narrativos inspirados en la literatura fantástica. Cada jugador asumía el papel de un aventurero (por ejemplo, un guerrero o un mago) y exploraba mazmorras, resolvía acertijos y combatía monstruos, todo bajo la guía de un GM. Esta revolucionaria idea de interpretar un personaje y tomar las decisiones desde su perspectiva sentó las bases de este género, que poco a poco se fue popularizando.

En la década de los 80, este género se expandió, dando a lugar a nuevos sistemas y ambientaciones, como la saga de horror *Call of Cthulhu* (Petersen y Willis, 1981)

o la de fantasía *RuneQuest* (Perrin et al., 1983). Los temas y las mecánicas comienzan a diversificarse, y se comienzan a publicar suplementos, novelas y productos derivados que expandían los universos de los juegos. También aparecen los primeros videojuegos que incorporaban mecánicas de estos juegos de tablero, apareciendo el género de los CRPG. Es a finales de esta época cuando comienzan a surgir los primeros LARP, y se empiezan a organizar convenciones temáticas, consolidando la cultura de jugadores de este género.

Entrando en la nueva década, crecen las comunidades LARP tanto en Estados Unidos como en Europa gracias a juegos como *Mind's Eye Theatre* (Rein-Hagen y Davis, 1993), mientras que la normalización de internet en los hogares permitió el surgimiento del rol por foro y correo electrónico, ampliando el acceso a este tipo de contenidos.

Finalmente, en el siglo XXI, el rol se diversificó aún más. Los CRPG se volvieron cada vez más complejos y tenían un mayor alcance, surgen los MMORPG (*massively multiplayer online role-playing game*, videojuegos de rol multijugador masivos en línea), que ofrecen experiencias compartidas masivas, mientras que el auge de plataformas como Discord, Roll20 o Foundry Virtual Tabletop revitalizó el juego de rol de mesa en línea.

La popularización del *streaming* en la década del 2010, y la aparición de *podcasts* de rol como *Critical Role* ayudaron a introducir el género a nuevas generaciones, mostrando su gran potencial narrativo y creativo. Hoy en día, los juegos de rol son practicados por millones de personas en todo el mundo, en formatos que van desde campañas caseras hasta multitudinarios eventos de LARP, o desde videojuegos AAA hasta partidas por texto en móviles.

2.3. Análisis de los principales juegos de rol

2.3.1. *Dungeons & Dragons*

Como se ha mencionado anteriormente, *Dungeons & Dragons* (Gygax y Arneson, 1974), a veces conocido únicamente como D&D, es ampliamente conocido como el primer juego de rol comercial y el más influyente del género, y su origen se basa en el sistema de reglas de *Chainmail* (Gygax y Perren, 1971), al cual se le añadieron elementos de exploración, personajes individuales y narrativa.

D&D se estructura en torno al concepto de un grupo de aventureros que exploran mundos de fantasía, enfrentan criaturas, resuelven conflictos y evolucionan mediante un sistema de niveles y experiencia. El director de juego (aquí llamado *dungeon master*, DM) narra la historia, describe el mundo y controla a los NPC y eventos.

A partir de la tercera edición, del año 2000, se introduce el *sistema d20*, un dado de veinte caras en el que se basa todo el sistema de reglas, así como hojas de personaje detalladas con diversos atributos.

En la figura 2.3 podemos ver varios ejemplos de partidas de D&D, siendo la que está capturada en la figura 2.3a un ejemplo de una partida normal, con un tablero



(a) Imagen de una partida común de D&D, extraída de Anh y Nguyén (2021).



(b) Imagen de un diorama para el escenario de una partida de D&D, extraída de García (2016).

Figura 2.3: Imágenes de distintas variedades de partidas de *Dungeons & Dragons*.

y sus diferentes *d20*, y, en la figura, 2.3b, un ejemplo de un diorama creado por un aficionado a la maquetación, que cuenta con todo lujo de detalles y que llega a ocupar toda una sala.

El juego ha pasado por múltiples ediciones, cada una introduciendo cambios en las mecánicas, balance y filosofía de diseño. Su edición más reciente, la quinta, del año 2014, ha tenido un gran éxito comercial y ha contribuido al resurgimiento del juego de rol gracias a su gran accesibilidad, su enfoque narrativo y la visibilidad en plataformas como YouTube o Twitch.

El impacto de este juego en la cultura popular y en el diseño de otros juegos es difícil de sobreestimar, ya que está considerado el estándar de facto en los TTRPG y ha inspirado hasta adaptaciones cinematográficas¹.

2.3.2. *Pathfinder*

Pathfinder (Jason Bulmahn, 2009) nació como una evolución del *sistema d20* de D&D. Ante el cambio de enfoque que supuso la cuarta edición de D&D, muchos jugadores buscaron una alternativa que conservara la complejidad y flexibilidad del sistema anterior. Se aprovechó de la licencia OGL (*Open Game License*, Licencia de Juego Abierto)² para crear un sistema propio que mejorara y expandiera el marco de reglas original.

El juego destaca por su nivel de detalle, la personalización de personajes y la riqueza de opciones tácticas durante el combate. El sistema de creación de personajes permite una gran variedad de combinaciones de clases, habilidades, dotes y objetos mágicos. Esto atrae especialmente a jugadores que valoran la optimización mecánica y la profundidad estratégica.

¹La saga *Dungeons & Dragons* (Solomon, 2000) ha recaudado más de 250 millones de dólares en total sumando las cuatro películas, siendo la última entrega, de 2023, la más exitosa de todas.

²Esta licencia se usa principalmente en TTRPG y concede, por parte de los diseñadores, permisos de modificación, copia y redistribución de algunos de los contenidos diseñados para sus juegos, principalmente mecánicas.

Narrativamente, mantiene el enfoque épico y de alta fantasía de D&D, pero con mundos propios, combinando elementos clásicos con tramas políticas, horror cósmico o mitologías exóticas.

En 2019 se publicó su segunda edición, que renovó el sistema con nuevas mecánicas más modernas, como un sistema de tres acciones por turno y una progresión más clara. Pese a que no ha alcanzado el nivel de popularidad de D&D, *Pathfinder* ha consolidado una gran comunidad y ha mantenido una identidad propia, por lo que muchas páginas lo sitúan siempre en segundo o tercer lugar de popularidad³.

2.3.3. *Call of Cthulhu*

Call of Cthulhu (Petersen y Willis, 1981) está basado en los mitos creados por H.P. Lovecraft, y representa un enfoque radicalmente distinto al de otros juegos de rol de la época, ya que se centra en la investigación, el horror psicológico y la fragilidad humana ante lo desconocido.

El sistema utiliza como base las reglas BRP *Basic Role-Playing*, Juego de Rol Básico, que utiliza porcentajes para determinar el éxito de las acciones. Los personajes son investigadores en los años 20 (aunque hay adaptaciones a otras épocas), y deben enfrentarse a cultos secretos, criaturas indescriptibles y horrores cósmicos. A diferencia de otros juegos, la muerte o locura de los personajes es común y está integrada en la narrativa, lo que fomenta una experiencia inmersiva y tensa.

El atributo de cordura (SAN, del inglés *sanity*) es central, ya que los encuentros con lo sobrenatural pueden erosionar la mente de los personajes hasta conducirlos a la locura. Esta mecánica refleja la atmósfera opresiva de las obras *lovecraftianas* y crea un estilo de juego más introspectivo y sombrío.

Call of Cthulhu ha sido aclamado por su capacidad de generar tensión narrativa, y ha influido en muchos otros juegos de rol e investigación, convirtiéndolo en un pilar fundamental del rol narrativo.

2.3.4. *Warhammer Fantasy Roleplay*

Warhammer Fantasy Roleplay (Halliwell et al., 1986) comparte ambientación con el universo de miniaturas de *Warhammer Fantasy Battle* (Ansell et al., 1983) (ver figura 2.4), pero se diferencia por su tono más oscuro, realista y decadente.

Ambientado en un mundo inspirado en la Europa del Renacimiento, pero plagado de corrupción, magia caótica y criaturas monstruosas, WFRP pone al jugador en la piel de personajes que suelen comenzar como personas comunes (rateros, campesinos o aprendices), y no como héroes épicos. El juego enfatiza la progresión lenta, la supervivencia y la toma de decisiones difíciles.

Su sistema de reglas, basado también en porcentajes, incluye un sistema de carreras profesionales que estructura el desarrollo del personaje de forma naturalista. Las

³La página *The Dragon's Trove* lo sitúa en tercer lugar en su histórico: <https://www.dragonstrove.com/blogs/news/the-best-tabletop-rpgs-of-all-time>.



Figura 2.4: Partida de *Warhammer Fantasy Battle*, extraída de Paddington (2023).

heridas son graves, la corrupción mágica puede deformar al personaje, y la muerte puede llegar de forma repentina. Esto genera un entorno de juego más peligroso y centrado en la narrativa emergente.

WFRP es un referente del estilo conocido como *grimdark*⁴, que enfatiza mundos crueles, moralidades grises y destinos trágicos. Esta saga ha sido particularmente influyente en Europa, y ha servido de inspiración para sagas de videojuegos como la de *The Witcher*.

⁴La expresión *grimdark* proviene de una de las entregas de la saga *Warhammer*, concretamente, *Warhammer 40000*, en la que se explica que «In the **grim** darkness of the far future there is only war» (En la sombría oscuridad del futuro lejano solo hay guerra).

Capítulo 3

Videojuegos de Rol

RESUMEN: En este capítulo se explicará qué son los videojuegos, haciendo un enfoque en los videojuegos de rol, se introducirá su historia y se hablará del proceso de desarrollo de estos, analizando la importancia de los motores y editores en el desarrollo de videojuegos.

3.1. ¿Qué es un videojuego?

- 3.1.1. El problema de los géneros de videojuegos
- 3.1.2. Historia de los videojuegos de rol

3.2. Sobre el desarrollo de videojuegos

3.2.1. Motor de videojuegos

- 3.2.1.1. Componentes de un motor de videojuegos
- 3.2.1.2. Separación entre motor y *gameplay*
- 3.2.1.3. Programación dirigida por datos (DDP) en videojuegos
- 3.2.1.4. Programación multiplataforma en videojuegos

3.2.2. Editor de videojuegos

- 3.2.2.1. Editores específicos de videojuegos para desarrollo de RPG

Capítulo 4

Planteamiento del Proyecto

RESUMEN: En este capítulo se tratarán los objetivos principales del proyecto, así como las decisiones tomadas en cuanto a diseño del motor y del editor.

4.1. Objetivos principales del Proyecto

La idea principal es el desarrollo de un motor de videojuegos, enfocado a los RPG 2D, acompañado de un editor que permita un desarrollo rápido y simple de juegos de este tipo para el motor desarrollado. El editor estaría pensado principalmente para gente no programadora o sin experiencia en el desarrollo de videojuegos, por lo que la interfaz tendría que ser intuitiva y fácil de utilizar y aprender.

El editor debería ser capaz de generar un ejecutable, que por debajo utilice el motor desarrollado previamente, con el diseño de juego realizado por el usuario en el propio editor, y que pueda ejecutarse en Windows, MacOS, Linux y Android. El editor, por su parte, ha de poder ser ejecutado tanto en Windows, MacOS o Linux, descartando la ejecución en dispositivos móviles¹.

El usuario podrá elegir la plataforma para la cual se va a generar el ejecutable, y el editor se encargará de transferir el contenido desarrollado en el proyecto a la *build*, asegurándose de que el comportamiento volcado es el mismo que el diseñado previamente y generando una *build* lista para empaquetar y distribuir, sin requerir pasos adicionales por parte del usuario.

Por otra parte, se espera que el editor pueda generar diversos proyectos (es decir, distintos juegos), y capaz de guardar el estado de un proyecto y poder recuperarlo cuando el usuario desee, sin que se hayan podido perder los cambios que se hayan realizado. Y, también, debe poder exportar proyectos, e importar otros que otros usuarios puedan haber diseñado en otras plataformas o sistemas sin mayor dificultad.

El motor, por su parte, aportará la mayor parte del *gameplay*, para que el usuario solo tenga que desarrollar la parte de diseño (principalmente el diseño artístico y

¹Los editores suelen tener elementos que son preferibles de ser utilizados mediante entrada de teclado y ratón. Si bien es cierto que Android permite la conexión de periféricos de entrada-salida, es una plataforma pensada para dispositivos móviles y táctiles.

visual. Tendrá que tener las funcionalidades básicas que se esperan de un RPG, así como soporte para periféricos de entrada-salida tradicionales (teclado y ratón) y entrada táctil (para los dispositivos móviles).

4.2. Toma de decisiones

La primera decisión a tomar fue la plataforma de desarrollo del Proyecto. Se decidió desarrollar el grueso del Proyecto en C++, ya que se quería aprovechar el alto rendimiento que ofrece en comparación a otros lenguajes², el soporte multiplataforma que tiene la familia C/C++ tanto en dispositivos de sobremesa como en móviles, y el uso de librerías más avanzadas que facilitarían el desarrollo del Proyecto.

Debido a la premisa de un desarrollo multiplataforma, se necesitaba usar un IDE (*integrated development environment*, entorno de desarrollo integrado) que fuese compatible tanto con Windows, MacOS, y Linux. La opción que en un principio se había valorado era la de utilizar *Visual Studio*, una de las herramientas más populares para el desarrollo en C++; sin embargo, debido a que este IDE carece de versiones para MacOS y Linux³, se optó por hacer el desarrollo del Proyecto en *CLion*, un IDE con soporte para CMake, que facilitaría a la hora de agilizar el trabajo (por su rapidez en la generación de proyectos complejos) y con la gestión de las dependencias externas.

Pese a que el aprender a usar CMake ocupó gran parte del inicio del desarrollo del Proyecto, las ventajas que ha supuesto a la hora del manejo de las distintas dependencias externas frente a otras alternativas que se habían manejado a lo largo del transcurso del Grado, han hecho que la inversión temporal en esta opción haya resultado beneficiosa.

Por otra parte, se tendría que utilizar otra herramienta para el desarrollo de la APK (*Android Application Package*, paquete de aplicaciones Android, es decir, el ejecutable de Android), ya que esta se debe desarrollar utilizando Java, por lo que se decidió utilizar *Android Studio*, la herramienta oficial de desarrollo para Android, que proporciona máquinas virtuales de dispositivos Android de distintas versiones y generaciones para poder probar la *build*. Otra ventaja añadida al uso de Android Studio es el soporte que tiene para CMake, que ha permitido disponer de un único archivo de configuración para ambas herramientas.

Dentro de *Android Studio*, se tendría que añadir también el módulo de NDK (*Native Development Kit*, kit de desarrollo nativo) que permite el desarrollo de aplicaciones para Android utilizando llamadas a C/C++ gracias a JNI (*Java Native Interface*, interfaz nativa de Java) integrada en el SDK de Java. JNI es un ejemplo de una FFI (*foreign function interface*, interfaz de funciones foráneas), es decir, un

²El hecho de poder gestionar la memoria utilizada en cualquier momento, así como la ausencia de una máquina virtual intermedia hacen que C++ sea el lenguaje idóneo para proyectos donde el rendimiento es crítico.

³MacOS y Linux cuentan con *Visual Studio Code*, que si bien sirve para poder compilar C/C++ mediante el uso de *plug-in*, es más complicado de configurar para proyectos más complejos como este.

mecanismo por el cual un lenguaje de programación puede llamar a funciones o rutinas programadas o compiladas en otro lenguaje distinto, lo cual es necesario para poder ejecutar el juego, ya que la entrada de la aplicación Android estaría en Java.

El resto de decisiones son propias de cada una de las partes del Proyecto y se detallarán a continuación.

4.2.1. Diseño del motor

4.2.1.1. Base con el sistema entidad-componente

La base del motor se estructurará atendiendo a un patrón EC (entidad-componente). Los sistemas específicos de los juegos RPG se construirán sobre esta base modular, que se puede separar en dos partes diferenciadas: el bucle de juego y la carga de recursos.

El bucle de juego se estructurará en escenas. Cada escena estará formada por un conjunto de entidades que se actualizarán en paralelo. A su vez, cada una de estas entidades agrupará un conjunto de componentes, que encapsularán funcionalidades específicas, permitiendo una gran flexibilidad mediante el uso de polimorfismo. Gracias a esta estructura se podrán implementar una amplia variedad de mecánicas mediante la creación de distintos componentes asociados a las entidades.

En cuanto a la carga de recursos, este sistema se encargará de informar al motor sobre las escenas creadas y su contenido. Estas escenas estarán descritas cada una en un fichero Lua, estructurado de forma que puede descomponerse fácilmente en sus entidades, y, a su vez, en los componentes que las conforman.

Lua ha sido elegido por su simplicidad, su integración fluida con C++ mediante librerías como `sol2`, y su compatibilidad multiplataforma. Estas librerías proporcionan *bindings*, es decir, código que facilita el uso de Lua desde C++ y viceversa.

La parte más compleja de este sistema es la conversión de estructuras de texto en Lua a instancias de clases de C++ específicas. Para ello, se usará un *patrón factoría*: cada componente estará asociado a un identificador, que estas «factorías» utilizarán para instanciar el componente correspondiente y añadirlo a la entidad.

Además de la carga de escenas, el sistema de carga de recursos permitirá gestionar múltiples tipos de recursos que pueden ser utilizados por el resto del motor. Cuando se solicite un recurso, este se cargará en memoria si no lo estaba previamente. Esta operación se realizará hasta alcanzar un límite de memoria, definido por el usuario. Una vez alcanzado dicho límite, si se solicitase un nuevo recurso, se aplicará un algoritmo LRU (*least-recently-used*, menos usado recientemente), que liberará el recurso que más tiempo lleve sin usarse para hacer espacio al nuevo.

4.2.1.2. Componentes genéricos de juego

Para desarrollar videojuegos, es necesario contar con ciertas funcionalidades básicas que faciliten la implementación de los sistemas específicos del juego.

En el motor, se utilizará la librería **SDL** para implementar estos sistemas. Esta elección se debe principalmente, a su sencillez de uso y a su robusto soporte multiplataforma, que permite ejecutar los juegos tanto en sistemas de escritorio como en dispositivos Android.

Los sistemas que se implementarán serán los siguientes:

- Sistema de *renderizado*: es imprescindible contar con un mecanismo que permita mostrar visualmente lo que ocurre en el juego. Dado que se trata de un motor para videojuegos 2D, se usarán imágenes y texto para cubrir esta necesidad. Estos sistemas de *renderizado* se expondrán al usuario a través de componentes que permitirán mostrar y animar imágenes, mostrar texto y controlar una cámara desplazable. Además, se implementará un sistema de *renderizado* basado en capas y escenas, que permitirá superponer diferentes escenas, lo cual resulta útil para mostrar elementos (como menús) en forma de *overlay*, y permite tener un mayor control sobre el orden de *renderizado*.
- Sistema de *input*: se desarrollará un sistema de entrada sencillo basado en clics y toques. Este diseño garantiza la compatibilidad multiplataforma y permite centrar la jugabilidad en la interacción mediante botones, en línea con la experiencia tipo *point-and-click* que se quiere ofrecer. El motor unificará el *input* mediante clic y toque en una estructura común que incluya la posición del evento y su estado (inicio, mantenido o final) en un determinado fotograma.
- Sistema de sonido: la retroalimentación auditiva es un componente esencial en los videojuegos. Para su implementación, se empleará el nuevo sistema de sonido incluido en **SDL3**. Desde el punto de vista del desarrollador, su uso es sencillo: se creará un componente que permita reproducir, pausar, detener y reanudar los sonidos. Además, estos pueden agruparse en diferentes conjuntos, lo que permite controlar de forma independiente el volumen general, el de música y el de los efectos sonoros.
- Sistema de colisiones: está diseñado para cubrir las necesidades de los juegos previstos, donde será suficiente con detectar intersecciones entre rectángulos. A través del componente correspondiente, será posible comprobar si un objeto colisiona con otro, si acaba de entrar en colisión o si ha dejado de colisionar.

4.2.1.3. Componentes específicos de RPG

En primer lugar, se ha diseñado un sistema de movimiento automático y detección de colisiones basado en una cuadrícula de casillas y un algoritmo A* de búsqueda de caminos. Para ello, se mantendrán actualizadas las posiciones ocupadas dentro de la cuadrícula, tanto por elementos estáticos como por elementos dinámicos, y se recalculará la ruta siempre que sea necesario. Este sistema se aplicará tanto a los NPC, a través del sistema de eventos y sus comportamientos asociados, como al jugador, mediante un sistema de entrada de tipo *point-and-click*.

También se implementará un sistema de carga dinámica de mapas y transición entre ellos. Mientras el jugador se encuentra en un determinado mapa, el motor

cargará en segundo plano los mapas adyacentes. Al cambiar de un mapa a otro, se descargará aquellos que ya no sean necesarios. Esta estrategia permite equilibrar el uso de memoria y los tiempos de carga, ofreciendo una solución eficiente y escalable.

Adicionalmente, se dispondrá de un sistema de diálogos con cuadros de texto que muestren el contenido de forma dinámica. El jugador podrá avanzar en la conversación mediante una entrada sencilla, mientras que el motor gestionará automáticamente los saltos de línea y el ajuste del tamaño del texto. Asimismo, se integrará un selector de opciones que permita al jugador elegir entre distintas alternativas mediante botones interactivos.

Las decisiones tomadas se registrarán, lo que permitirá diseñar desde el editor sistemas de interacción complejos al estilo clásico de los RPG. Este sistema hará uso de variables locales (propias de cada objeto del mapa) y globales del jugador (asociadas a la partida), permitiendo un control detallado de la progresión sin requerir programación adicional.

Por último, se desarrollará un componente destinado a facilitar el diseño de la lógica específica del juego mediante comandos sencillos: el gestor de eventos. Este componente constituirá el núcleo del sistema de interacción y comportamiento en el mundo del juego.

4.2.1.4. Sistema de eventos

Uno de los objetivos principales del motor es ofrecer la posibilidad de crear un RPG sin necesidad de conocimientos de programación. Para ello, se ha diseñado un sistema de eventos: una estructura que permite controlar la lógica del juego mediante instrucciones de alto nivel.

La idea fundamental es sencilla: gracias al componente de gestión de eventos, una entidad podrá contener un conjunto de eventos. Cada uno de estos eventos estará compuesto por dos elementos principales: una condición y un conjunto de comportamientos.

La condición determinará bajo qué circunstancias deberá activarse el evento. Existirán múltiples tipos de condiciones, encargadas de evaluar distintos aspectos del estado del juego. Si una de las condiciones se cumple, el evento asociado comenzará su ejecución. Además, será posible combinar condiciones mediante operadores lógicos (*not*, *or*, *and*), lo que permitirá definir reglas más complejas y flexibles.

La segunda parte del evento serán los comportamientos, encargados de definir qué acciones se deben realizar una vez activado el evento. Estos comportamientos actuarán como una lista de instrucciones que modifiquen el estado del juego a través de parámetros sencillos. Estarán escritos en Lua y se encargarán de invocar distintas funciones del motor: desde mover objetos, cambiar animaciones o modificar la música, hasta iniciar diálogos.

El sistema está diseñado para que la ejecución de los comportamientos sea progresiva: en cada actualización del juego se ejecutará uno de los comportamientos activos del evento. En los casos en los que una acción requiera más de una actualización para completarse, el comportamiento se limitará a establecer un objetivo,

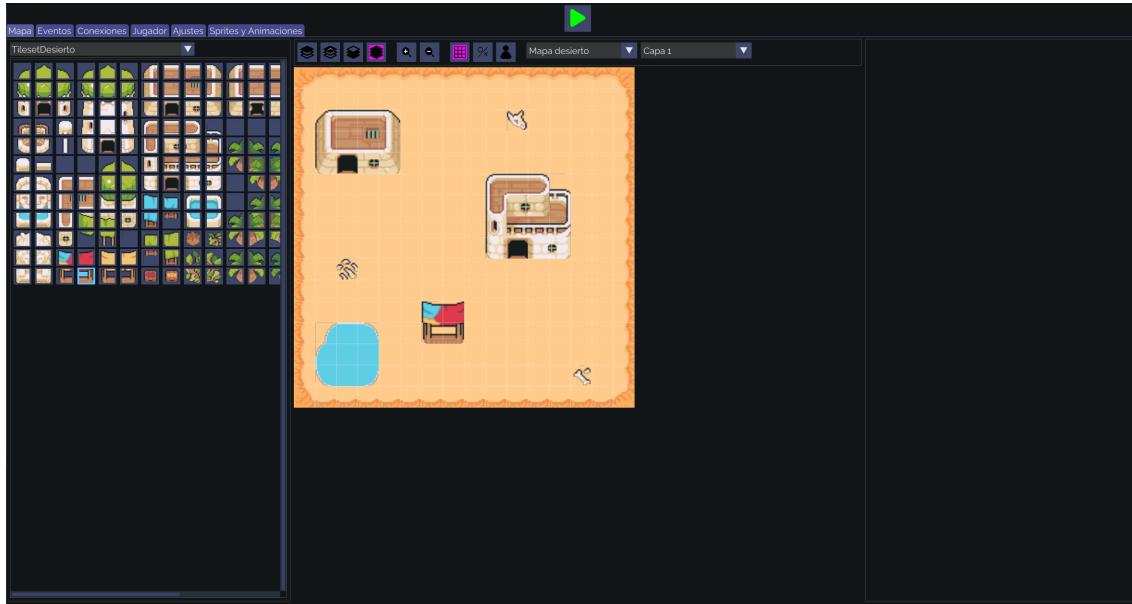


Figura 4.1: Editor de mapas de *RPGBaker*.

que será procesado por el resto de sistemas mientras el evento continúa ejecutando el resto de sus comportamientos.

Existirán además comportamientos orientados al control del flujo de ejecución. Por ejemplo, uno de ellos permitirá esperar a que se cumpla una condición antes de continuar, actuando como un mecanismo de bloqueo temporal. También, se implementarán instrucciones de salto que permitirán la creación de bucles o la bifurcación de la ejecución según el estado del juego.

Gracias a este sistema, que actúa como lenguaje de *scripting* simplificado, será posible crear lógicas de juego complejas mediante herramientas accesibles y visuales, sin la necesidad de escribir código complejo.

4.2.2. Diseño del editor

4.2.2.1. Funcionalidad del editor

La esencia del editor es la de una interfaz intuitiva y sencilla de utilizar y aprender para aquellos usuarios noveles que nunca hayan utilizado una herramienta similar. Para evitar cargar cognitivamente a los usuarios con información textual, se ha optado por el uso de símbolos e iconos acompañados por descripciones emergentes cortas en la mayoría de elementos.

Las características fundamentales que el editor tendría correspondían con aquellos elementos que podían ser modificados en el motor, es decir:

- Editor de mapas (figura 4.1), que permitiese al usuario cargar sus propios *tilesets*; decidir el tamaño de la cuadrícula que ocuparía el mapa; dibujar el mapa sobre la cuadrícula utilizando las baldosas del *tileset*, con la posibilidad

de tener varias capas para poder simular un efecto de profundidad; y establecer las regiones de colisión del mapa.

- Editor de eventos, que permitiese al usuario la creación de eventos, asignándoles la condición de lanzamiento y los comportamientos que se ejecutarán cuando la condición se cumpla. El editor de eventos tendría que tener soporte para eventos complejos, en el caso de que el usuario requiera de la creación de uno de ellos.
- Editor de *sprites* y de animaciones, que permitiese al usuario generar un *sprite* dada una imagen o *spritesheet*, y, posteriormente, animaciones dada una serie de *sprites*. Ambos editores tendrían una previsualización del *sprite* o animación, añadiéndose controles para la reproducción en este último caso.
- Editor de objetos, que permitiese al usuario generar un objeto para cada una de las baldosas en la cuadrícula del mapa y asignarle un *sprite* y un evento.
- Editor de personaje, que permitiese al usuario personalizar su *sprite*, añadirle animaciones de movimiento y configurar diversos parámetros adicionales, como su posición de origen en el mapa.
- Editor de conexiones entre los mapas, que permitiese al usuario establecer las posiciones de los distintos mapas en el mundo, de manera visual e intuitiva.
- Editor de ajustes generales del ejecutable final, como por ejemplo el nombre del juego, dimensiones de la cámara, la fuente por defecto a utilizar en los textos, o el mapa inicial del juego.

Todos estos elementos tendrían la capacidad de poder ser configurados al gusto del usuario, permitiendo ser editados y eliminados cuando este desee.

Además, el editor incluiría un *viewport* para que el usuario pudiese ver en tiempo real el diseño final del juego, sobre el cual se podría ejecutar y probar las funcionalidades implementadas, al estilo de *Unity*.

Por otra parte, el editor contará con un sistema de persistencia, que permita una carga y guardado de los *proyectos*, es decir, la representación de un juego en el editor. Todas los *assets* utilizados tendrán que estar referenciados en archivos de configuración del *proyecto*, y el guardado actualizará estos archivos de configuración.

4.2.2.2. Modularidad y arquitectura

En cuanto a la estructura de la interfaz, se optó por utilizar un patrón típico en el desarrollo de *software* no relacionado con videojuegos, basado en ventanas (o subventanas) anidadas en otras ventanas, con el uso de ventanas modales que apareciesen sobre estas. Esta estructura permite una escalabilidad del proyecto mucho más sencilla en caso de futuras expansiones y una mayor modularidad con cada uno de los componentes que se quisiese integrar.

Las ventanas tendrían comunicación unas con otras utilizando al *proyecto*, que almacenaría toda la información referente a cada uno de los juegos que el usuario crease (por ejemplo, referencias a los *tilesets*, *sprites*, animaciones o mapas creados).

Se tendrían también diversos gestores, tanto de *scripting*, como de elementos de entrada-salida de ficheros, como de preferencias del usuario y hasta un gestor de idiomas, que permite que el Proyecto sea multilingüe⁴ con una amplia escalabilidad en el caso de que se quisiesen añadir más idiomas.

4.2.2.3. Tecnologías utilizadas

Para conseguir todos los objetivos anteriores, se investigó qué librerías se iban a poder utilizar para conseguir desarrollar toda la interfaz y la funcionalidad básica. Al ya haber optado por utilizar **SDL3** como base para el motor, se optó también como base para el editor, acompañada de **DearImGui** para el dibujado y manejo de los elementos de la interfaz.

También, al ya haber optado por el uso de **Lua** como lenguaje de *scripting* y de definición de datos para el motor, se optó por el uso del mismo para el editor, acompañados por la anteriormente mencionada librería **sol2**.

Los archivos de configuración del proyecto difieren de los que el motor espera recibir, ya que muchas veces el editor espera recibir más datos de los que el motor necesita (por ejemplo, rutas específicas de *assets*, guardado de *tilesets*, etc...). Es por eso que el editor se tiene que encargar de «traducir» estos ficheros de definición de datos a los datos que el motor espera; esto se hará en el tiempo de generación del ejecutable final. Esta decisión se ha tomado para que, si no se dispone de los ficheros de configuración del *proyecto*, un usuario ajeno al diseño del juego sea incapaz de poder modificarlo, al igual que como ocurre en la gran mayoría de motores⁵.

Si bien es cierto que una de las técnicas más comunes en la implementación de un editor es la de utilizar el motor para el que se desarrolla como base, ya que esto evita el tener que implementar dos veces las mismas funcionalidades (principalmente en *renderizado*, *input* y *scripting*), se decidió independizar el desarrollo del editor del motor para poder avanzar más rápidamente en el Proyecto. Esta opción, pese a que ha supuesto una mayor carga de trabajo, ha permitido flujos de iteración en el desarrollo más cortos.

⁴En un principio, únicamente en castellano y en inglés, pero debido al sistema implementado, es muy sencillo añadir nuevos idiomas en el caso que fuese necesario.

⁵Esto es debido a que, en muchos casos, se quiere que el jugador final evite poder hacer modificaciones al juego si no dispone del código original de este, evitando posibles casos de piratería o distribución no autorizada.

Capítulo 5

Desarrollo del Proyecto

RESUMEN: En este capítulo se tratará la implementación del motor, la del editor, así como una puesta en marcha del entorno de desarrollo utilizado.

5.1. Puesta en marcha del *toolchain*

Antes de comenzar a desarrollar ambas herramientas, se desarrolló un *toolchain* (cadena de herramientas) que permitiese la generación de proyectos, tanto para el desarrollo en C++, como para el desarrollo en Java de Android.

Para ello, fue necesario configurar el archivo `CMakeLists.txt` de la raíz del Proyecto. Este fichero CMake permite la descarga de las dependencias externas mediante el uso del módulo `FetchContent`. Aquí se encuentra el primer «inconveniente», derivado del uso de este módulo: `FetchContent` descarga las librerías externas que se le pidan tantas veces como plataformas y tipos de compilación posibles haya. Esto provoca que una librería muy sencilla se necesite descargar ocho veces para poder generar el ejecutable final del juego para cada una de las plataformas destino deseadas (en total, cuatro plataformas destino y dos tipos de compilación por plataforma).

Pese a haber investigado alguna manera sencilla de poder evitarlo, el comportamiento de este módulo es fijo y no se ha podido encontrar una alternativa.

Por otra parte, atendiendo a la figura 5.1, cada uno de los proyectos contiene a su vez un fichero `CMakeLists.txt` que configura las direcciones de inclusión y de enlazado de cada una de las herramientas, así como la detección de los ficheros `.cpp` necesarios para la compilación. Esta estructura modular permite un mejor mantenimiento de cada proyecto, manteniendo cada módulo separado y encapsulado, lo que permite a su vez facilidad a la hora de las pruebas durante el desarrollo.

Debido a que el proyecto `Engine` ha sido concebido como una librería dinámica, se ha necesitado la creación de un proyecto `Executable`, que genera un ejecutable con llamadas a la API provista en `Engine`. Este será el ejecutable final que lance al juego.

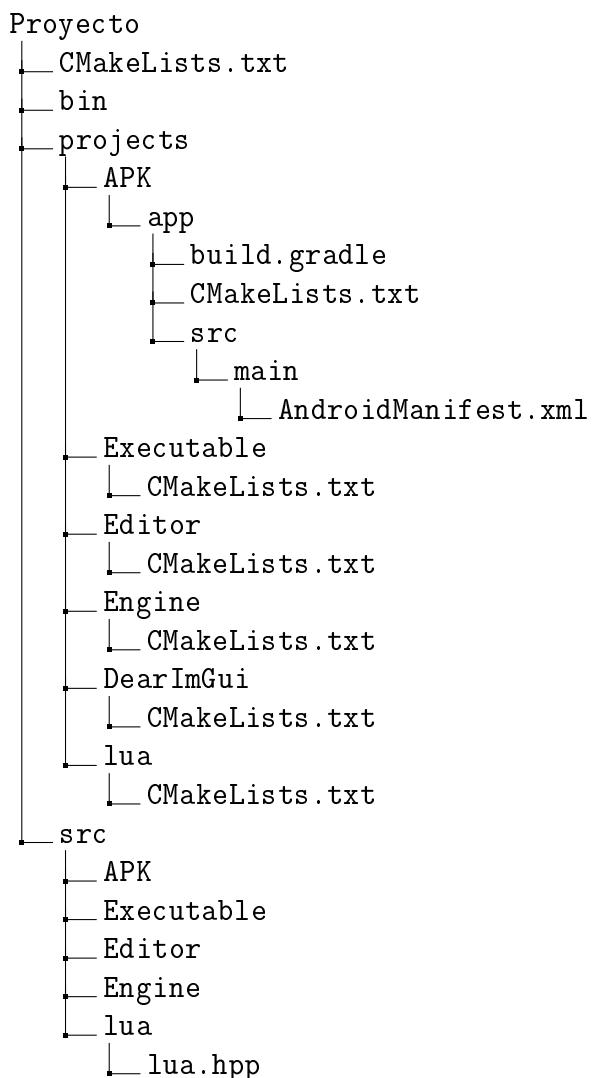


Figura 5.1: Estructura de carpetas del Proyecto.

En cuanto a los proyectos `lua` y `DearImGui`, se han generado los `CMakeLists.txt` correspondientes para su sencilla incorporación, debido a que estas dos librerías no disponen de soporte inicial para CMake.

También es necesario mencionar que se ha añadido un fichero de cabeceras de C++, `lua.hpp`, con el siguiente contenido:

```
extern "C" {
    #include <lua.h>
    #include <lualib.h>
    #include <lauxlib.h>
}
```

Esto es debido a que la librería de Lua para C/C++ está escrita en C, por lo que es necesario decirle al compilador de C++ que trate todo el contenido de la librería como si fuese código C. Así se evita cualquier problema a la hora de enlazar el proyecto con la librería de Lua debido al *name mangling* (o modificación de nombres) que el compilador de C++ pueda hacer.

En cuanto a la generación del proyecto en Android, se ha generado un fichero `build.gradle` que habilita el uso del NDK y JNI para poder ejecutar el motor desde una APK, así como la configuración de la entrada de la aplicación y del manifiesto de Android (archivo en formato XML que tiene información esencial de la aplicación).

Gracias a `SDL`, no ha hecho falta programar la entrada a la aplicación en Java, sino que es la propia librería la que se encarga de llamar automáticamente al código C++ mediante un extra para la programación en Android.

5.2. Desarrollo del motor

5.3. Desarrollo del editor

5.3.0.1. Estructura del código y organización de módulos

Capítulo 6

Evaluación y Conclusiones

Conclusiones del trabajo y líneas de trabajo futuro.

Antes de la entrega de actas de cada convocatoria, en el plazo que se indica en el calendario de los trabajos de fin de grado, el estudiante entregará en el Campus Virtual la versión final de la memoria en PDF.

6.1. Objetivo de la evaluación

6.2. Metodología

6.3. Resultados

6.4. Análisis de los resultados y conclusiones

Capítulo 7

Trabajo Futuro

Introduction

“I had this really bizarre conversation once with a couple of lawyers and they were talking about «How do you pick your target market? Do you use focus groups and poll people and all this?» It’s like «No, we just write games that we think are cool.»”

— John Carmack

Motivation

Role-playing video games have been one of the most influential genres in the industry, from their origins in the 1980s to the present day, where they account for a large portion of the market share.

The development of this type of game has undergone major changes over the years, driven by continuous hardware and software improvements, which have allowed us to evolve from machines designed solely to run a single game to the wide range of multimedia devices available today.

Numerous video game development and editing tools have appeared in the last two decades, but the more commercial ones are designed for games of all kinds. As a result, they tend to offer more general features that are not closely related to role-playing game development, and they often restrict certain functionality, making the development process more complicated.

Those tools that are specifically designed for role-playing game development have two fundamental flaws:

- Those that offer an intuitive, easy-to-use interface that is friendly to newcomers in game development are locked behind a paywall which, although not very expensive, forces amateur users to pay for a software they will rarely unless they consolidate as developers.
- Those that are not locked behind a paywall, that is, open source, often have interfaces and systems that are difficult for beginners to understand, leading many to abandon game development due to the complexity of these tools.

The main motivation behind this Project arises from the need to have accessible and flexible tools, both for experimented independent developers who desire to create games without the limitations imposed by general-purpose engines, and for people with little knowledge of video game development or programming.

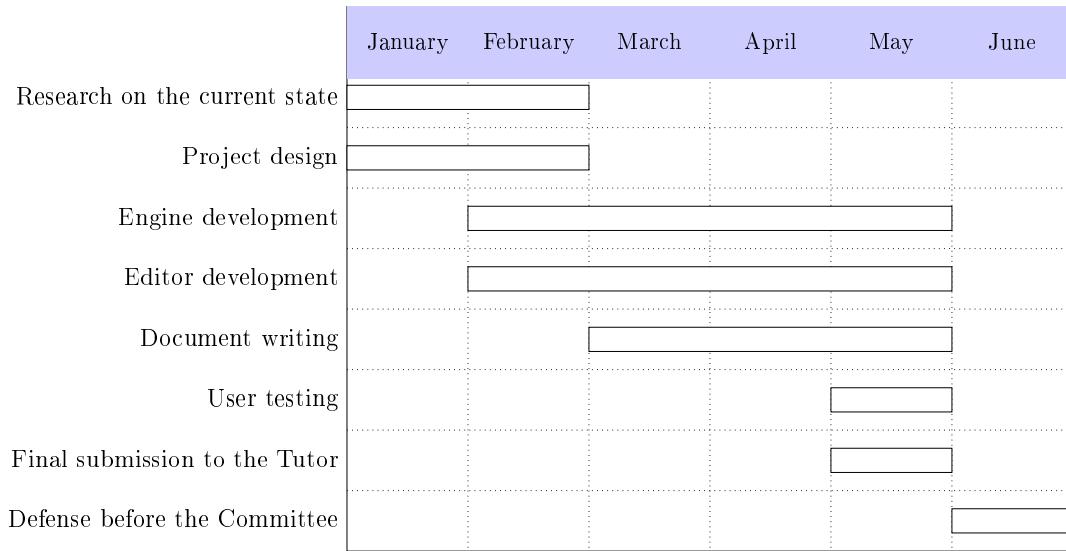


Figure 7.1: Gantt diagram showing the temporal planning of the work.

Objectives

This project aims to develop a 2D game engine specifically designed for role-playing games, along with an editor that enables simple and straightforward game development for this engine. The editor will be capable of generating executable files that users can distribute without the need for any additional steps after development.

The editor's interface will be designed for beginners in game development, while still allowing more experienced users to create larger and more ambitious projects.

To accomplish this:

- A cross-platform engine that allows the user to implement role-playing games will be developed.
- A cross-platform editor that eases the implementation of a role-playing game in the developed engine will be developed.
- The Project will be tested with users to demonstrate its functionality, and necessary conclusions will be drawn, along with potential improvements for the future.

Work Plan

To accomplish the previous objectives, the work plan will be divided in three phases:

- Research on the current state of engines and editors specifically designed for role-playing games, as well as on role-playing games themselves. This section will aim to identify the common features shared by all engines, editors and

games, both open-source and commercial ones, and propose improvements to address the issues these tools may present for inexperienced new users.

- Project Design. Based on the common features identified in the previous phase, an initial design will be proposed to serve as a basis for the development of the Project. This design, while not immutable, should be as close to final as possible to avoid any issues during the development phase.
- Project Development. Once the design phase is completed, development will begin. This phase will, in turn, be split across multiple phases:
 - Setting up the development environment. A development environment will be chosen based on the Project's needs, and everything will be configured to minimize effort during the development of the applications.
 - Engine development. An engine will be developed according to the previously established plans, with cross-platform support for both PCs and Android mobile devices.
 - Editor development. Like the engine, the editor will be developed according to the established design.
- User testing. To ensure the correct functioning of the Project, the final tools will be tested by a variety of users not involved in their development. After the tests are completed, the necessary conclusions will be drawn, and the Project will be adjusted to address any critical issues that require attention before releasing the public version. Features that would require excessive effort to implement within the available time will be left as future work.

Tools and Methodology

Regarding the tools, Git will be used as the version control system, with a repository hosted on GitHub and managed through GitHub Desktop. Task management will be carried out using GitHub Projects, available on the GitHub website.

Access to the repository containing the code may be done through this link: <https://github.com/almasso/rpgbaker>.

Regarding the development tools for the Project, CLion will be used as the C++ development environment, and Android Studio for Android development in Java. CMake will be used as the tool for generating external libraries, and MinGW for handling compilation on Windows, Clang on MacOS, and GCC on Linux. Detailed reasons for choosing these tools can be found in section 4.2.

On the other hand, PDF generation of the document will be carried out using L^AT_EX, with the T_EX^IStemplate, and Texmaker will be used as the editing environment.

Regarding the working methodology, bi-weekly meetings will be proposed with the tutor, although the number of weeks may vary depending on the progress made.

During these meetings, the current status of the Project will be presented, and advice related to the design or development of the Project will be requested.

Communication with the team will be established through both in-person meetings when important issues need to be addressed and via software that allows messaging and voice chat, such as Discord. This tool will also be used for conducting user testing.

Document Structure

In Chapter ??, *State of the Art*, the current situation of role-playing video games, video game development, and a bit of historical context of both will be discussed.

In Chapter 4, *Project Planning*, the design and decisions made prior to the development of the applications will be discussed in detail.

In Chapter 5, *Project Development*, the internal structure of the engine and editor in terms of code will be discussed, while addressing the problems encountered during this phase and the solutions proposed.

In Chapter 6, *Evaluation and Conclusions*, the research questions and objectives will be presented, along with the development of the user testing phase and an analysis of the tests, from which conclusions will be drawn.

Finally, in Chapter 7, *Future Work*, potential improvements for the final implementation of the applications that may be interest will be detailed.

Conclusions and Future Work

Conclusions and future lines of work. This chapter contains the translation of Chapter 6.

Capítulo 8

Contribuciones Personales

Miguel Curros García

Mis tareas en el proyecto se centraron en el diseño y desarrollo del motor genérico de videojuegos así como dentro del apartado de *gameplay* todo lo relacionado con el sistema de eventos, tanto a nivel de motor como de editor. Adicionalmente, también creé la gestión de persistencia con lectura y escritura de los archivos de datos específicos del editor y parte del proceso de *build*.

En primer lugar, participé en conjunto con Alejandro González en la fase de diseño del motor donde dejamos bien claras cuáles serían las distintas capas de abstracción de la implementación. Desde el principio teníamos claro que la base del motor iba a estar basada en una arquitectura EC (*Entity-Component*), construyendo el resto de las funcionalidades alrededor. Comenzamos a definir cómo estructuraríamos esa arquitectura a través de un diagrama *UML* donde definimos tanto los distintos módulos que formarían el motor como la forma en la que se conectarían las partes más básicas de este con las funcionalidades específicas de *gameplay*.

A continuación, comencé con el desarrollo de este centrándome en los apartados de carga y gestión de recursos, audio y colisiones. Comenzando con la gestión de recursos, esta es una parte crucial del motor al estar este dirigido por datos; todo aquello que se quiere que ocurra en un juego estará definido en los datos a interpretar al motor. Esta parte tiene 3 partes clave: **Resource**, **ResourceHandler** y **ResourceMemoryManager**. Cada una de esas clases se encarga de una parte clave de la carga y descarga de recursos.

- **Resource** servirá de clase base para todos los tipos de recursos que se quieran cargar, cada uno de ellos implementará cómo se carga y descarga desde los archivos de datos.
- **ResourceHandler** será a quien recurran las distintas partes del motor que necesiten cualquier tipo de recurso. A través de la ruta a un recurso de un tipo especificado dará acceso a una instancia conteniendo ese mismo recurso cargado.
- Por último, **ResourceMemoryManager** se encargará de gestionar cuánta memoria está siendo ocupada en este momento por los recursos. A partir de un

límite especificado en un archivo de configuración esta clase se encargará de que no se supere ese umbral de memoria máxima ocupada por los recursos. Si se solicitara un recurso y no hubiera memoria suficiente para cargarlo a través de un algoritmo LRU se irá liberando memoria de otros recursos hasta que haya suficiente para cargar el nuevo. Con esto, conseguí estructurar la carga de recursos bajo demanda que evitaría posibles largos tiempos de espera los juegos.

Una vez esto estaba listo desarrollé el sistema de sonido que permitiría el control sobre la reproducción de archivos de audio dentro de los juegos. Para esto decidimos usar el nuevo sistema de audio de *SDL3* que cumplía todas nuestras necesidades. Nuestro objetivo con este sistema era la implementación de un componente **AudioSource** a través del que gestionar la reproducción de un archivo de audio. Además, queríamos que cada uno de estos **AudioSource** pudieran estar asociados a un conjunto de sonidos, pudiendo tener de esta forma un control más profundo del volumen; pues es típico y útil en los videojuegos permitir al jugador controlar el volumen general, de la música o de los efectos por separado.

Para poder reproducir un sonido creé una clase **AudioClip** que se encargaría de envolver las funcionalidades de *SDL_AudioStream*; la base de la reproducción de sonidos en *SDL3*; y ofrecer una interfaz acorde a nuestras necesidades. Para poder asignar estas pistas a distintos conjuntos de sonidos implementé la clase **AudioMixer**, que tendría un control de volumen asociado que se aplicaría a cada uno de sus **AudioClip**. Una vez con estas clases básicas pude implementar **AudioSource** que las utilizaría para ofrecer su funcionalidad esperada.

Por último en la base del motor, para la detección y gestión de colisiones buscábamos algo sencillo. Por el tipo de videojuegos que ofrecemos implementar basta con comprobaciones de colisiones entre formas rectangulares. Son comprobaciones sencillas y *SDL* ya ofrece funciones para comprobar si dos rectángulos tienen intersección. Con esto, la implementación de un componente **Collider** fue sencilla. A través de una clase **CollisionManager**, en cada actualización del juego se comprobarían las colisiones entre estos **Collider**. En estas comprobaciones se guardaría la información de qué **Collider** está colisionando, acaba de empezar a colisionar o acaba de dejar de colisionar con otro. De esta manera se ofrece un control sencillo para implementar comprobaciones dependientes de colisiones entre elementos de juego.

Una vez terminamos el apartado del motor genérico comenzamos con el desarrollo de partes específicas de *GamePlay* donde centré mis aportaciones en el sistema de eventos. De cara a ofrecer una experiencia personalizable de crear un videojuego sin necesidad de programar un punto clave era nuestro sistema de eventos. Este se centra alrededor del componente **EventHandler** que sirve como conjunto de eventos de una entidad. Cada **Event** está compuesto por una condición, **EventCondition**, y comportamientos, **EventBehaviour**.

- Comenzando por las condiciones, estas se encargan de manifestar si el estado del juego es el que esperan. Para ello, cada una de las condiciones debe implementar sus propias comprobaciones, esto lo hice a través de un sistema de

herencia. Además, igual que pasa con los componentes, cada una de estas condiciones deben poder crearse a partir de los datos proporcionados por el juego, por esta razón creé una clase `EventConditionFactory`, que se encargaría de instanciar el tipo correspondiente de `EventCondition` dado un identificador.

- Los comportamientos decidimos que estuvieran implementados en *Lua*. De esta manera se podrían ampliar las funcionalidades de un juego sin necesidad de recompilar el motor, ofreciendo una experiencia más flexible. Para implementarlo de la manera que diseñamos era necesario poder tener algún tipo de POO (Programación orientada a objetos) en *Lua*, pero no es algo que ofrezca el lenguaje por defecto. A través de asignar funciones a tablas y modificando sus *metatablas* pude obtener un comportamiento similar a las clases y la herencia. A partir de esa base implementé la clase `EventBehaviour` en *C++* envolviendo a cada instancia que hubiera en una escena de los distintos comportamientos implementados en *Lua*. Además de esto, para poder implementar cada uno de los comportamientos fue necesario definir desde *C++* qué clases y cómo se podían modificar dentro de las implementaciones de estos en *Lua*.

Cuando esto estuvo listo comencé con el desarrollo del editor donde creé la gestión de persistencia con lectura y escritura de los archivos de datos específicos del editor. Estos datos decidimos que íbamos a guardarlos también en *Lua* pues nos sería más sencillo de implementar al tener ya la infraestructura montada para ello. Gracias a la clase del editor `LuaManager` creada por Alejandro Massó, que permite acceder a tablas de *Lua* de un archivo, crear nuevas y guardar estas en nuevos archivos, la tarea consistió en escribir y leer estas tablas. Cada vez que se quisiera salvar un proyecto cada uno de sus recursos guardarían sus parámetros en nuevas tablas de *Lua* que se escribirían en archivos en subdirectorios específicos dentro de un directorio «`projectfiles`» dentro de la ruta del proyecto.

A continuación desarrollé todo el apartado de Eventos, desde la creación, edición y posterior traducción a archivos preparados para ser leídos por el motor. Para este apartado lo primero que hice fue crear una nueva pestaña en el editor, la pestaña de edición de eventos. En esta se podría ver un desplegable donde escoger un evento, una sección donde se mostraría la condición del evento y otra sección donde se mostrarían los comportamientos. La parte de creación y selección de eventos fue sencilla, aprovechando que Alejandro Massó ya había implementado la creación y selección de mapas, reutilicé el código que pude para esta parte. Para hacer los apartados de la condición y los comportamientos, igual que con la condición en el motor, tuve que crear factorías que permitieran crear los distintos tipos de instancias de cada uno de estos a partir de identificadores. Esto es porque cada condición y cada comportamiento debe definir en el editor su propia clase, ya que su persistencia e interfaz gráfica difieren entre sí, de modo que requieren implementaciones distintas; además por esa misma persistencia es por lo que se necesita la factoría, cada evento puede tener cualquier tipo de condición o comportamiento y se debe poder reconstruir al abrir un proyecto guardado.

Una vez completé todas las interfaces gráficas y la persistencia de todos los tipos de condiciones y comportamientos comencé con el proceso de *build*; convertir estos

datos a la sintaxis que el motor reconoce. En el caso de las condiciones fue sencillo, pues el formato en el que guardan su información en el editor es casi idéntico a lo que necesita el motor. Lo que refiere a los comportamientos no es así, hubo dos puntos clave en este apartado: dependencias de componentes y formato de escritura. Algunos de los comportamientos en el motor funcionan por su cuenta, es decir, con que existan dentro de su `EventHandler` cumplirán su función sin problema; en cambio hay otros que necesitan otros componentes para funcionar también de modo que, para que su proceso de *build* fuera correcto necesitaban indicarle a su entidad que escribiera los componentes faltantes con los parámetros correctos. Por último, a diferencia del resto de proceso de *build* este apartado no podía hacer uso del mecanismo que usamos para escribir el resto de tablas de *Lua*. Esto ocurre porque no estamos añadiendo otra tabla al uso, se necesita la llamada a la construcción de ese `EventBehaviour` porque el mecanismo de persistencia que se usa en el resto del editor no graba las *metatablas* asociadas a cada tabla, y esto es crucial de cara al funcionamiento de los comportamientos.

Alejandro González Sánchez

Mi contribución al Proyecto se ha centrado en el diseño e implementación del motor y parte del editor. En primer lugar, realicé una investigación sobre las posibles formas de adaptar el motor para que pudiera ser multiplataforma, realizando unas primeras pruebas técnicas y de concepto, y acabé decantándome por utilizar `SDL` como núcleo central del motor, principalmente alentado por todo el soporte multiplataforma que este aporta.

Una vez concluido esto, y con una versión básica de «juego» funcional tanto en *Android* como en *desktop*, desarrollamos un pequeño programa que mostraba un rectángulo de colores en pantalla, cuyos atributos (tamaño y color) se leían de un archivo `.lua`.

A continuación, empecé junto a Miguel Curros García la fase de diseño del motor. En este paso intentamos dejar bien definidas las diferentes partes que íbamos a necesitar, así como la forma en que se comunicarían entre sí. Para ello, generamos un diagrama UML de los diferentes componentes que íbamos a utilizar y lo separamos en diferentes niveles de abstracción, dejando clara la separación entre los componentes básicos del motor y los componentes de *gameplay* que íbamos a necesitar para poder implementar todas las funcionalidades específicas que queríamos para los juegos que ofrece nuestro editor.

Una vez cerrado este diseño, comenzamos con la implementación. Yo me centré en toda la parte de `Render`, `Core` (sistema de Entidades y Componentes) e `Input`. En primer lugar, desarrollé el *core* del motor: la estructura de Entidades y Componentes, todas ellas organizadas en escenas y gestionadas por un `SceneManager`. Creé cada una de las partes del ciclo de vida de estos elementos y monté toda la estructura para que pudieran ser cargados a partir de datos leídos de un archivo *Lua*. Para esto, nos apoyamos en `ComponentFactory` y en la clase `ComponentTemplate`, que, a través de una macro y una estructura de plantilla, aceleraba mucho el proceso de declarar nuevos componentes. A continuación, creé la clase `ComponentData`, que utilizariamos

para poder parametrizar los componentes que declarásemos en Lua. Finalmente, creé unos métodos en el `SceneManager` para poder instanciar entidades y escenas a partir de unos *blueprints* creados leyendo los archivos Lua correspondientes.

Una vez terminado esto, pasamos a la parte del *renderizado*. Aquí, generalicé los métodos de pintado de `SDL` y creé un bucle de *clear*, *present* y *render*, que se llama desde el bucle principal. Para acceder a las funciones de pintado, creé una clase virtual `RenderComponent` que implementa el método `render(RenderManager*)`. A continuación, creé todos los componentes básicos de *renderizado* que íbamos a necesitar (`Camera`, `Rectangle`, `Text`, `SpriteRenderer`, `Animator`), así como los recursos que estos iban a utilizar (`Sprite`, `Animation`, `Font`, `Color`). Todo esto quedó integrado con el ciclo de vida que creé en el apartado de `Core`, para permitir su inicialización parametrizada desde Lua.

Finalmente, hice una implementación sencilla de un sistema de *input* al que se pudiera acceder desde los componentes. Dado que, por diseño, solo íbamos a utilizar *clics/touchs* para mantener de una forma más simple el soporte multiplataforma, generalicé estos en un `struct`. Adicionalmente, creé el componente `Button`, al que se le podía asignar una función de Lua que se llamaría con unos parámetros preestablecidos al detectar un *input* en su área.

Con esto y las aportaciones de Miguel, dimos por terminado el motor genérico y pasamos a implementar los elementos específicos de *gameplay*. Aquí desarrollé un sistema de diálogo formado principalmente por dos componentes: un gestor de *textboxes*, que mostraba texto poco a poco y esperaba el *input* del usuario, y unas opciones compuestas por varios botones con posibles respuestas por parte del usuario. Lo siguiente fue crear un sistema de movimiento basado en A^* , que utilizarían tanto los NPC como el jugador a partir de un input de tipo *point and click*; también añadí la opción de aplicar animaciones como parámetro a este movimiento. Una vez cerrado esto, creé un gestor de mundo cuya función sería controlar qué mapas están activos en escena en cada momento, instanciando nuevos en caso necesario a partir de los *blueprints*. Todos estos sistemas serían la base del *gameplay* de *overworld* que ofrece nuestro editor, combinado con el sistema de eventos.

Con esto listo, pasamos al desarrollo del editor, que ya tenía una base implementada por Alejandro Massó. Aquí me centré, en primer lugar, en generar un sistema de traducción que convirtiera los datos generados por el editor en datos que siguieran la estructura del motor (entidades, componentes, escenas, *sprites* y animaciones), así como en la generación de otros archivos de configuración. A partir de esto, concreté un proceso de *build* que se encargaría de obtener los binarios precompilados del motor en formato ejecutable y los combinaría con los *assets* del usuario y los archivos de datos en formato motor, previamente traducidos, para obtener el producto final: el juego, contenido en un único directorio denominado `Build`.

Posteriormente, implementé otras funcionalidades del editor, como el inspector de objetos, la pestaña de conexiones entre mapas, la pestaña de configuración del jugador y la de configuración general. Para esta última, quise implementar una previsualización para el texto del jugador, por lo que tuve que gestionar una carga asíncrona de fuentes con `DearImGui`. Todas estas funcionalidades supusieron también una ampliación del sistema de traducción y *build* previamente comentado.

Por último, me dediqué a realizar pruebas con usuarios y corregir los problemas

que encontrábamos en estas. Finalmente, adapté la funcionalidad de *build* para que también fuese posible generar una APK lista para usar en dispositivos Android.

Alejandro Massó Martínez

Mi contribución al Proyecto se ha basado en la investigación, diseño y desarrollo del editor, la creación del *toolchain* de generación de dependencias y compilación de los distintos subproyectos, y la redacción de esta memoria.

Bibliografía

*En un lugar de La Mancha
de cuyo nombre «sí quiero»
acordarme... un caballero
que antaño «enganchó y engancha»
... dejaba pasar los días
leyendo constantemente
libros de caballerías
sin dormir lo suficiente.*

Valeriano Belmonte

ANH, T. y NGUYÊN, L. T. H. Seeking Solace in the Fantastical Adventures of Dungeons and Dragons. <https://saigoneer.com/parks-and-rec/20798-seeking-solace-in-the-fantastical-adventures-of-dungeons-and-dragons-saigon> 2021. Imagen del tablero de juego de *Dungeons & Dragons* accedida desde la web Saigoneer el día 18-05-2025.

ANSELL, B., HALLIWELL, R. y PRIESTLY, R. *Warhammer Fantasy Battle*. Games Workshop. 1983.

BARTON, M. *Dungeons and Desktops: The History of Computer Role-Playing Games*. EBL-Schweitzer. CRC Press, 2008. ISBN 9781439865248.

JASON BULMAHN. *Pathfinder*. Paizo Publishing. 2009.

EDWARDS, R. Gns and other matters of role-playing theory. <http://www.indie-rpgs.com/articles/1/>, 2001. Accedido el día 17 de mayo de 2025.

FINE, G. A. *Shared Fantasy: Role-Playing Games as Social Worlds*. University of Chicago Press, 1983.

GARCÍA, R. Amazing D&D Fantasy diorama tabletop terrain. <https://volomir.com/en/dungeonsanddragons-fantasy-diorama-tabletop-terrain/>, 2016. Imagen del diorama de Ryan Devoto extraída del blog de volomir el día 18-05-2025.

GYGAX, E. G. y ARNESON, D. L. *Dungeons & Dragons*. Tactical Studies Rules. 1974.

GYGAX, E. G. y PERREN, J. *Chainmail*. Tactical Studies Rules. 1971.

- HALLIWELL, R. F., PRIESTLY, R., DAVIS, G., BAMBRA, J. y GALLAGHER, P. *Warhammer Fantasy Roleplay*. Games Workshop. 1986.
- HEINEMANN, J. How I fell in love with *Kriegsspiel*. <https://kriegsspiel.org/how-i-fell-in-love-with-kriegsspiel/>, 2022. Imagen del tablero de juego de *Kriegsspiel* accedida desde la web oficial de la Sociedad Internacional de *Kriegsspiel* (*International Kriegsspiel Society*) el día 18-05-2025.
- LORTZ, S. L. Role-Playing. *Different Worlds*, (1), páginas 36–41, 1979.
- MONTOLA, M. y STENROS, J. *Nordic Larp*. Fëa Livia, Stockholm, Sweden, 2010. ISBN 9789163378560.
- PADDINGTON, L. A Matter of Honor - a Warhammer Fantasy Battle Report. <https://frontlinegaming.org/2023/02/03/a-matter-of-honor-a-warhammer-fantasy-battle-report/>, 2023. Imagen de una partida de *Warhammer Fantasy Battle* extraída de la web *Frontline Gaming* el día 18-05-2025.
- PERRIN, S. H., TURNER, R., HENDERSON, S. y JAMES, W. *RuneQuest*. Avalon Hill. 1983.
- PETERSEN, C. S. J. y WILLIS, L. *Call of Cthulhu*. Chaosium. 1981.
- POHJOLA, M. Dragonbane Memories. <https://nordiclarp.org/2021/08/18/dragonbane-memories/>, 2021. Imagen de una partida de LARP nórdico extraída de la web *Nordic LARP* el día 18-05-2025.
- REIN-HAGEN, M. y DAVIS, G. *Mind's Eye Theatre*. White Wolf. 1993.
- VON REISSWITZ, G. L. y VON REISSWITZ, G. H. R. J. *Kriegsspiel*. 1812.
- SOLOMON, C. Dungeons & Dragons (película). 2000. Silver Pictures y New Line Cinema.
- TEKINBAS, K. y ZIMMERMAN, E. *Rules of Play: Game Design Fundamentals*. ITPro collection. MIT Press, 2003. ISBN 9780262240451.
- TYCHSEN, A., HITCHENS, M., BROLUND, T. y KAVAKLI, M. Live action role-playing games: Control, communication, storytelling, and mmorpg similarities. *Games and Culture*, vol. 1(3), páginas 252–275, 2006. ISSN 1555-4120.
- WELLS, H. G. *Little Wars*. Frank Palmer, 1913.