Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

CHIMERA: herramienta de diseño y puesta en escena de aventuras de rol

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Pastor Pérez

Tutor: Miguel Rebollo Pedruelo

2018-2019

Resumen

Tradicionalmente, los juegos de rol se han jugado siempre utilizando papel, lápiz y la imaginación de los participantes. Sin embargo, con la entrada en la era digital, empezaron a surgir varias aplicaciones que facilitaban el desarrollo de las campañas.

En este trabajo se abordará el desarrollo de una herramienta para juegos de rol de sobremesa llamada CHIMERA. La herramienta ayudará tanto a diseñar las campañas como a proporcionar facilidades durante la puesta en escena de la misma, haciendo uso de un nuevo sistema para organizar la información basado en lugares, y funcionalidades para gestionar el estado de los personajes y hacer los cálculos necesarios.

Palabras clave: juego de rol, aplicación móvil, android.

Abstract

Traditionally, role-playing games have always been played using paper, pencil and the imagination of the participants. However, with the arrival of the digital age, several applications began to emerge to facilitate the development of games.

This work address the development of a tool for table top role-playing games, called CHIMERA. The tool will help both, to design the campaigns and to provide facilities during their staging, making use of a new system to organize the information based on places and functionalities to manage the state of the characters and make the necessary calculations.

Keywords: role-playing game, mobile application, android.

Tabla de contenidos

Índice de figuras

Índice de Tablas

1. Introducción

Los juegos de rol de sobremesa llevan con nosotros desde principios de los años setenta[1], ayudándonos a vivir aventuras y ponernos en la piel de todo tipo de personajes. Pero al contrario de lo que se pudiera pensar, los juegos de rol están en auge, sin ir más lejos, Dungeons & Dragons[2] (el juego de rol de sobremesa más famoso) registró en 2017 su mejor año en término de ventas[3].

Lo único que se necesita para jugar una campaña de rol es: una persona que se encargue de diseñar y dirigir la historia; un grupo de personas que interpreten los personajes jugadores y un lugar donde jugar. Sin embargo, una aplicación que nos ayude a gestionar la campaña hace que podamos centrarnos más en jugar y menos en preparar.

1.1. Motivación

Cuando se decide jugar una campaña de rol, es necesario que una persona prepare y dirija la historia. La cantidad de información que se prepara varía mucho dependiendo de la persona; hay quienes apuntan cada detalle y otros que confían más en la improvisación, pero al menos siempre habrá que disponer de algún tipo de guía para la aventura o algún apunte sobre las características de los enemigos. Además, hay que llevar la cuenta de muchas variables durante la campaña (puntos de vida, flechas, oro, etc.) que muchas veces puede ralentizar el desempeño de la misma.

En cuanto a la tecnología, es indudable que los dispositivos móviles están en auge[4], y su portabilidad hace que sean la plataforma idónea para desarrollar una aplicación que agilice las campañas. Además, personalmente no se había desarrollado ninguna aplicación en Android[5], por lo que resultó ser una oportunidad de oro para aprender una de las tecnologías más punteras de la actualidad. Esta situación, nos llevó a querer desarrollar una herramienta para agilizar todo el proceso, así surge Companion for Humans Intending to Master Extreme Role Adventures, o para abreviar CHIMERA.

1.2. Objetivos

1.2.1. Objetivo general

Como se ha mencionado anteriormente, el objetivo principal es el desarrollo de una aplicación móvil que ayude a agilizar las sesiones de rol.

1.2.2. Objetivos específicos

Para lograr el objetivo principal se establecieron los siguientes objetivos específicos:

- Organizar la información en un mapa de eventos.
- Facilitar la gestión del estado de los personajes.
- Automatizar los cálculos del combate.

1.3. Impacto esperado

Se identificaron tres tipos de usuarios sobre los que tiene impacto la aplicación: jugador, director y diseñador. Además, también tiene impacto sobre dos de los diecisiete objetivos para el desarrollo sostenible.

1.3.1. Jugador

Las personas que hagan el rol de jugador son capaces de mantener actualizada la información de su personaje sin tener que borrar y escribir en una ficha en papel (p.ej. flechas, oro, magia o energía).

1.3.2. Director

La persona que tenga el rol de director es capaz de tener toda la información organizada en un mapa de eventos y de ver y modificar el estado de los NPC (Personajes No Jugadores). Cada evento puede tener vinculados los combates y los personajes que participan en dicho evento para así tener la información accesible. También es capaz de resolver rápidamente los combates.

1.3.3. Diseñador

La persona que diseñe la campaña (que suele ser la misma que el director) puede crear ubicaciones en el mapa donde guardar la información sobre lo que pasa en cada lugar de un modo intuitivo, así como preparar de antemano NPC y combates.

1.3.4. Objetivos para el Desarrollo Sostenible

A parte de los usuarios, la aplicación también tiene impacto sobre los Objetivos de Desarrollo Sostenible[7] planteados por la ONU. El primero es en el objetivo 15, Vida de Ecosistemas Terrestres, ya que, al digitalizar las fichas de personaje y los apuntes utilizados para dirigir la partida, reducimos considerablemente el impacto sobre la deforestación que correspondía a los juegos de rol.

Por otro lado, tiene un impacto en el objetivo 4, Educación de Calidad, ya que los juegos de rol aportan numerosos beneficios a la educación[6], y la digitalización de estos hará más fácil incorporarlos a las actividades en las escuelas. Entre los beneficios que aportan, los más destacables son: mejora del cálculo mental, adquisición de riqueza expresiva, soltura a la hora de tomar notas y esquematizar, estímulo del potencial creativo, trabajo en equipo, capacidad de resolución de problemas e improvisación, empatía y tolerancia.

1.4. Metodología

En cuanto a la metodología, se ha empleado Scrum[8], siguiendo un desarrollo incremental en *sprints* de dos semanas. El procedimiento ha sido el siguiente:

1. El primer paso es realizar un análisis de las funcionalidades que debe tener la aplicación.
2. En segundo lugar, se prepara el primer *sprint* determinando que se va a realizar en las primeras dos semanas.
3. Una vez está listo el *sprint* se procede a diseñar e implementar las funcionalidades que están dentro del *sprint*.
4. Antes de cerrar el sprint, se prueban todas las funcionalidades en conjunto para asegurarse de que esta todo en orden.
5. Cuando acaba el *sprint* se realiza una revisión con los usuarios potenciales para que prueben las funcionalidades implementadas y aporten retroalimentación.
6. A partir de la retroalimentación de los usuarios, se registran las correcciones de errores, modificaciones y nuevas funcionalidades que deberían implementarse en los siguientes *sprints,* y se priorizan con el resto de las tareas para decidir que entra en el siguiente *sprint*.
7. Con las tareas ya priorizadas, se repite el ciclo desde el paso dos. Este ciclo se repetirá hasta que la aplicación se haya completado.

Es importante puntualizar que las tareas que conforman el *sprint* deben tener como resultado un mínimo producto viable para que pueda ser probado por los usuarios.

1.5. Estructura

En los siguientes puntos se expone todo el proceso de desarrollo de CHIMERA. En el punto dos se analiza el estado del arte, donde se exponen las aplicaciones existentes para jugar a juegos de rol y una comparativa entre ellas. En el punto tres se realiza un análisis más a fondo del problema, donde se incluyen los casos de uso, riesgos y demás. En el punto cuatro se presentan los diseños de la aplicación (diagramas de clases, relacionales, de secuencia…); En el punto cinco se encuentra redactado el desarrollo de la solución, con las particularidades y problemas que ha habido. En el punto seis se expone un apartado de pruebas, tanto las unitarias para probar el código como las realizadas con los usuarios; en el punto siete se encuentran las conclusiones obtenidas y el punto ocho se trata de un apartado con las nuevas funcionalidades y mejoras que han surgido durante el desarrollo y que se prevé implementar en futuras versiones.

Adicionalmente, en el punto nueve se encuentra un apartado de referencias, donde están recogidas las referencias bibliográficas y al final del documento un apartado de glosario, en el que se recogen las palabras que pueden prestar a confusión.

2. Estado del Arte

Como se mencionaba anteriormente, los juegos de rol nacieron con Dungeons & Dragons (D&D) en 1974, convirtiéndose en el primer juego de rol en ser comercializado[1]. Esto produjo una revolución en el ámbito de los juegos de mesa, los juegos de rol eran los primeros en los que el objetivo ya no era enfrentarse al resto de jugadores, sino cooperar para conseguir un bien común.

A partir de entonces fueron apareciendo nuevos juegos de rol y D&D se fue refinando, dando lugar a una comunidad inmensa de jugadores. Al principio se centraban más en la creación del personaje en sí, pero alrededor de 1984 empezaron a poner el foco en la narración. Ya no solo se trataba de hacer un personaje sino de contar una historia.

2.1. ¿Cómo es una sesión de juego?

La experiencia de juego varía mucho dependiendo del sistema que se utilice, sin embargo, la mayoría de los juegos de rol se tienen los siguientes elementos en común:

Descripciones del entorno: Quien dirija la partida deberá describir la situación en la que se encuentran los participantes para que así puedan actuar en consecuencia.

Improvisación: Una vez descrita la situación en la que se encuentran, los jugadores deberán actuar como consideren oportuno, y declarar las acciones que quieren realizar.

Fichas de personaje: La forma más habitual de representar un personaje es mediante una ficha de personaje. Estas fichas representan tanto a los personajes jugadores como a los NPC y varían mucho dependiendo del sistema de rol, los hay más simples como D&D que solo necesitan una o dos y más complejos como Anima: Beyond Fantasy[9] que requieren cuatro o cinco. Sin embargo, esencialmente son una serie de hojas de papel en la que se representan las capacidades de los personajes de forma numérica (salud, reflejos, puntería, habilidad de costura…).

Tiradas de dados: La mayoría de las acciones que realicen los jugadores tendrán siempre algún factor de azar. Ni el mejor arquero puede acertar siempre en cualquier situación, ni el mejor cocinero preparará siempre el mejor plato posible; por ello, es necesario representar la suerte de alguna manera. Aquí es donde intervienen los dados. Los dados varían mucho dependiendo del juego de rol; en Anima por ejemplo solo se requieren dos dados de diez caras, mientras que en D&D necesitas siete dados diferentes. En cualquier caso, en casi todos los juegos de rol es necesario tirar algún tipo de dado y sumarlo a la propia habilidad del personaje para determinar si tiene éxito o no en la acción que deseaba realizar. De esta manera siempre habrá un factor de incertidumbre que ayuda a generar tensión en las partidas.

Tablero y figuras: No suele ser necesario tener un tablero y miniaturas para jugar una campaña, sin embargo, resultan de gran utilidad para visualizar la situación. Suelen utilizarse sobre todo para representar combates.

2.2. Nuevas tecnologías

A medida que se fueron creando juegos nuevos y se fueron refinando los ya existentes, la complejidad y la necesidad de gestionar la información necesaria fue aumentando. En un principio se escribía toda la información que había que llevar a la sesión en hojas de papel, sin embargo, con la entrada en la era de la información se empezaron a crear aplicaciones que ayudaban tanto a jugadores como a directores en algunos aspectos del juego. Algunos de los tipos de aplicaciones[8] que existen son los siguientes:

- **Lanzadores de dados:** Son aplicaciones que, como su nombre indica, sirven para realizar tiradas de dados sin necesidad de tener unos dados físicos. Algunos también permiten guardar configuraciones de tirada, lanzar varios dados a la vez o incluso crear macros. Algunos ejemplos son: RPG Simple Dice[32], Legend of the Five Rings[33] o DnD Dice[34].
- **Fichas:** Hoy en día hay muchas aplicaciones que permiten guardar las fichas de los personajes en el móvil. Cada juego de rol tiene una ficha diferente, pero ya hay muchos que aceptan varios tipos de ficha. Algunos ejemplos son: Squire[20], Fith Edition Character Sheet[21], Fight Club 5th Edition[22].
- **Generadores de contenido:** Existen también aplicaciones que generan automáticamente nombres, monstruos, mazmorras, etc. facilitando mucho la labor del director de juego. Este tipo de aplicaciones ahorran mucho tiempo del diseño de la campaña, permitiendo al director centrarse en la trama. Algunos ejemplos son: FaNG[23], Dungeon Map Generator[24], Monster Factory[25].
- **Gestores de personajes:** Este tipo de aplicaciones es el más variado, engloba todas aquellas aplicaciones que les sirven a los jugadores para llevar la cuenta del estado de su personaje. Desde las que sirven para saber lo que tiene en el inventario hasta las que hacen de grimorio. Algunos ejemplos son: Bag +5[26], Character Story Planner[27], Anima Magic Cards[28].
- **Ayudas interpretativas:** Estas sirven como ayuda a los directores y jugadores a la hora de dar vida a los personajes. Las más comunes son las librerías de sonidos e imágenes, pero también las hay que generan formas épicas de narrar un golpe o incluso insultos élficos. Algunos ejemplos son: Cthulhu Sounds[29], Fantasy Insult Generator[30], Fantasy Soundboard [30].
- **Aplicaciones completas:** Por último, quedan las que, en una sola aplicación, realizan varias de las funcionalidades de las aplicaciones mencionadas anteriormente y algunas nuevas. Estas aplicaciones no son tan abundantes debido a su complejidad, y se mueven en un entorno más profesional que las

anteriores. Nuestra aplicación se encontrará en este grupo, y a continuación se realizará una comparativa con las más importantes que hay en el mercado actualmente.

2.3. Crítica al estado del arte

Hoy en día, hay diversas aplicaciones que nos facilitan varias de las tareas que se necesitan para jugar una campaña de rol. Aunque todas tienen una gran variedad de funcionalidades, ninguna es capaz de hacerlo todo y cada una está centrada en un público diferente. Por ello, CHIMERA rellena un hueco que han dejado el resto de las aplicaciones. A continuación, se procederá a analizar las características de las cinco que más similitudes tienen con nuestra aplicación:

2.3.1. D&D Beyond

Compatibilidad: Dungeons & Dragons 5
Modo de juego: En persona
Plataforma: Web
Fecha de lanzamiento: 15/8/2015

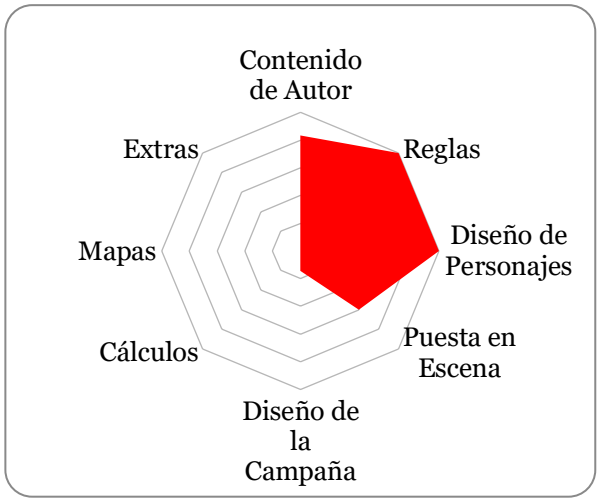

Figura 1 Gráfico D&D Beyond – Elaboración propia

D&D Beyond[11] es una aplicación que se define a sí misma como un set de herramientas para la quinta edición de Dungeons & Dragons. Se trata de una página web que nos permite con facilidad crear contenido para D&D (personajes, trasfondos, objetos mágicos, etc.) y además nos proporciona la posibilidad de comprar los manuales oficiales para tenerlos siempre disponibles. Los manuales es una de las funcionalidades más importantes de la aplicación, puesto que permite compartirlos con otros miembros de la campaña sin necesidad de que estos también lo compren. Además, también permite comprarlos por secciones, para que no sea necesario comprar todo el manual si solo estamos interesados en una parte.

No obstante, hay tres aspectos importantes que no cubre. El primero es la creación de campañas, la única facilidad que aporta es un tosco editor de texto que es mucho más inconveniente que escribir en papel. El segundo es que no proporciona ninguna facilidad para hacer cálculos ni realizar tiradas de dados. Y el tercero es la disponibilidad sin conexión, ya que al tratarse de una página web no se puede utilizar sin conexión a internet.

2.3.2. Roll20

Compatibilidad: Cualquiera
Modo de juego: Online
Plataforma: Web
Fecha de lanzamiento: 17/9/2012

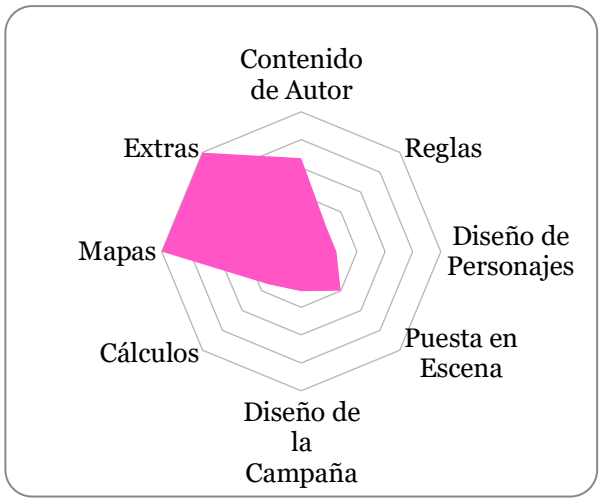

Figura 2 Gráfico Roll20 – Elaboración propia

Roll20[12] también es una página web, aunque esta se centra en proporcionar a sus usuarios facilidades para jugar en red. Esto lo consigue mediante la capacidad de realizar llamadas de voz, publicar las fechas de las sesiones, crear mapas, etc. La creación de mapas es su mayor fortaleza, puesto que pone a disposición de los usuarios una gran cantidad de herramientas para que puedan crear sus propios escenarios. Además, durante la sesión permite que cada jugador mueva su avatar en el mapa como si fuera un tablero físico.

Sin embargo, al contrario que D&D Beyond, esta no facilita la creación de personajes ni por supuesto la del resto de extras. Tampoco proporciona muchas facilidades a la hora de llevar la cuenta del estado de tu personaje y al ser también una página web, necesita conexión a internet para ser usada.

2.3.3. Fantasy Grounds

Compatibilidad: +50 juegos diferentes (D&D, Numenera, Pathfinder, etc.)
Modo de juego: Online
Plataforma: Escritorio (Windows y Mac)
Fecha de lanzamiento: 1/5/2004

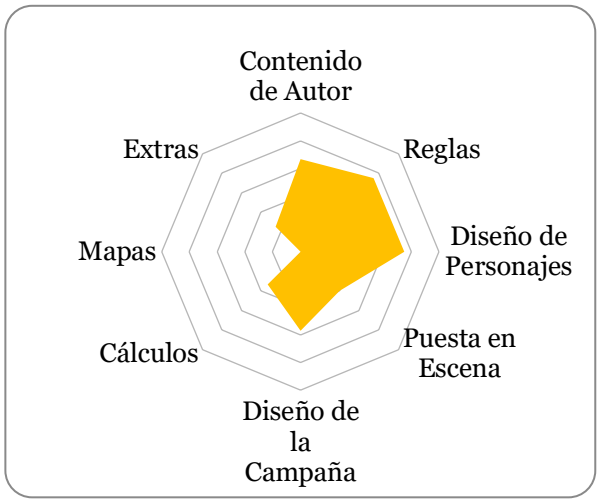

Figura 3 Gráfico Fantasy Grounds – Elaboración propia

Fantasy Grounds[13] es una aplicación de escritorio, por lo que los usuarios pueden utilizarla sin conexión a internet. Al igual que en D&D Beyond, nos permite comprar manuales y también conecta a los jugadores como en Roll20.

Sin embargo, las funcionalidades anteriormente mencionadas no están tan desarrolladas como en las dos aplicaciones anteriores, y sigue manteniéndose el problema de la creación de campañas.

2.3.4. Astral

Compatibilidad: D&D
Modo de juego: En persona y Online
Plataforma: Web
Fecha de lanzamiento: 19/6/2017

Nuevamente, Astral[14] se trata de una aplicación web. Debido a que es posterior a Roll20, tiene muchas de las funcionalidades en línea de esta última, haciéndola una competidora directa.

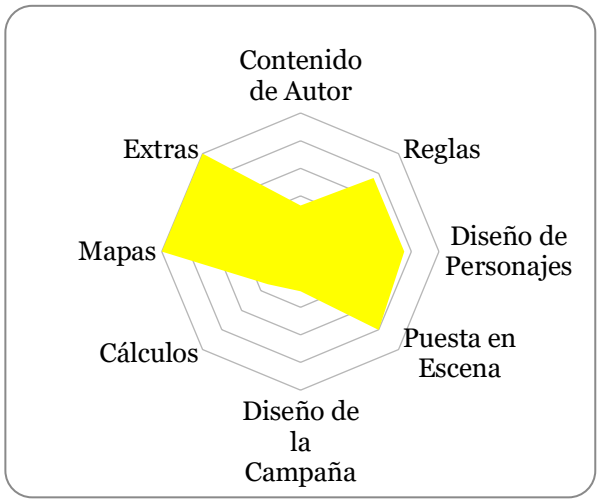

Figura 4 Gráfico Astral – Elaboración propia

Pero al igual que Roll20, aunque sí que facilita la creación de personajes, no da facilidades para crear el resto de los extras que permitía D&D Beyond. Además, presenta el mismo problema que los anteriores al crear una campaña.

2.3.5. Anima: Master Toolkit

Compatibilidad: Anima
Modo de juego: En persona
Plataforma: Android
Fecha de lanzamiento: Acceso beta desde 10/8/2018

Difiriendo del resto de las aplicaciones de la comparativa, Anima: Master Toolkit[15] es una aplicación para dispositivos móviles, lo que proporciona una portabilidad excelente a la hora de jugar en persona. Su característica estrella es la resolución de combates que, aunque solo resuelva combates simples, agiliza enormemente los enfrentamientos.

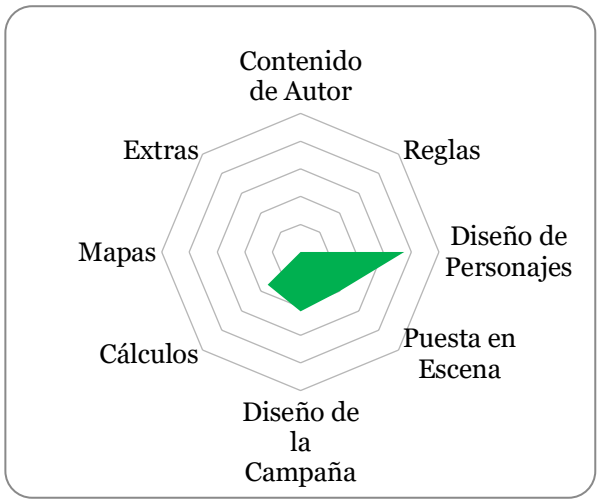

Figura 5 Gráfico Anima Master Toolkit – Elaboración propia

No obstante, no está pensado para ser usado en tabletas y tiene los mismos problemas que el resto en la creación de campañas.

2.4. Propuesta

2.4.1. CHIMERA

Compatibilidad: Anima
Modo de juego: En persona
Plataforma: Android

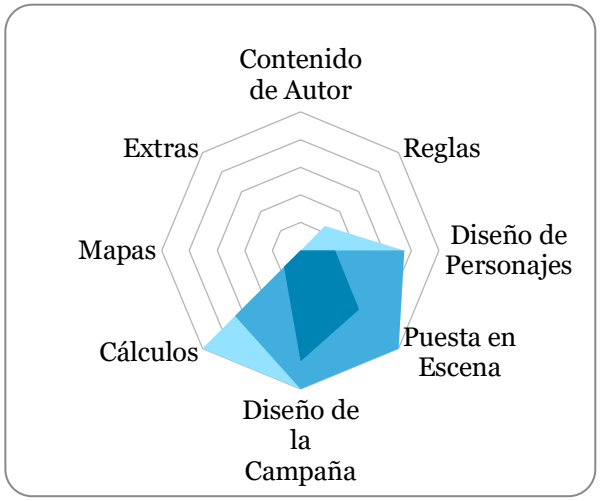

Figura 6 Gráfico CHIMERA – Elaboración propia

Como se ha podido observar en la comparativa y como se puede ver en la Tabla 1, la creación de campañas es un aspecto en el que todas las aplicaciones fallan de una manera u otra y la gestión del estado de los personajes también es un aspecto que no se suele desarrollar. Además, el cálculo automático del combate casi no se ha explorado. Por ello, estos tres aspectos son los pilares principales de CHIMERA. El objetivo es desarrollar una aplicación que permita organizar la información para que sea fácilmente accesible y modificable y que proporcione ayudas durante el desarrollo de la campaña.

Diseño de la campaña

Para poder jugar a cualquier juego de rol, un director de juego debe encargarse de crear la historia que van a jugar los jugadores. Esto tradicionalmente se consigue redactando un guion con una serie de apuntes, notas, tablas y cualquier otra información que el director de juego vaya a necesitar durante la puesta en escena de la campaña. Sin embargo, esta manera de hacerlo supone un problema, y es que la información se encuentra dispuesta secuencialmente, por tanto, si los jugadores no realizan las acciones que espera el director en el orden que había pensado, se ve obligado a saltar de una página a otra para encontrar lo que sucede en cada momento. Además, al crear las campañas de forma secuencial, el director a menudo no contempla situaciones inesperadas que pueden surgir si los jugadores no siguen el orden planificado.

Como se ha visto en la comparativa, no hay ninguna aplicación que resuelva este problema. Anima: Master Toolkit plantea una solución mediante su sistema de lugares, sin embargo, se trata simplemente de una lista con los nombres de los lugares como cabecera, dejando la información dispuesta de manera secuencial. Roll20 también propone una solución similar con su línea de tiempo, pero nuevamente, al ser una línea sigue siendo secuencial.

Por ello CHIMERA hace uso de un mapa de eventos en el que se guarda la información de lo que pasa en cada lugar. En las campañas muchas veces los jugadores no tienen un

objetivo fijo, sino que tienen que actuar por su cuenta, haciendo muy difícil saber que van a visitar primero, por ello, al guardar la información en puntos de un mapa, es más fácil de localizar que en formato texto.

Estado de los personajes

Una vez que la historia está preparada, ya se puede organizar una sesión para jugarla. Durante el transcurso de la campaña, los jugadores realizan muchas acciones; por ejemplo, comerse una de sus raciones de viaje, disparar una flecha, vender una espada, lanzar un hechizo, recibir daño y un largo etcétera.

Muchas de estas acciones requieren que llevemos la cuenta de diversas variables. Siguiendo el ejemplo anterior de la flecha; supongamos que tenemos un personaje que hace uso de un arco; este personaje durante un combate va gastando sus flechas cada vez que realiza un ataque; tradicionalmente para llevar la cuenta tendríamos una hoja de papel donde pone las flechas que tenemos, y cada vez que cambiara la cantidad de flechas, borraríamos dicho número y pondríamos el nuevo valor.

Esta manera de llevar la cuenta es bastante ineficiente, ya que la sesión se ralentiza mucho al tener que borrar y escribir cada vez que se gasta algún tipo de consumible. Es por eso por lo que CHIMERA permitirá, tanto a los jugadores como al director, modificar el estado de sus personajes, para evitar acabar borrando una y otra vez una hoja de papel.

Cálculo automático del combate

En la mayoría de los juegos de rol, el combate es uno de los aspectos que más carga reglas tiene, ocupando buena parte del manual. Dependiendo del juego, el combate puede ser muy variado; desde los más simples como Savage Worlds[16], hasta los más complejos como Anima: Beyond Fantasy. En cualquier caso, suele ser un elemento que requiere tiempo, ya que al contrario que en los videojuegos, en los juegos de rol tradicionales los cálculos tienen que hacerlos las personas y no una máquina.

Este problema lo soluciona en parte Anima: Master Toolkit con su sistema para calcular el resultado de un combate, sin embargo, solo resuelve combates simples. CHIMERA dispondrá de un apartado de combate en el que se resolverán los ataques automáticamente tanto simples como en area. Esta funcionalidad depende mucho del juego de rol al que se juegue, por lo que esta primera versión tendrá como objetivo el sistema de Anima: Beyond Fantasy, que se caracteriza por ser especialmente complejo.

Como se puede observar en la Tabla 1, las funcionalidades se han dividido según el método MoSCoW[17] que se explica más en detalle en el punto 3. Los requisitos identificados como *Must* tendrán fondo azul oscuro, los identificados como *Should* serán azul celeste, los identificados como *Could* serán azul claro y los *Wouldn't* tendrán un fondo de color rojo; este código de color es el mismo que el usado en el gráfico de radar de la Figura 6. A continuación, se muestra la tabla comparativa:

	Aplicaciones					
Funcionalidades	DDB	Roll 20	FG	Astral	AMT	CHIMERA
CAR1: Reglas						
CAR1.1: Manuales	Si	Si	Si	Si	No	Co
CAR1.2: Crear Compendios	Si	No	Si	Si	No	W
CAR1.3: Compartir manuales	Si	No	Si	Si	No	W
CAR1.4: Comprar manuales parciales	Si	No	No	No	No	W
CAR2: Diseño de Personajes						
CAR2.1: Imágenes para los personajes	Si	Si	Si	Si	Si	S
CAR2.2: Gestión de personajes	Si	No	Si	Si	Si	Mo
CAR2.3: Gestión de hechizos/habilidades	Si	No	Si	Si	No	S
CAR2.4: Explicación de reglas	Si	No	No	No	Si	W
CAR3: Diseño de la Campaña						
CAR3.1: Redacción del guion	Si	No	Si	No	Si	Mo
CAR3.2: Gestión de combates	No	Si	Si	Si	No	Mo
CAR3.3: Gestión de NPC	No	Si	Si	Si	Si	Mo
CAR3.4: Creación de controles de habilidad	No	Macros	Si	No	Si	S
CAR3.5: Árbol de caminos/encuentros	No	Lineal	No	No	Lineal	Mo
CAR4: Puesta en Escena						
CAR4.1: Info. del estado de los personajes	Si	Si	Si	Si	Si	Mo
CAR4.2: Notas públicas para los jugadores	Si	Si	Si	Si	No	S
CAR4.3: Gestión de consumibles	No	No	No	Si	Algunos	Mo
CAR4.4: Reproducción de música	Twitch	No	No	Si	No	S
CAR4.5: Curar/dañar a un personaje	Si	No	No	No	Si	Mo
CAR5: Cálculos						
CAR5.1: Cálculo de iniciativa	No	Macros	Si	Si	Si	S
CAR5.2: Generador de tiradas	No	Si	Si	No	Si	S
CAR5.3: Resolución de ataques	No	No	No	No	Simples	Mo
CAR5.4: Ataques en área	No	No	No	No	No	S
CAR5.5: Gasto de consumibles automático	No	Macros	No	Macros	No	Co
CAR5.6: Macros	No	Si	No	Si	No	Co
CAR6: Mapas						
CAR6.1: Creación de mapas	No	Si	No	Si	No	W
CAR6.2: Iluminación dinámica	No	Si	No	Si	No	W
CAR7: Contenido de Autor						
CAR7.1: Marketplace de la comunidad	No	Si	No	Si	No	W
CAR7.2: Reglas adicionales	Si	No	Si	No	No	W
CAR7.3: Monstruos	Si	Si	Si	No	No	W
CAR7.4: Personajes	Si	No	No	No	No	W
CAR7.5: Imágenes	Si	Si	Si	Si	No	W
CAR7.6: Campañas	Si	Si	Si	No	No	W
CAR8: Extras						
CAR8.1: Chat	No	Si	Si	Si	No	W
CAR8.2: Llamada	Twitch	Si	No	Si	No	W
CAR8.3: Organización de las sesiones	No	Si	No	Si	No	W
CAR8.4: Almacenamiento en la nube	No	Si	No	Si	No	W

Tabla 1 Comparativa de aplicaciones – Elaboración propia

3. Análisis del problema

3.1. Identificación y análisis de soluciones posibles

Como es lógico, había varias decisiones que tomar a la hora de diseñar la aplicación. A continuación, se expondrán las más críticas:

3.1.1. Modo de Juego

Con la llegada de la era de la información dejó de ser imposible jugar a los juegos de rol sin estar presente; tradicionalmente siempre se había jugado en persona, pero las videollamadas permitieron jugar a través de la red. A raíz de esto, surgieron aplicaciones que se orientaban a esta modalidad de juego.

Sin embargo, nosotros no hemos orientado la aplicación hacia esta modalidad, y es lógico; hoy en día las aplicaciones que se centran en la jugabilidad en línea ya tienen un público consolidado que es poco probable que quiera cambiar de aplicación, sobre todo los que han comprado manuales y otro contenido. Además, de esta manera, nuestra aplicación no es un sustituto de las que ya existen, sino que puede ser utilizada en conjunto con el resto.

3.1.2. Plataforma

La elección de la plataforma fue posiblemente la más difícil de las que se tomaron en el desarrollo de la aplicación.

Por una parte, desarrollarla para la web hubiera sido la mejor opción en cuanto a compatibilidad entre sistemas operativos, y además permite a los usuarios no tener la aplicación descargada para poder usarla. Sin embargo, esto acarrea varios problemas ya que es necesario tener conexión a internet, la cual no siempre está disponible cuando se juega en persona. También supone un problema para el almacenamiento, ya que sería necesario un sistema de almacenamiento en la nube que muchos usuarios prefieren ahorrarse pagar. Además, la implantación de la aplicación hubiera sido más compleja que en el resto de las plataformas.

Por otra parte, desarrollarla para escritorio hubiera sido una buena opción para solucionar el problema de la conexión a internet y el almacenamiento. Sin embargo, sería complicado que todos los jugadores la utilizaran, ya que en una sesión en persona el único que dispone de un ordenador suele ser el director.

Al final se optó por Android, ya que, aunque la creación de campañas sea menos cómoda en un dispositivo móvil que en un ordenador, no es necesario disponer de conexión a internet y usando la memoria interna o externa del móvil no hay problemas de almacenamiento. Además, al estar la mayoría de las aplicaciones de la comparativa desarrolladas en plataformas diferentes a Android, resulta fácil que la usemos en conjunto con el resto sin necesidad de pantallas adicionales o grandes pantallas.

A continuación, se muestra una tabla comparativa:

Plataforma	Ventajas	Desventajas
Web	• Compatibilidad entre sistemas operativos • No es necesario descargar la aplicación • Comodidad a la hora de redactar la historia	• Necesidad de conexión a internet • Almacenamiento de pago • Dificultad de uso en conjunto con otras aplicaciones • Implantación compleja
Escritorio	• Sin necesidad de conexión a internet • Almacenamiento gratuito • Comodidad a la hora de redactar la historia	• Necesidad de que cada persona disponga de su ordenador • Baja compatibilidad entre sistemas operativos • Es necesario descargar la aplicación • Dificultad de uso en conjunto con otras aplicaciones
Android	• Facilidad de uso en conjunto con otras aplicaciones • Sin necesidad de conexión a internet • Almacenamiento gratuito	• Baja compatibilidad entre sistemas operativos • Es necesario descargar la aplicación • Incomodidad a la hora de redactar la historia

Tabla 2 Comparativa plataformas – Elaboración propia

3.1.3. Compatibilidad

La compatibilidad también resultó un aspecto clave de la aplicación. En un principio se tenía pensado que fuera una herramienta para cualquier juego ya que así tendríamos más usuarios que probaran la aplicación, sin embargo, eso supondría un tiempo de desarrollo muy superior, y no dispondríamos de la retroalimentación de los usuarios tan pronto, lo que le quitaría sentido al concepto de mínimo producto viable. Por ello se decidió implementar en esta primera versión, el sistema de combate de Anima: Beyond Fantasy, ya que es un juego con un sistema muy complicado que aprovecharía mucho la funcionalidad.

3.2. Solución propuesta

Por todo lo expuesto en el punto anterior, se decidió desarrollar una aplicación en Android orientada a las sesiones en persona, que se centrara en Anima. Para determinar que funcionalidades serían las más relevantes se utilizó la metodología MoSCoW.

Esta metodología divide las funcionalidades en cuatro categorías:

- ***Must Have* (Mo):** Funcionalidades indispensables.
- ***Should Have* (S):** Funcionalidades que son importantes pero que no llegan a ser críticas o que pueden dejarse para más adelante.
- ***Could Have* (Co):** Funcionalidades que son deseables pero que no son necesarias.
- ***Won't Have* (W):** Funcionalidades que se han determinado como las menos críticas o no aplicables. Estas funcionalidades no se planea introducirlas en el proyecto, aunque en un futuro se pueden reconsiderar y subir a una de las anteriores.

Debido a que la duración de un TFG es limitada, el alcance de este TFG cubre únicamente las funcionalidades catalogadas como "Must Have" y algunas de las catalogadas como "Should Have", que se decidieron implementar como resultado de la retroalimentación de los usuarios. Dichas funcionalidades se agruparán por categorías y se detallarán en siguiente apartado.

3.3. Especificación de requisitos

Es importante mencionar que el desarrollo de la aplicación se ha realizado en inglés, por lo que todos los diagramas, clases y esquemas tendrán los nombres de sus elementos en inglés. Para que no haya confusión con los términos, a continuación se muestran las equivalencias y una breve descripción:

Castellano	Inglés	Descripción
Campaña	Campaign	Es la historia que será jugada durante la sesión.
Personaje	Character	Son los seres que aparecerán en la campaña. Pueden representar tanto a personajes jugadores como a NPC.
Combate	Combat	Representa un enfrentamiento. Los personajes jugadores suelen tener que combatir contra algún enemigo para progresar en la historia.
Consumible	Consumable	Representa cualquier elemento que se pueda gastar durante la partida: flechas, oro, vida, etc.
Evento	Event	Representa un suceso concreto de la campaña. Se utiliza para disponer el guion de manera no lineal.
Director	Master	Es el rol que adopta quien dirige la sesión. Se encarga de describir las situaciones y de interpretar a los NPC. Es uno de los actores de la aplicación.
Jugador	Player	Es el rol que adoptan las personas que juegan la campaña. Cada uno deberá interpretar a su personaje (o personajes). Es uno de los actores de la aplicación.
Diseñador	Designer	Es el rol que adopta quien diseña la campaña. Se encarga de pensar la historia, balancear los combates, crear puzles, etc. Es uno de los actores de la aplicación.

Tabla 3 Términos de la aplicación – Elaboración propia

3.3.1. Requisitos funcionales

A continuación, a partir de las características analizadas en la comparativa del punto dos, se procede a describir los requisitos funcionales:

RF1 - Gestión de personajes: Este requisito engloba las características “Gestión de personajes”, “Info. Del estado de los personajes” y “Gestión de NPC” que se encuentran en la Tabla 1. Comprende las operaciones típicas de creación, modificación, consulta y borrado de los personajes (operaciones CRUD).

RF2 - Gestión de la campaña: La gestión de la campaña incluye a la característica “Redacción del guion”. Al igual que la gestión de personajes, también comprende las operaciones de creación, modificación, consulta y borrado, aunque esta vez de las campañas.

RF3 - Gestión de eventos: Este requisito engloba las características “Árbol de caminos/encuentros” y “Redacción del guion”, aunque en este caso solo la parte del guion relacionada con el evento. Además de las operaciones CRUD como el resto de las características, también comprende la vinculación de personajes y combates.

RF4 - Gestión de combates: La gestión de combates engloba las características “Gestión de combates”, “Curar/dañar a un personaje” y “Resolución de ataques”. Nuevamente comprende a las operaciones CRUD y en este caso también comprende los cálculos que se realizan en el combate y la vinculación de personajes.

RF5 - Gestión de consumibles: Por último, la gestión de consumibles incluye la característica "Gestión de consumibles" y comprende las operaciones CRUD de los consumibles.

Para tener una visión global de la aplicación, se realizó el siguiente diagrama de casos de uso con los casos de uso de alto nivel.

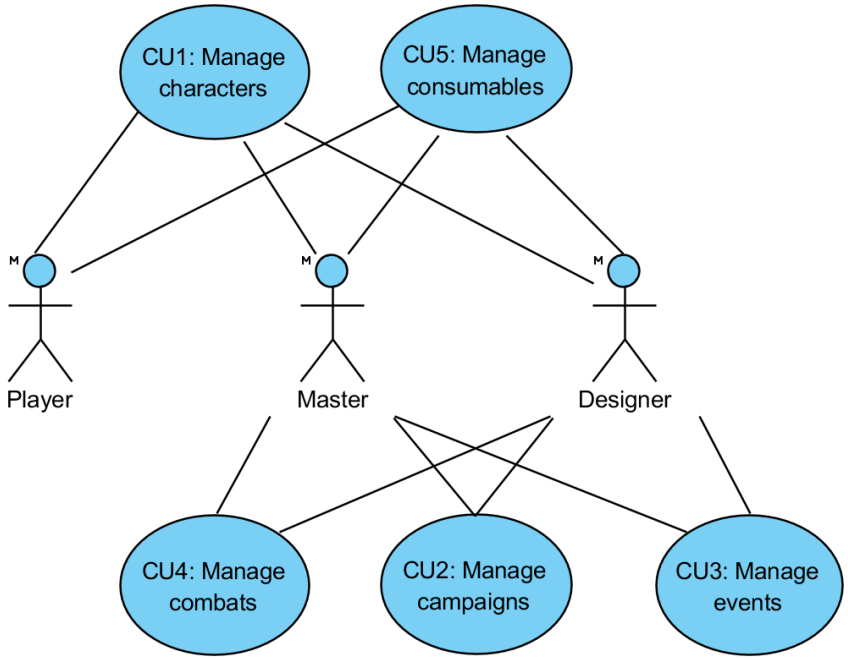

Figura 7 Casos de uso generales – Elaboración propia

Características	Requisitos funcionales	Casos de uso
CAR 2.2, CAR 3.3, CAR4.1	RF1	CU1
CAR3.1	RF2	CU2
CAR3.1, CAR3.5	RF3	CU3
CAR3.2, CAR4.5, CAR5.3	RF4	CU4
CAR4.3	RF5	CU5

Tabla 4 Trazabilidad de las características – Elaboración propia

CU1: Gestión de personajes

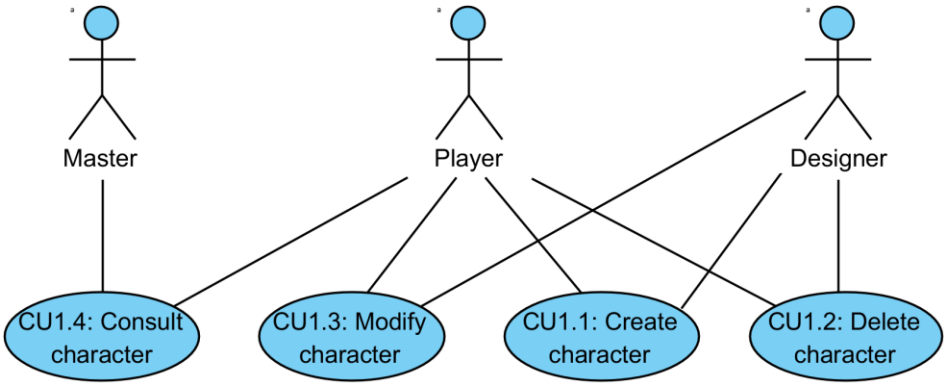

Figura 8 CU1 Gestión de personajes – Elaboración propia

CU1.1: Crear nuevo personaje

Descripción	El jugador y el diseñador deben ser capaces de crear un personaje. Un personaje debe tener nombre, descripción, una imagen y sus estadísticas de combate. Deberá ser posible crear un personaje tanto desde el perfil de la campaña como desde el de uno de los eventos. Si se crea desde este segundo, deberá vincularse automáticamente al evento.
Precondición	

CU1.2: Eliminar personaje

Descripción	El jugador y el diseñador deben ser capaces de eliminar cualquiera de sus personajes desde el perfil de una campaña o desde los personajes del jugador. El personaje se borrará completamente de la base de datos y no podrá ser recuperado.
Precondición	Tener un personaje.

CU1.3: Modificar datos del personaje

Descripción	El jugador y el diseñador deben ser capaces de modificar y guardar en la base de datos los datos de sus personajes (nombre, descripción, imagen y el perfil de combate).
Precondición	Tener un personaje.

CU1.4: Consultar datos del personaje

Descripción	El jugador y el director deben ser capaces de consultar los datos de sus personajes (nombre, descripción, imagen y el perfil de combate).
Precondición	Tener un personaje.

CU2: Gestión de la campaña

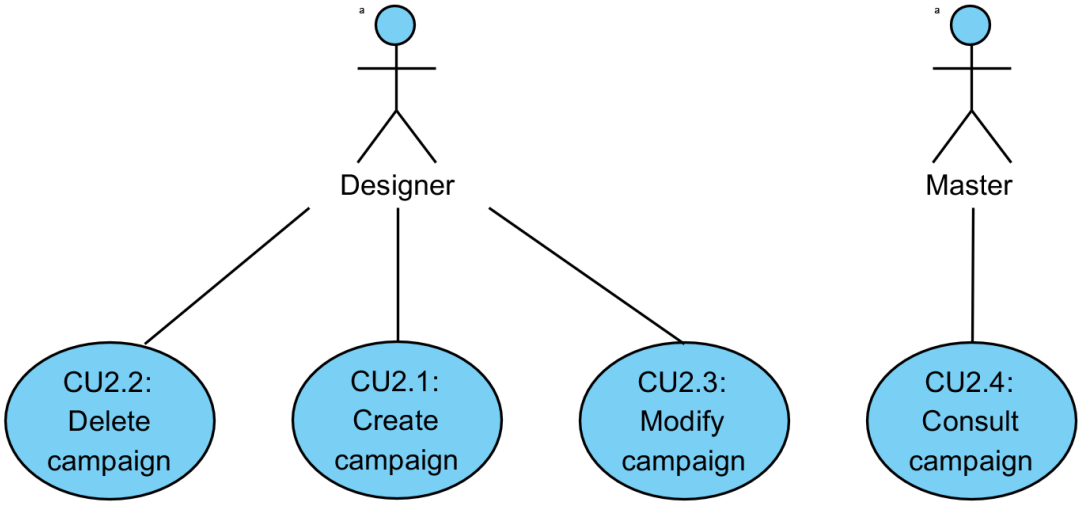

Figura 9 CU2 Gestión de la campaña– Elaboración propia

CU2.1: Crear campaña

Descripción	El diseñador debe ser capaz de crear una campaña (nombre y descripción).
Precondición	

CU2.2: Eliminar campaña

Descripción	El diseñador debe ser capaz de eliminar cualquiera de las campañas que tenga. La campaña se borrará completamente de la base de datos y no podrá ser recuperada.
Precondición	Tener una campaña.

CU2.3: Modificar datos de la campaña

Descripción	El diseñador debe ser capaz de modificar los datos de cualquier campaña que tenga (nombre y descripción).
Precondición	Tener una campaña.

CU2.4: Consultar datos de la campaña

Descripción	El director debe ser capaz de consultar los datos de cualquier campaña que tenga (nombre y descripción).
Precondición	Tener una campaña.

CU3: Gestión de eventos

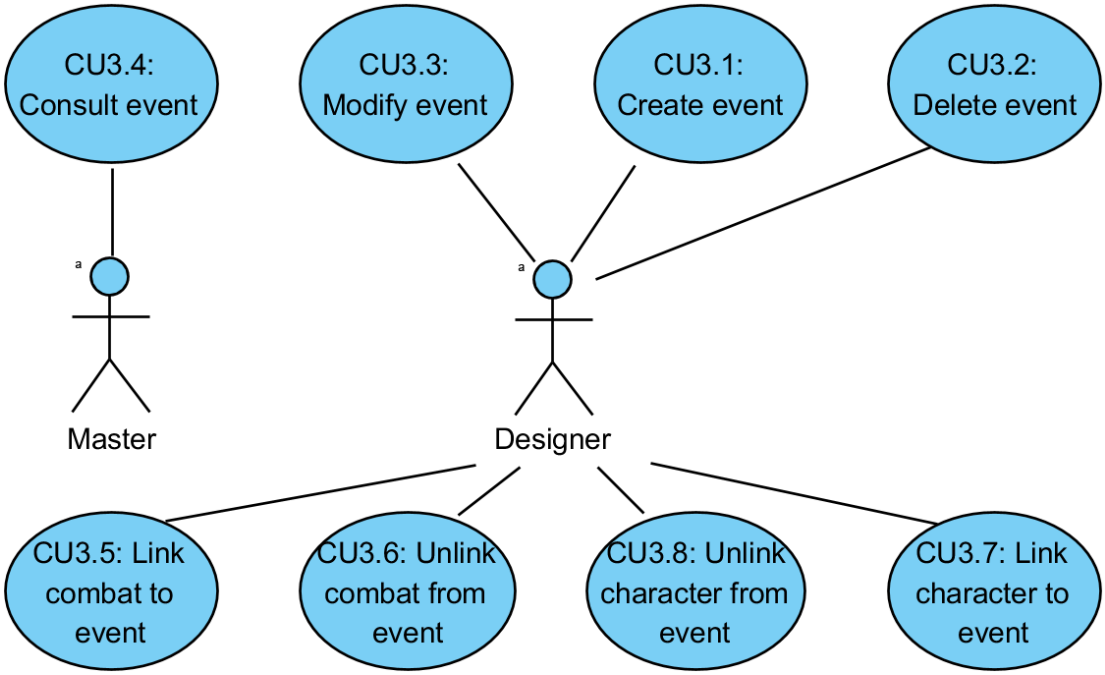

Figura 10 CU3 Gestión de eventos – Elaboración propia

CU3.1: Crear evento

Descripción	El diseñador debe ser capaz de crear eventos en el mapa de eventos de la campaña. Los eventos tendrán un nombre, una descripción y una posición en el mapa.
Precondición	Tener una campaña.

CU3.2: Eliminar evento

Descripción	El diseñador debe ser capaz de eliminar eventos del mapa de eventos de la campaña. El evento se borrará completamente de la base de datos y no podrá ser recuperado.
Precondición	Tener un evento.

CU3.3: Modificar evento

Descripción	El diseñador debe ser capaz de redactar lo que pasa en un lugar determinado en el guion del evento. Además, será capaz de cambiar su nombre, y desde el mapa de eventos, podrá cambiar su posición.
Precondición	Tener un evento.

CU3.4: Consultar evento

Descripción	El director debe ser capaz de consultar lo que pasa en un lugar determinado.
Precondición	Tener un evento.

CU3.5: Vincular combate al evento

Descripción	El diseñador debe ser capaz de vincular uno de los combates que haya creado a un evento.
Precondición	Tener un combate y un evento al que asociarle el combate.

CU3.6: Desvincular combate del evento

Descripción	El diseñador debe ser capaz de desvincular uno de los combates de un evento de dicho evento. Dicho combate no se borrará de la base de datos a menos que se borre desde el perfil de la campaña.
Precondición	Tener un evento con combates.

CU3.7: Vincular personaje al evento

Descripción	El diseñador debe ser capaz de vincular uno de los personajes que haya creado a un evento
Precondición	Tener un personaje y un evento al que asociarle el personaje

CU3.8: Desvincular personaje del evento

Descripción	El diseñador debe ser capaz de desvincular uno de los personajes de un evento de dicho evento. Dicho personaje no se borrará de la base de datos a menos que se borre desde el perfil de la campaña.
Precondición	Tener un evento con personajes.

CU4: Gestión de combates

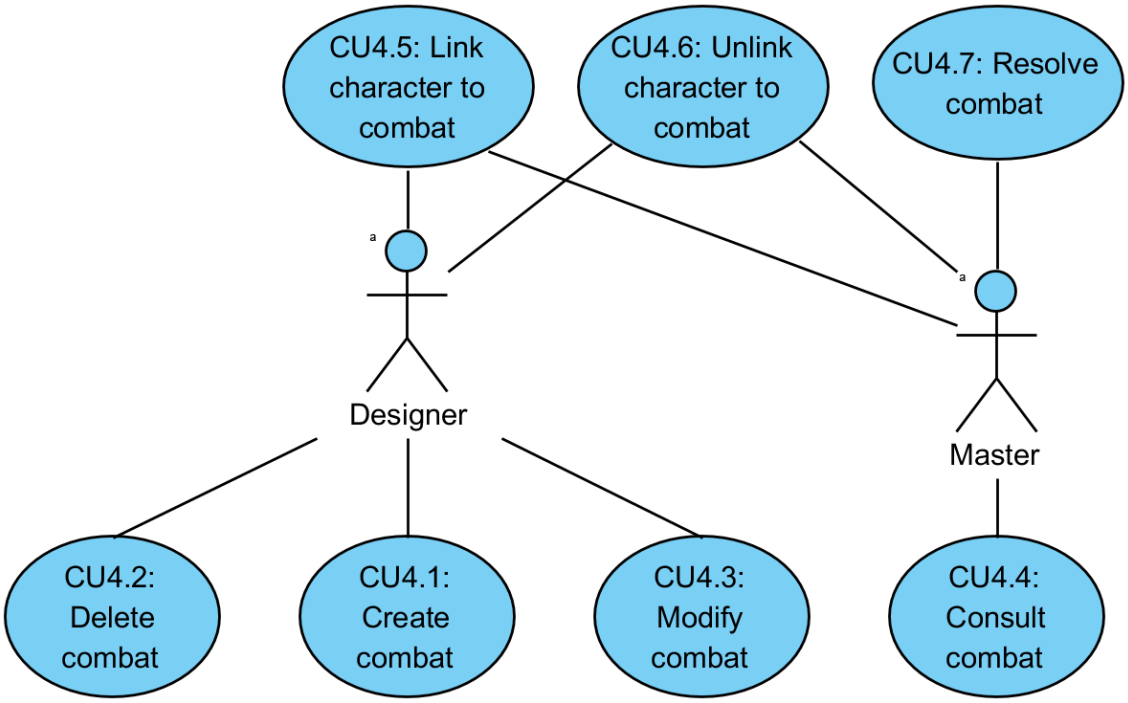

Figura 11 CU4 Gestión de combates – Elaboración propia

CU4.1: Crear combate

Descripción	El diseñador debe ser capaz de crear un combate. Un combate debe tener nombre. Deberá ser posible crear un combate tanto desde el perfil de la campaña como desde el de uno de los eventos. Si se crea desde este segundo, deberá vincularse automáticamente al evento.
Precondición	

CU4.2: Eliminar combate

Descripción	El diseñador debe ser capaz de eliminar cualquiera de sus combates desde el perfil de una campaña. El combate se borrará completamente de la base de datos y no podrá ser recuperado.
Precondición	Tener un combate.

CU4.3: Modificar combate

Descripción	El diseñador debe ser capaz de modificar el nombre de cualquier combate.
Precondición	Tener un combate.

CU4.4: Consultar combate

Descripción	El director debe ser capaz de consultar el estado de cualquier combate.
Precondición	Tener un combate.

CU4.5: Vincular personaje al combate

Descripción	Tanto el diseñador como el director deben poder vincular un personaje a un combate.
Precondición	Tener un combate y un personaje que añadir al combate.

CU4.6: Desvincular personaje del combate

Descripción	Tanto el diseñador como el director deben poder desvincular un personaje de un combate. Dicho personaje no se borrará de la base de datos a menos que se borre desde el perfil de la campaña.
Precondición	Tener un combate con personajes.

CU4.7: Resolver ataque

Descripción	El director debe ser capaz de resolver el ataque en curso únicamente con pulsar un botón. El combate presentará tres fases: calcular las habilidades de combate, calcular el daño y reducir la vida del defensor.
Precondición	Tener a dos o más personajes combatiendo.

CU5: Gestión de consumibles

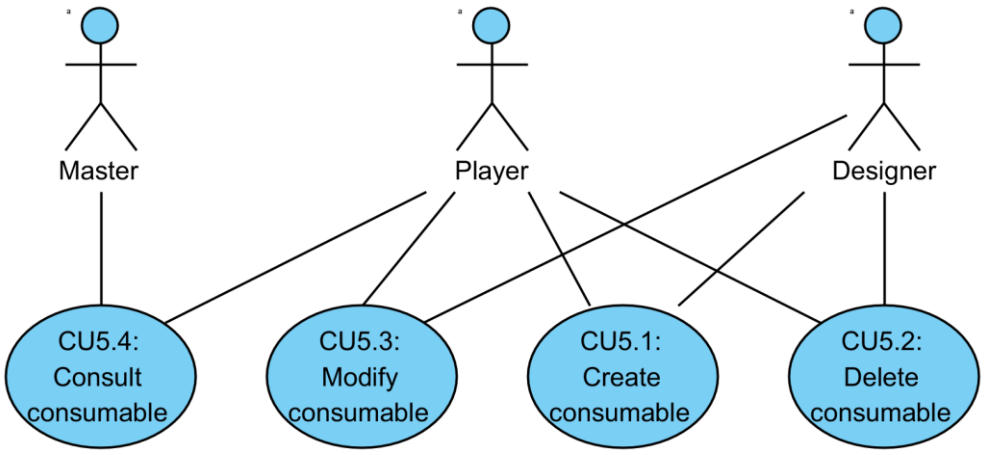

Figura 12 CU5 Gestión de consumibles – Elaboración propia

CU5.1: Crear consumible

Descripción	El diseñador debe ser capaz de crear un consumible. Los consumibles tendrán nombre, valor máximo, valor mínimo, y valor actual.
Precondición	

CU5.2: Eliminar consumible

Descripción	El diseñador debe ser capaz de eliminar cualquiera de los consumibles que tenga. El consumible se borrará completamente de la base de datos y no podrá ser recuperado.
Precondición	Tener un consumible.

CU5.3: Modificar datos del consumible

Descripción	El diseñador debe ser capaz de modificar los datos de cualquier consumible (nombre, valor máximo, valor mínimo, y valor actual). El valor actual deberá poder ser modificado rápidamente sin necesidad de escribir el valor.
Precondición	Tener un consumible.

CU5.4: Consultar datos del consumible

Descripción	El director debe ser capaz de consultar los datos de cualquier consumible (nombre, valor máximo, valor mínimo, y valor actual).
Precondición	Tener un consumible.

3.3.2. Requisitos no funcionales

RNF1: Interfaz intuitiva

Descripción	La interfaz debe ser lo suficiente intuitiva para que un usuario nuevo sea capaz de utilizarla sin necesidad de mirar ninguna guía.

RNF2: Interfaz reajustable

Descripción	La interfaz debe escalarse acorde con el dispositivo sobre el que se esté ejecutando la aplicación.

RNF3: Confirmación de acciones críticas

Descripción	Todas las acciones críticas (como por ejemplo las que supongan la eliminación permanente de algún recurso) deben requerir la confirmación del usuario antes de realizarse.

RNF4: Acciones corregibles

Descripción	Todas las acciones que no sean críticas deben de poderse corregir de una manera u otra sin que suponga un esfuerzo mayor al que se necesitó para realizar la tarea.

RNF5: Retroalimentación

Descripción	Todas las acciones deben proporcionar algún tipo de retroalimentación para que el usuario sepa si ha tenido éxito o no.

3.4. Prototipos

Durante la fase de *brainstorming*, la aplicación se pensó para que la parte del director de juego estuviera en una tableta mientras que los jugadores utilizan sus móviles, sin embargo, tras valorar el estado del arte y especificar los casos de uso, se decidió diseñar la aplicación pensando principalmente en móviles, tanto para director como para jugador, ya que de esta manera se consigue llegar a un mayor número de usuarios. A continuación, se muestran los diseños; en primer lugar los realizados tras la sesión de *brainstorming* y en segundo lugar los realizados después de la especificación. Para los segundos se utilizó una página web de prototipado llamada MockFlow.

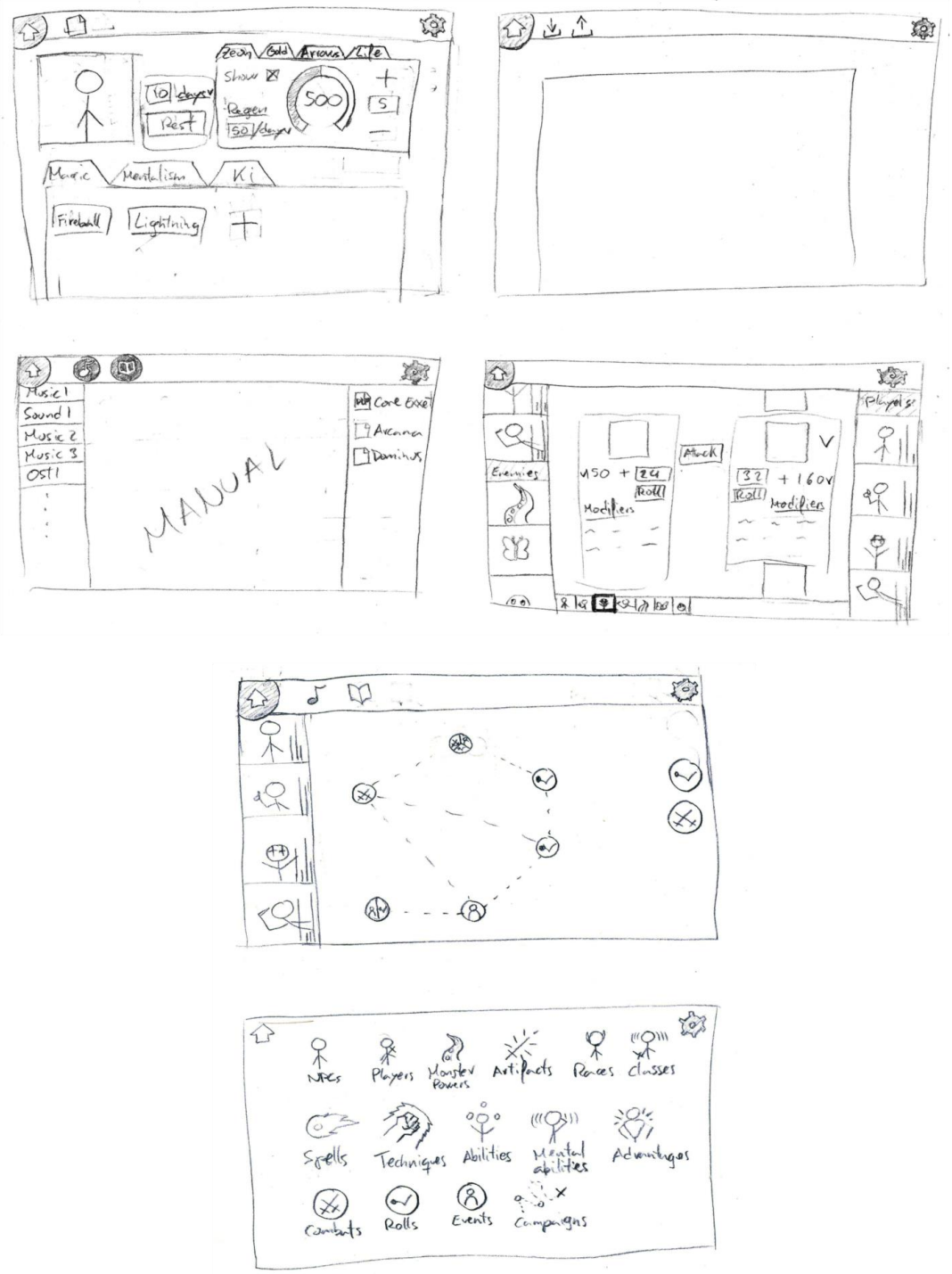

Figura 13 Prototipos del brainstorming – Elaboración propia

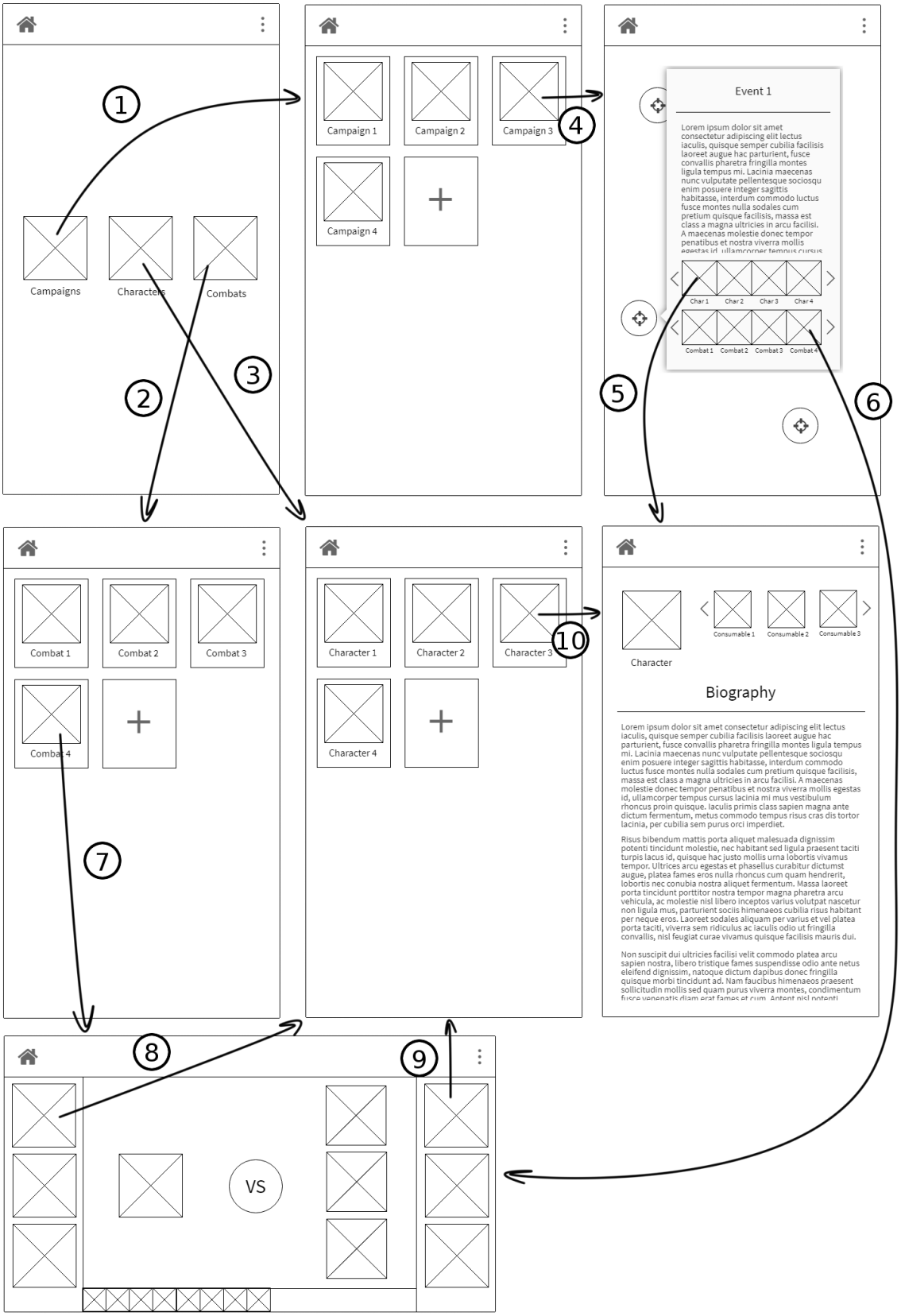

Figura 14 Prototipos de la especificación – Elaboración propia

Aunque a lo largo del desarrollo hubo varios cambios en el diseño, mediante los prototipos que se realizaron tras la especificación ya es posible observar el flujo normal de la aplicación.

La pantalla inicial es la que se encuentra en la esquina superior izquierda de la Figura 14, y desde ella nos es posible navegar al resto de pantallas. Mediante la transición 1, que se realiza pulsando el botón de campañas que se encuentra a la izquierda, podemos navegar hasta la pantalla con la lista de campañas que hayamos creado. Las transiciones 2 y 3 realizan la misma acción, pero en vez de la lista de campañas, se mostrarán la lista de combates y la de personajes respectivamente.

Al pulsar en una de las campañas de la lista de campañas, es posible navegar al mapa de eventos de dicha campaña como se indica en la transición 4. En dicho mapa, seremos capaces de crear eventos e ir creando poco a poco la historia de la campaña mediante la descripción de estos. A los eventos es posible vincular combates y personajes para así tenerlos accesibles. Si pulsamos en uno de esos personajes (transición 5), llegaremos a la pantalla de detalles del personaje, donde es posible cambiar sus datos. Adicionalmente, también podremos gestionar y modificar los consumibles desde esta pantalla.

Si desde un evento pulsamos un combate en vez de un personaje (transición 6), iremos a la pantalla de combate, donde aparecerán los personajes que combaten en dicho combate en la parte izquierda y derecha. En la parte inferior se dispone una barra donde aparecen los personajes ordenados por iniciativa. Desde las listas laterales, podremos seleccionar a el atacante y los defensores para luego hacerles pelear con el botón del centro; el daño que recibirá cada uno de los defensores se calculará automáticamente. Adicionalmente, también podremos navegar hasta la pantalla de detalles de los personajes del combate pulsando sobre ellos como se muestra en las transiciones 8 y 9.

Po último, como es lógico, también es posible acceder a las pantallas de detalle del personaje y combate desde las listas de personajes y combates respectivamente.

3.5. Plan de trabajo

Como se mencionó anteriormente, el sistema utilizado para la gestión del desarrollo de la aplicación es Scrum. El *sprint cero* se dedicó a analizar el estado del arte, a realizar la especificación de requisitos y a preparar el tablero *Kanban* con las tareas identificadas; además, también se priorizaron las tareas y se prepararon las que serían implementadas en el primer *sprint*.

Para estimar el tiempo que requiere cada tarea, se utilizó un sistema de *Story Points*. Este sistema estima la duración de las tareas mediante una unidad de tiempo llamada *Story Point (SP)*, que en nuestro caso representa medio día. Así, una tarea que requiera de 3 SP significa que para implementarla se requerirá un día y medio de trabajo. Por tanto, teniendo en cuenta que los *sprints* son de dos semanas, en un *sprint* como mucho deberían haber 28 SP. El diagrama de Gantt obtenido fue el siguiente:

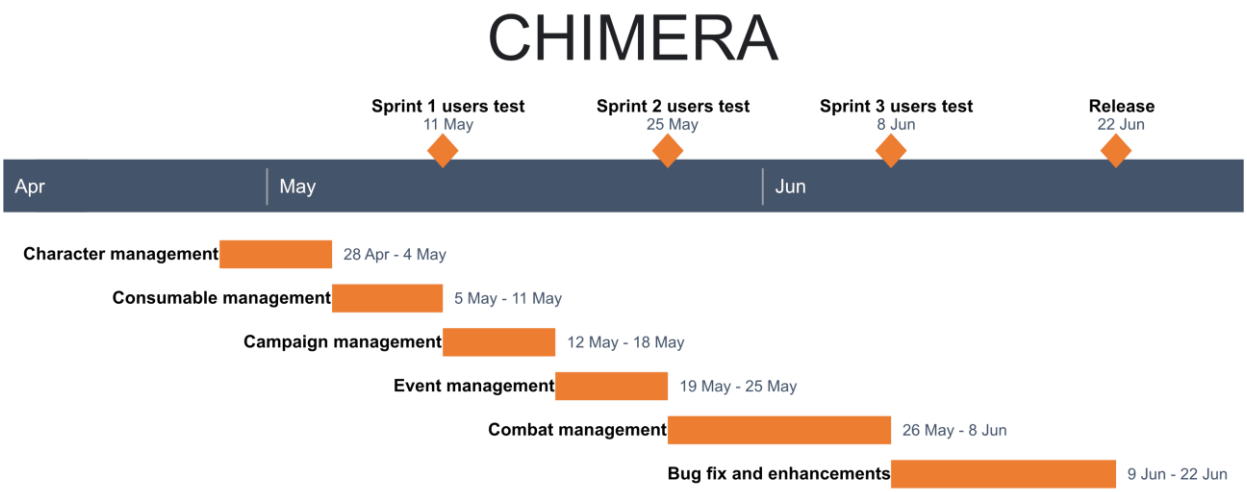

Figura 15 Diagrama de Gantt – Elaboración propia

3.6. Presupuesto

En cuanto al presupuesto del proyecto, se asume que se dispone de un ordenador con la capacidad suficiente para compilar el proyecto. Tampoco hizo falta tener en cuenta ninguna clase de licencia ya que, como se explica más adelante en el punto 4.3 todas las herramientas utilizadas son gratuitas. Además, debido a que el producto final no requiere de ningún componente hardware, y que no se contempló introducir contenido de autor en una primera versión del producto, solo hubo que tener en cuenta el sueldo del único desarrollador del proyecto.

Suponiendo un sueldo de 1500€/mes, es necesario un presupuesto inicial de 3.750€ para el desarrollo de la primera versión del producto.

3.7. Análisis de riesgos

A continuación, se disponen los riesgos que se identificaron durante la etapa inicial del proyecto. Como se indica más adelante en el punto 5.1.5, el riesgo número 4 se manifestó y se mitigó en el sprint 4.

RI1: Cambios en los requisitos

Impacto:	Baja el índice de aceptación de los usuarios.	Tipo:	Proyecto y Producto
Medidas:	Aplicar el desarrollo ágil para poder identificar dichos cambios cuanto antes y así evitar que queden fuera del proyecto.		

RI2: Subestimación del tamaño de un requisito

Impacto:	Retrasos en las entregas.	Tipo:	Proyecto
Medidas:	Realizar reuniones con usuarios potenciales para reevaluar el alcance.		

RI3: Aparece una aplicación competidora

Impacto:	Posible cierre del proyecto.	Tipo:	Producto
Medidas:	Realizar reuniones con los usuarios potenciales para reevaluar la dirección del proyecto y priorizar las funcionalidades que marquen la diferencia.		

RI4: Los usuarios no pueden participar en las reuniones

Impacto:	Dificultades para la gestión de prioridades y dirección del proyecto.	Tipo:	Proyecto
Medidas:	Uso de foros y encuestas web para la recopilación de retroalimentación.		

4. Diseño de la solución

4.1. Arquitectura del sistema

Para la arquitectura del sistema se ha optado por una aproximación tradicional de los sistemas para dispositivos móviles.

En primer lugar, tenemos las `Activity` y los `Fragment` que representan las pantallas y subpantallas de la aplicación. Las `Activity` son las clases encargadas tanto de mostrar al usuario la representación gráfica de la aplicación, como de definir el comportamiento de cada uno de los elementos gráficos. Los `Fragment` tienen la misma finalidad que las `Activity` con la diferencia de que los `Fragment` no representan por si mismos una pantalla completa de la aplicación, sino que representan un fragmento de esta, pudiendo ser reutilizados en diferentes `Activity`. Por ejemplo, los perfiles de la aplicación tienen un `Fragment` para cada sección.

El segundo elemento de la arquitectura lo forman los `ViewModel`. Las `Activity` por definición son volátiles y solo deberían encargarse de la parte gráfica de la aplicación; una de las razones más notorias de porqué debería ser así es que, al girar la pantalla, la `Activity` se destruye y se crea una con la nueva resolución de pantalla (horizontal/vertical). Por ello es necesario que algo ajeno a la `Activity` mantenga la información de la pantalla; esa es la funcionalidad de un `ViewModel`. Los `ViewModel`, mantienen la información obtenida de la persistencia mediante los llamados `LiveData`. Los `LiveData` son representaciones de los objetos de la base de datos que lanzan eventos siempre que son modificados de alguna manera. De esta forma, los `ViewModel` serán informados de los cambios haciendo que sea mucho más rápido y eficiente mostrar estos cambios por pantalla.

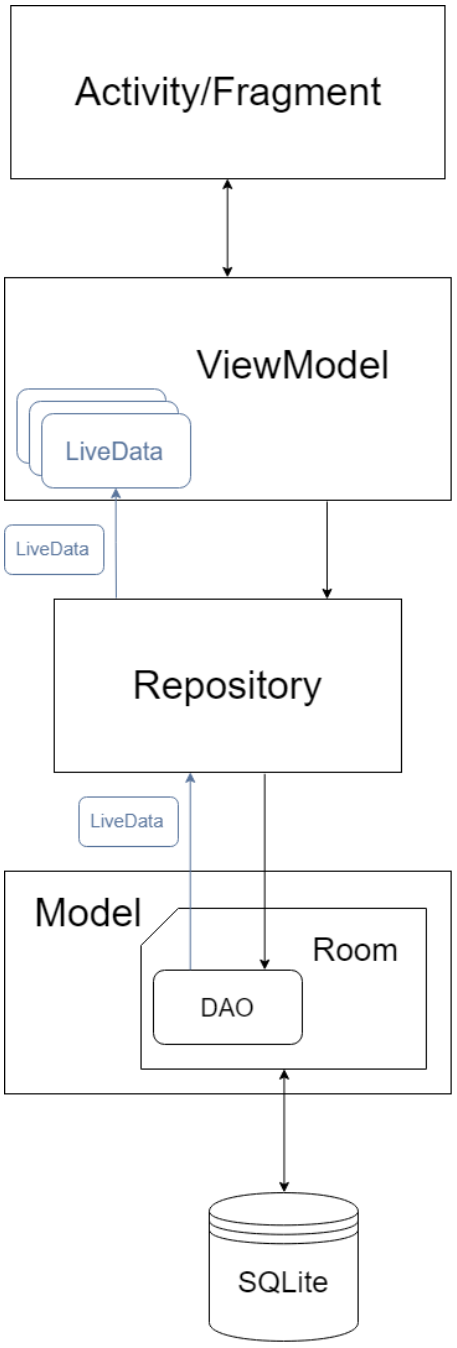

Figura 16 Arquitectura – Elaboración propia

El tercer elemento es el repositorio. Un repositorio sirve de capa intermedia entre el modelo y los `ViewModel`, haciendo que este quede desacoplado de las clases de la presentación. Este elemento no es indispensable, pero hace mucho más fácil la mantenibilidad y nos permitiría cambiar la base de datos en un futuro, en caso de que en algún momento se requiera un tipo de persistencia diferente o más potente.

El cuarto elemento lo forman los modelos. Los modelos son las representaciones digitales de los objetos de la aplicación (personajes, consumibles, combates, etc.) y conforman las clases principales. Estos son los objetos que se encargan de almacenar la información mientras la aplicación se está ejecutando, y son los que viajan dentro de los `LiveData`. En esta capa también se encuentran los DAO (*Data Access Object*) que son puntos de acceso a las diferentes tablas de la base de datos. Los DAO se encargan de hacer llamadas a la base de datos para realizar las operaciones deseadas (consulta, actualización, borrado e inserción).

Por último, solo queda mencionar la base de datos. Se trata de una base de datos SQLite, a la que se accede mediante una librería muy extendida llamada Room, que hace transparente toda la implementación de la base de datos, agilizando enormemente el desarrollo de la aplicación.

4.2. Diseño detallado

4.2.1. Diagramas de clases

Se realizaron tres diagramas de clases, uno para los modelos de la lógica, otro para mostrar las relaciones entre los repositorios y los `ViewModel` y un último diagrama para las clases de la presentación (`Activity` y `Fragment`).

Diagrama de modelos

El siguiente diagrama de clases muestra las principales clases de la aplicación y sus relaciones entre si. Como se observa, una campaña puede tener asociados diversos combates, personajes y eventos. Esos eventos se mostraran en el mapa de eventos de la campaña y a su vez, los eventos pueden tener personajes y combates asociados a ellos. Esta manera de asociar elementos, permite tener la información que será necesaria para la partida de una forma estructurada, y al estar todo vinculado, se podrá acceder rápidamente a los elemntos relacionados.

Por otra parte también se puede observar que en un combate pueden participar multiples personajes. Sin embargo todos los elementos tienen que estar ligados a una única campaña para mantener el estado de cada elemento.

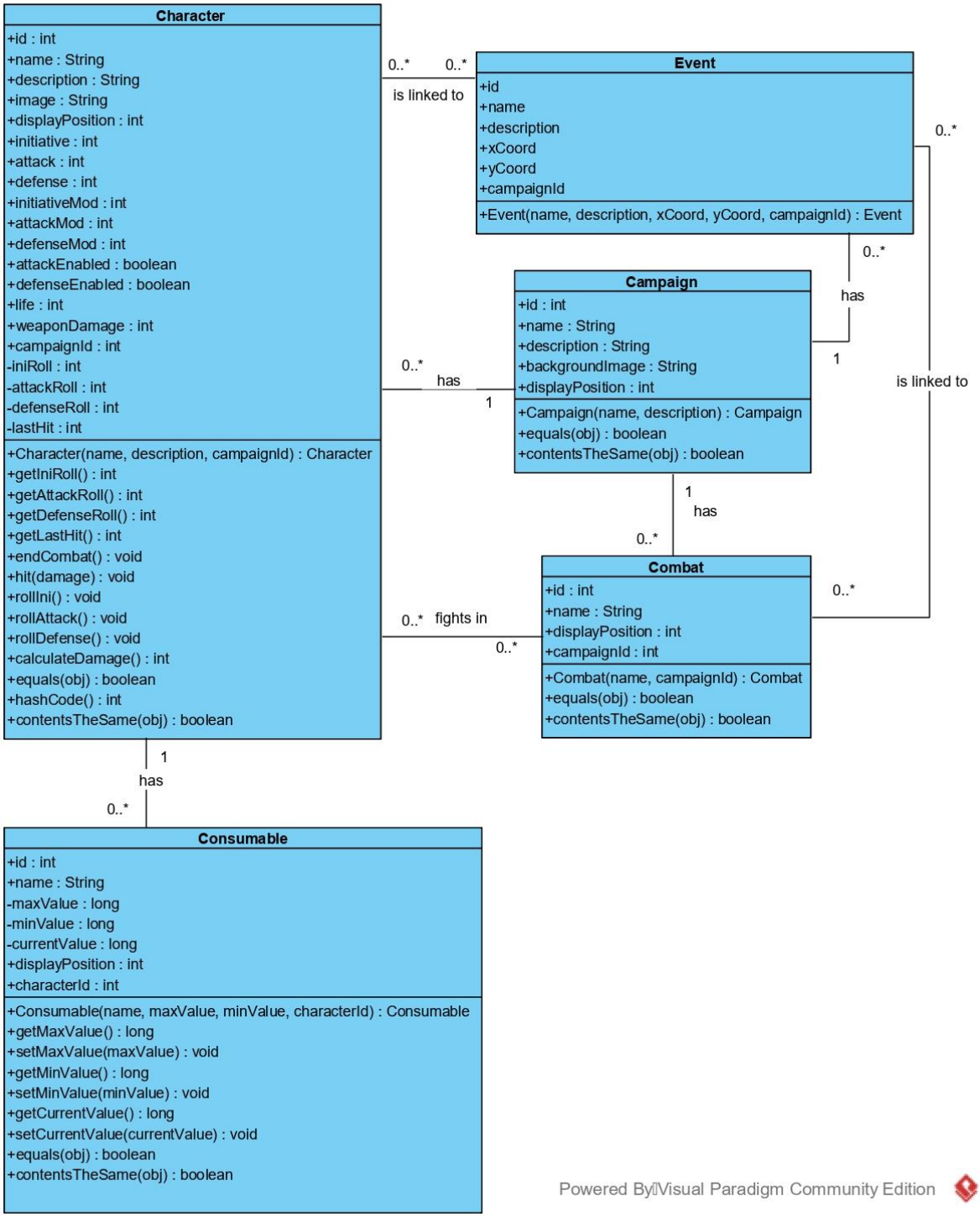

Figura 17 Diagrama de clases de la lógica – Elaboración propia

Diagrama de repositorios y VM

En los siguientes diagramas se puede observar como los `ViewModel` hacen uso de los diferentes repositorios, y estos a su vez, de los DAOs. Los DAOs no fueron implementados directamente, sino que se crearon interfaces anotadas con anotaciones de Room; Room, cuando se compila el proyecto, lee esas anotaciones y crea sus implementaciones para que podamos obtener objetos de la base de datos.

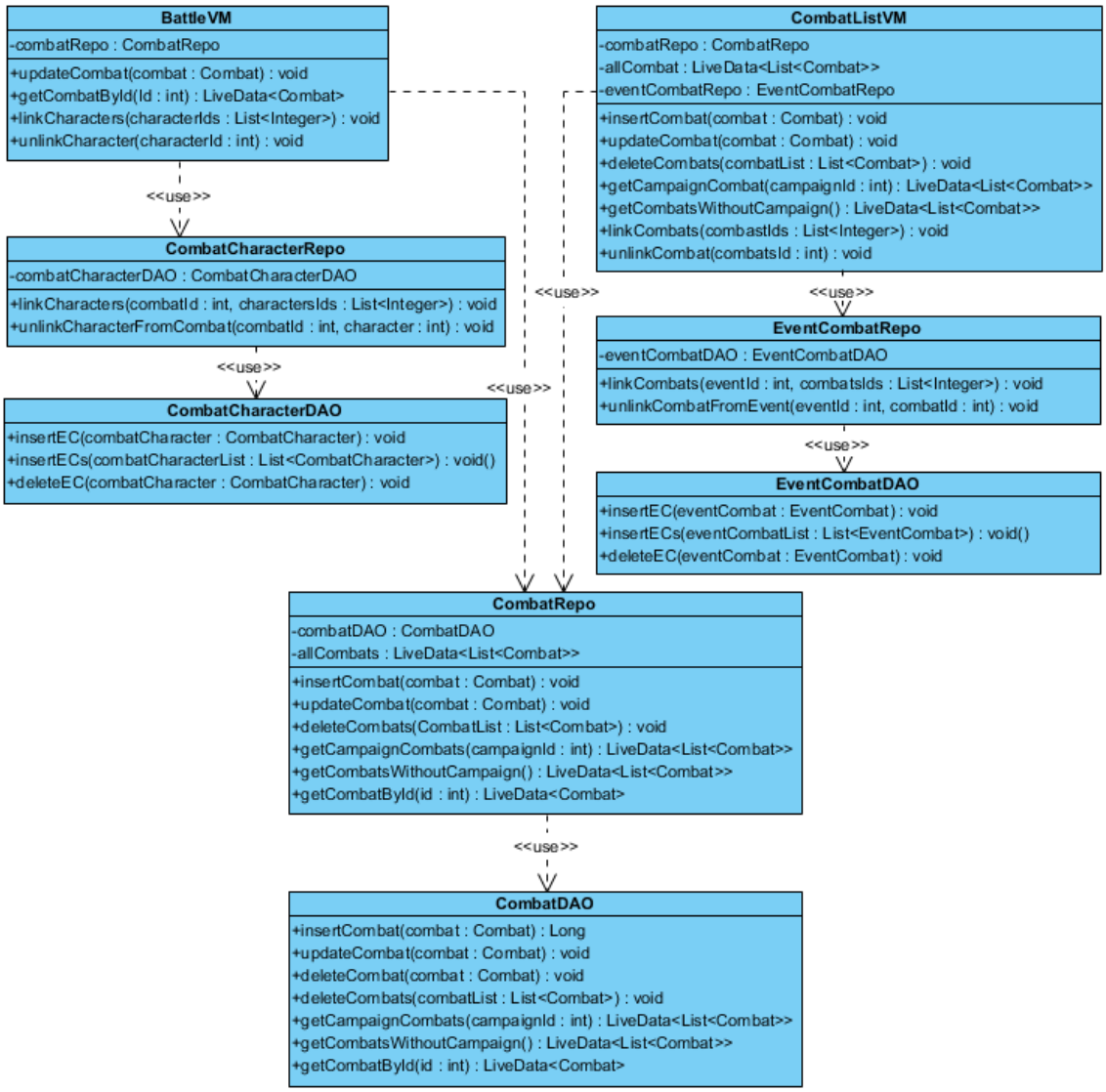

Figura 18 Diagrama de clases de ViewModel, Repositorios y DAO (1) – Elaboración propia

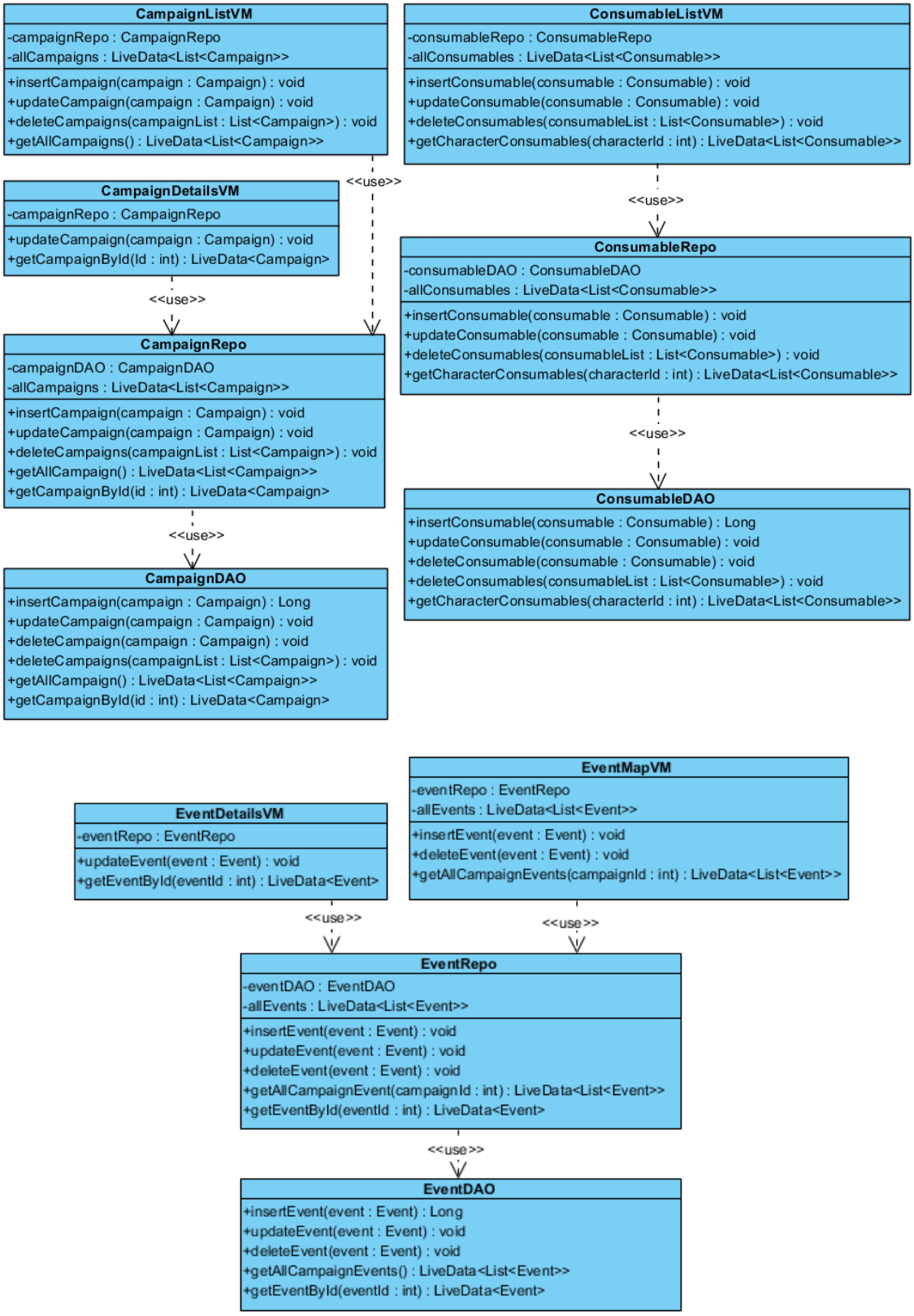

Figura 19 Diagrama de clases de ViewModel, Repositorios y DAO (2) – Elaboración propia

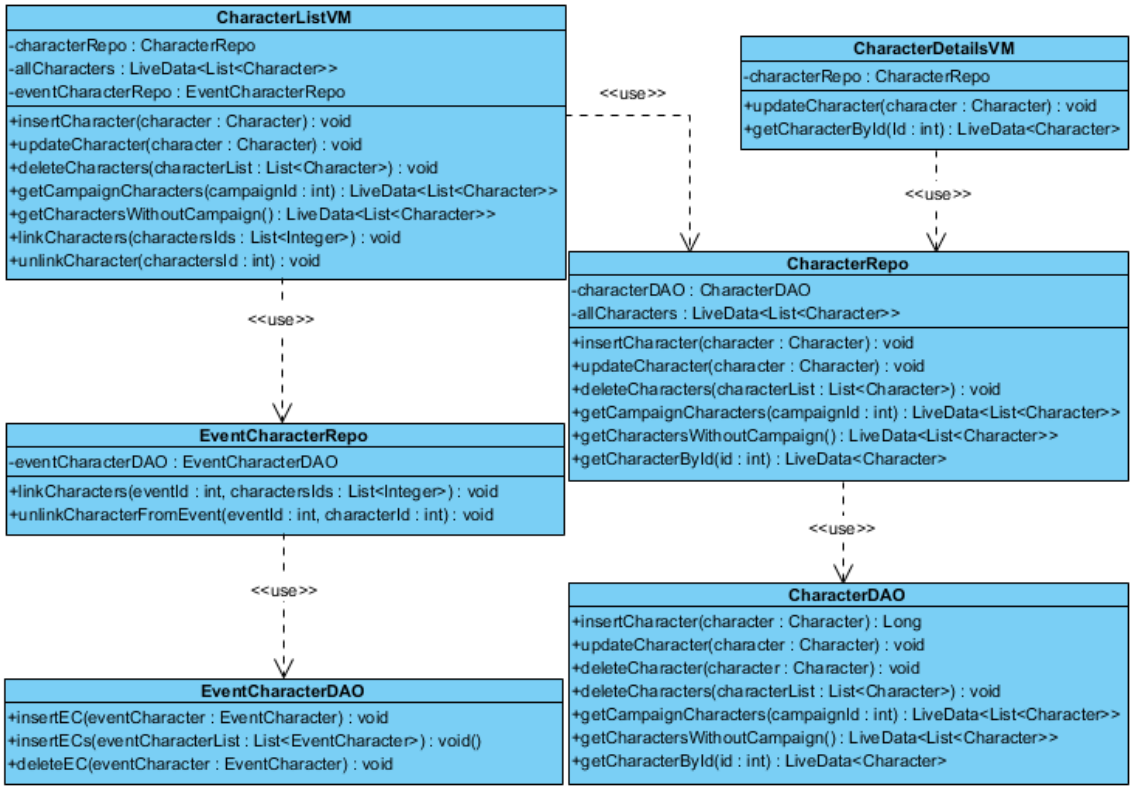

Figura 20 Diagrama de clases de ViewModel, Repositorios y DAO (3) – Elaboración propia

Diagramas de las Activity y los Fragment

A continuación, se muestran el diagrama de las `Activity` y el de los `Fragment`. Como se puede observar, todas las `Activity` dependen de `ActivityWithUpperBar`; esto es así para poder tener una barra superior customizada que tenga el texto centrado. También se puede observar que `BattleActivity` es notoriamente más grande que el resto; y es que esta `Activity` necesita hacer muchas llamadas a la lógica de la aplicación ya que los combates requieren de realizar muchos cálculos (el cálculo de la iniciativa, el daño, etc.).

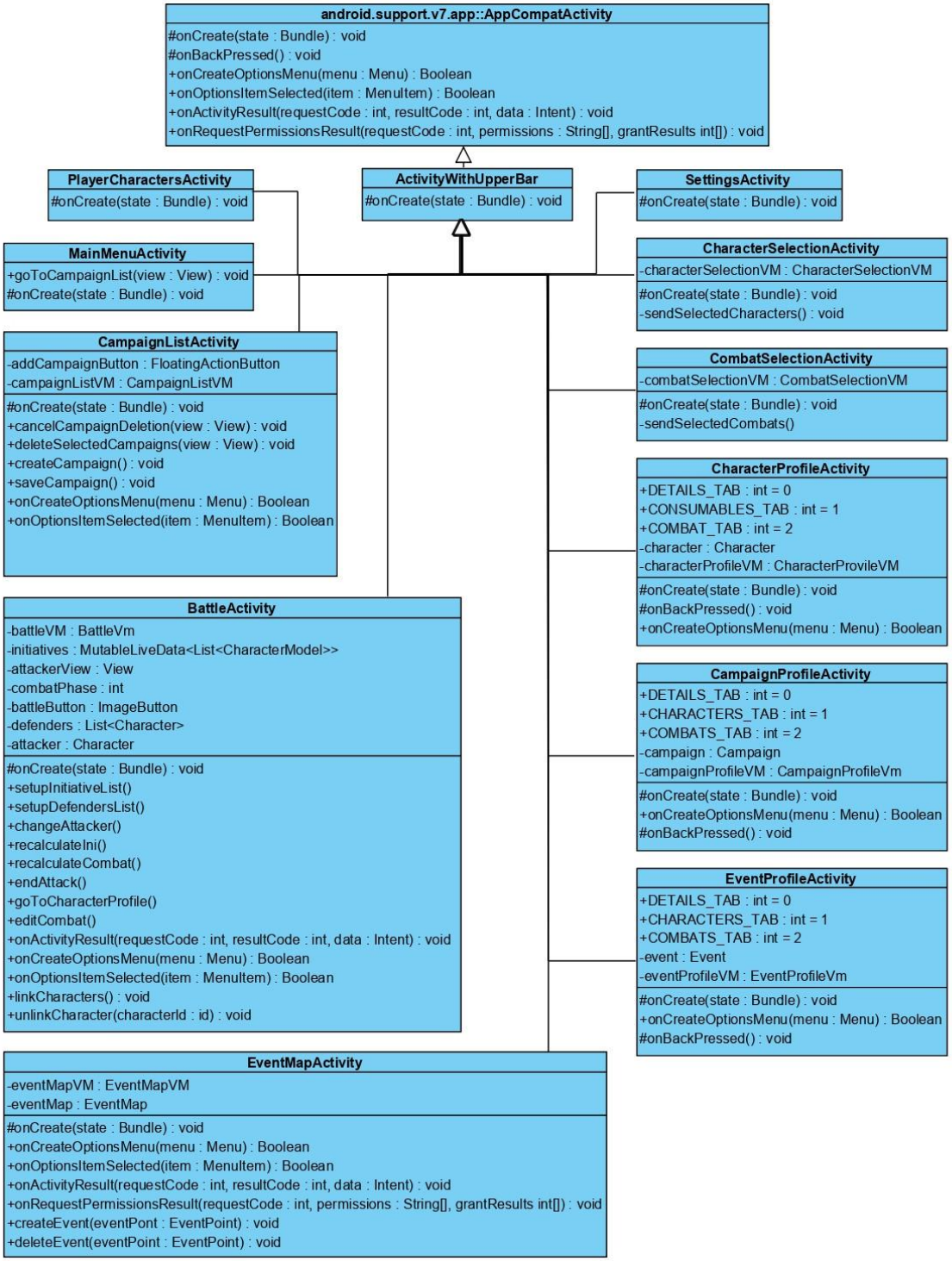

Figura 21 Diagrama de clases de las Activity – Elaboración propia

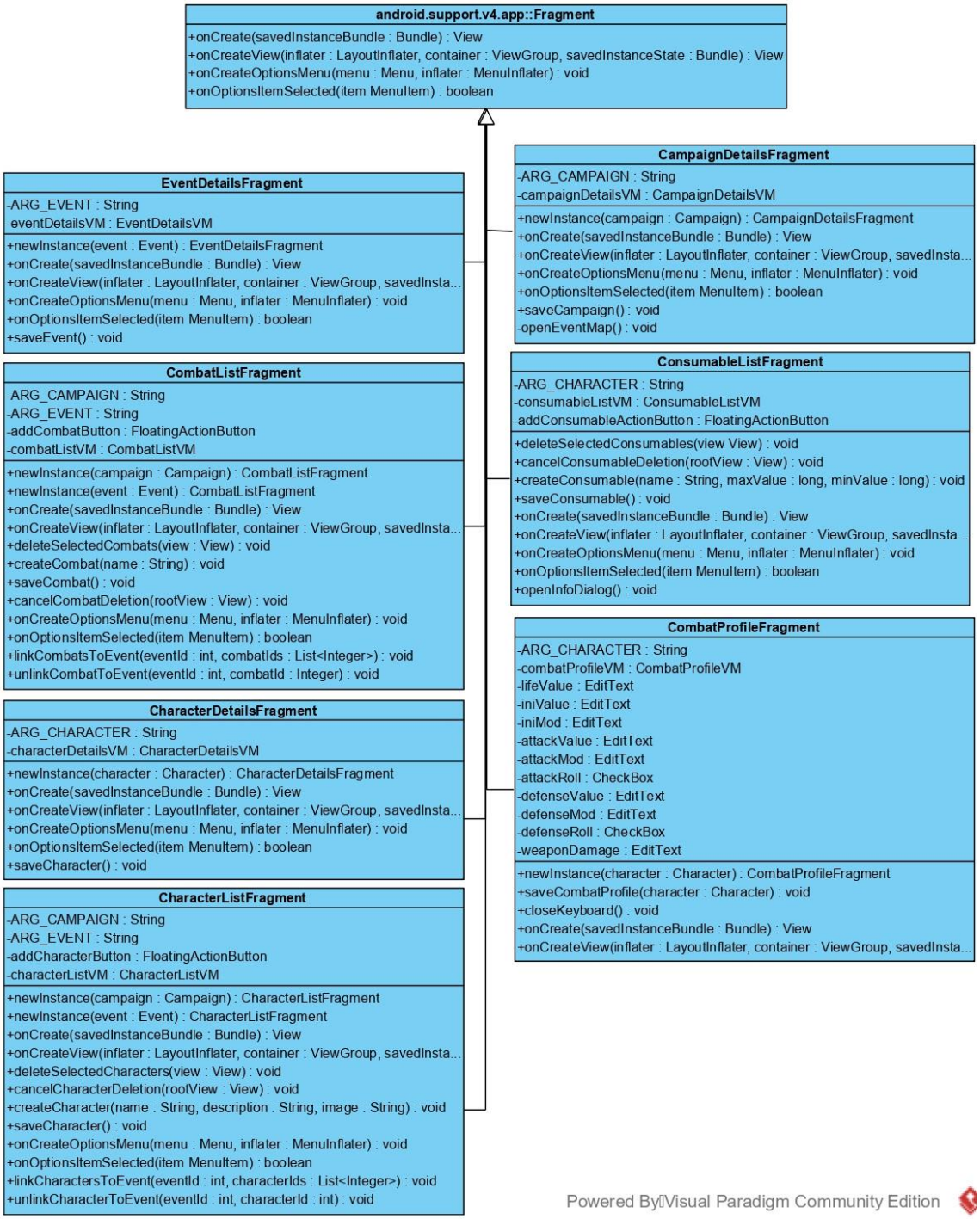

Figura 22 Diagrama de clases de los Fragment– Elaboración propia

4.2.2. Diagrama relacional

A continuación, se muestra el esquema de la base de datos SQLite. Al utilizar la librería Room no fue necesario especificar directamente las tablas que aparecen en el diagrama, en vez de eso, se anotaron los modelos de la lógica con las etiquetas correspondientes y la librería generó automáticamente las tablas necesarias.

La tabla principal sobre la que se organiza la base de datos es la campaña (Campaign), que dispone de un nombre, una descripción y una imagen de fondo para incluir un mapa. Dentro de la campaña hay una lista de eventos (Event), con su nombre, la descripción de lo que sucede en ese lugar y las coordenadas del mapa en las que se encuentra.

Las campañas, además, también tienen una lista de personajes (Character) que participan en ella, los cuales tienen nombre, descripción, imagen de perfil y una serie de características de combate (ataque, defensa, iniciativa, etc.). Los personajes a su vez disponen de una lista de consumibles (Consumable), que tienen nombre, valor máximo, valor mínimo y valor actual. Como se puede observar, un consumible pertenece siempre a un único personaje.

Los combates (Combats), al igual que los personajes y los eventos, también dependen de una campaña. Estos, únicamente tienen nombre, ya que lo importante es su relación con los personajes. Un personaje puede estar en muchos combates y en un combate pueden participar muchos personajes, por lo que es necesario modelar una relación muchos a muchos.

Lo último que queda por destacar, son las relaciones que presentan los eventos. Dejando de lado la relación que tiene con la campaña, un evento se relaciona también con combates y con personajes. En ambos casos se trata de una relación muchos a muchos, es decir, en un evento pueden estar vinculados muchos combates y/o personajes y los personajes y los combates pueden aparecer en muchos eventos. También es importante mencionar que tanto los eventos como los combates tienen que estar en una única campaña, sin embargo, los personajes pueden no participar en ninguna. Los personajes que no participan en ninguna, son los personajes del jugador.

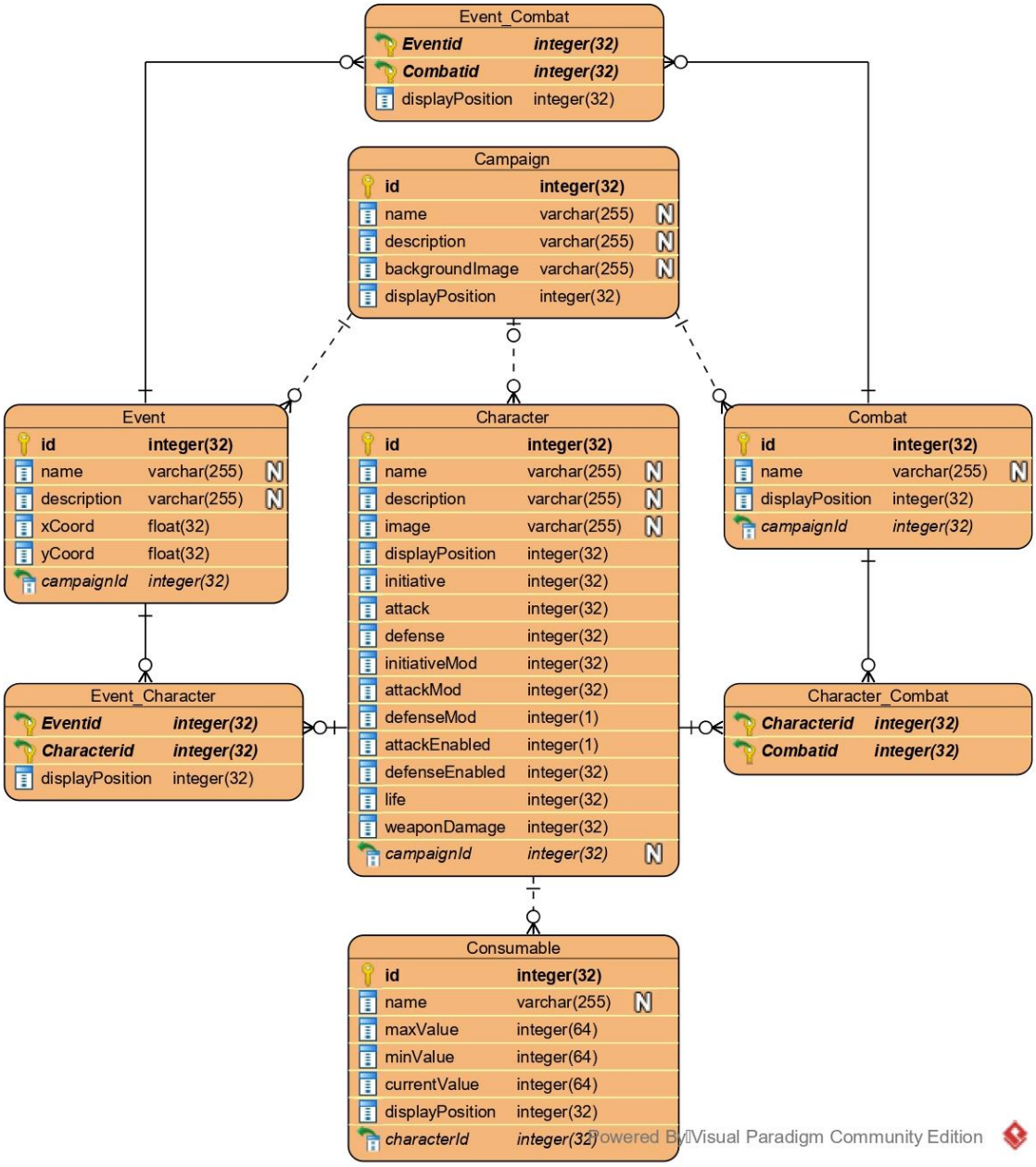

Figura 23 Diagrama relacional – Elaboración propia

4.2.3. Diagramas de secuencia

A continuación se listan los diagramas de secuencia realizados, cada uno corresponde con uno de los casos de uso identificados durante la especificación de requisitos.

CU1.1: Crear nuevo personaje

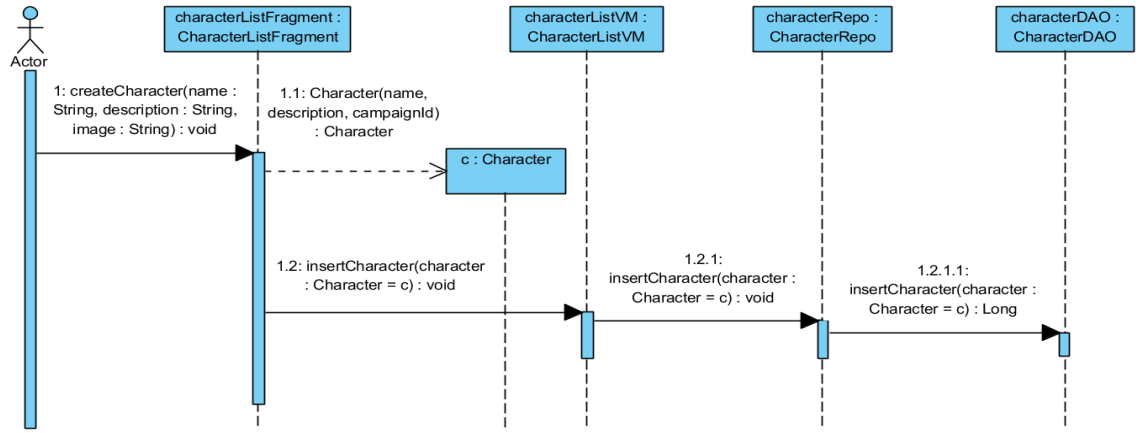

Figura 24 Diagrama de secuencia de crear nuevo personaje – Elaboración propia

CU1.2: Eliminar personaje

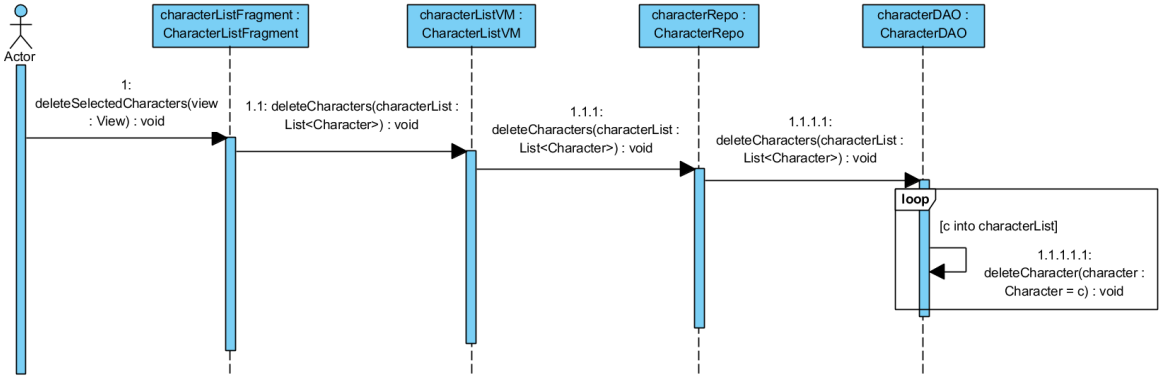

Figura 25 Diagrama de secuencia de eliminar personaje– Elaboración propia

CU1.3: Modificar datos del personaje

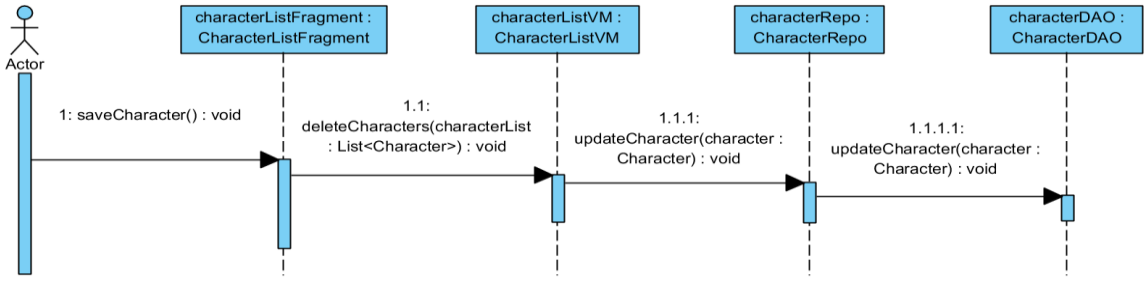

Figura 26 Diagrama de secuencia de modificar personaje– Elaboración propia

CU1.4: Consultar datos del personaje

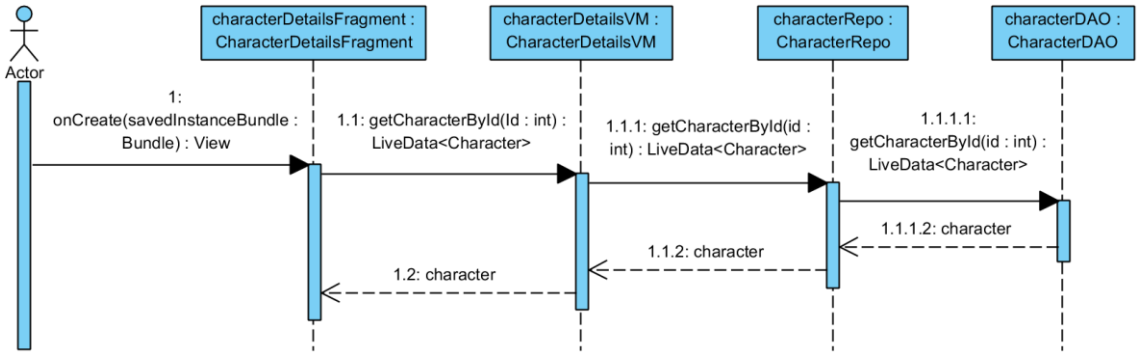

Figura 27 Diagrama de secuencia de consultar datos del personaje– Elaboración propia

CU2.1: Crear campaña

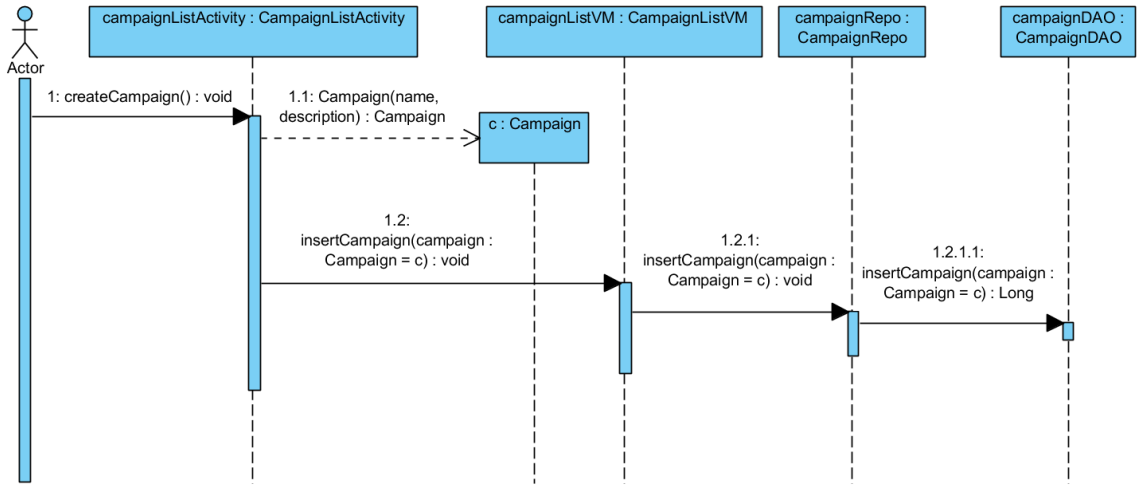

Figura 28 Diagrama de secuencia de crear campaña – Elaboración propia

CU2.2: Eliminar campaña

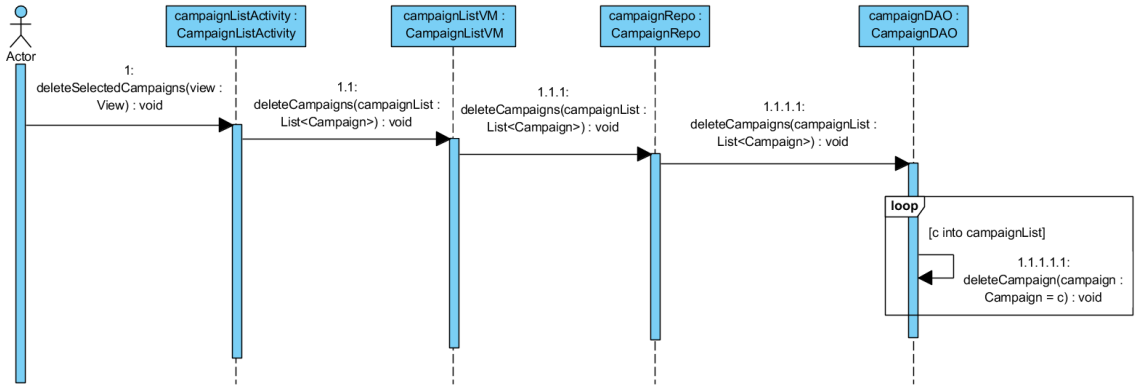

Figura 29 Diagrama de secuencia de eliminar campaña – Elaboración propia

CU2.3: Modificar datos de la campaña

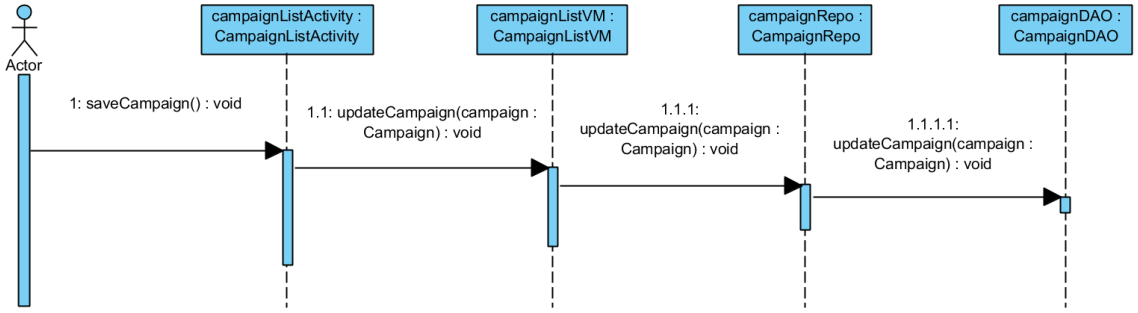

Figura 30 Diagrama de secuencia de modificar datos de la campaña – Elaboración propia

CU2.4: Consultar datos de la campaña

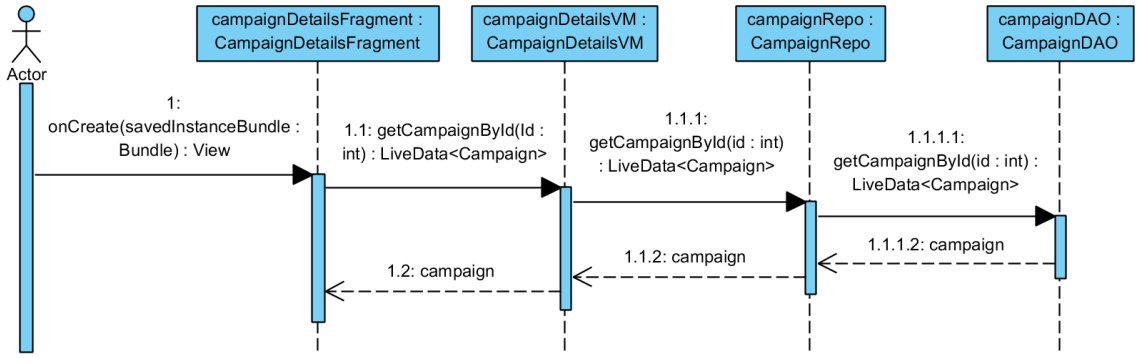

Figura 31 Diagrama de secuencia de consultar datos de la campaña – Elaboración propia

CU3.1: Crear evento

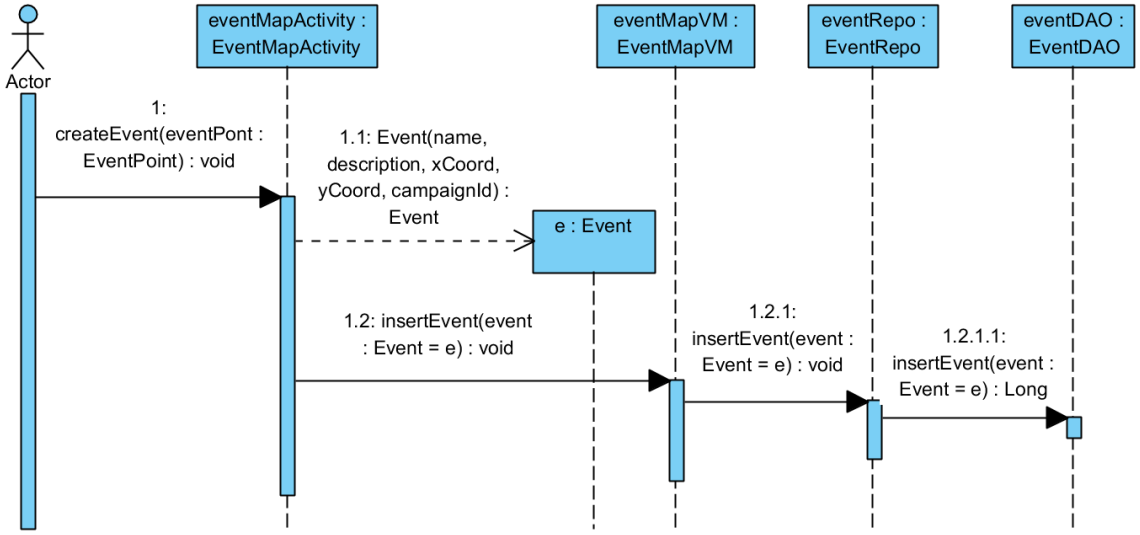

Figura 32 Diagrama de secuencia de crear evento – Elaboración propia

CU3.2: Eliminar evento

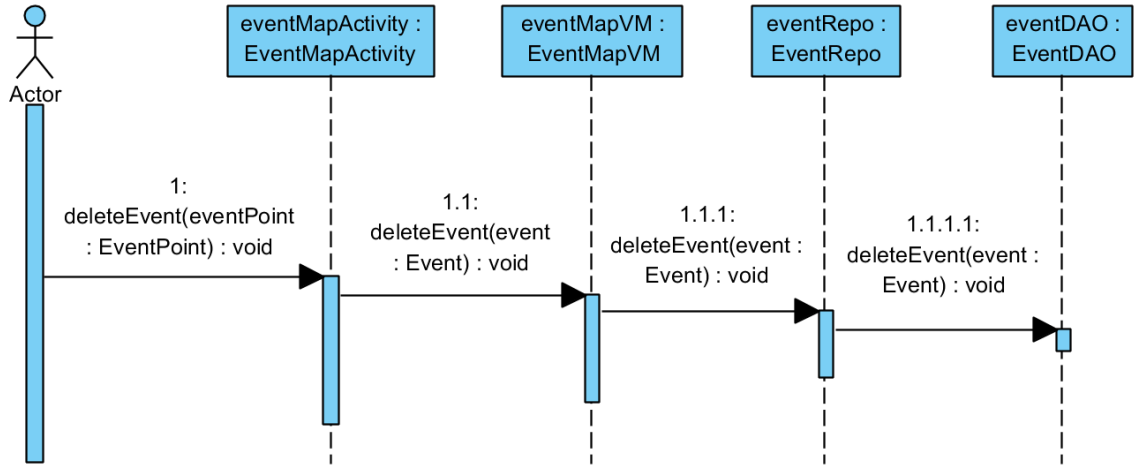

Figura 33 Diagrama de secuencia de eliminar evento – Elaboración propia

CU3.3: Modificar evento

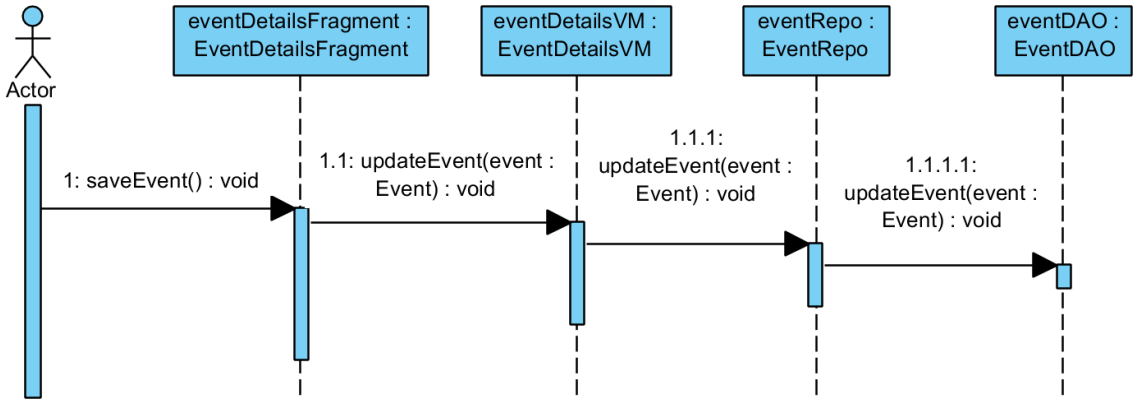

Figura 34 Diagrama de secuencia de modificar evento – Elaboración propia

CU3.4: Consultar evento

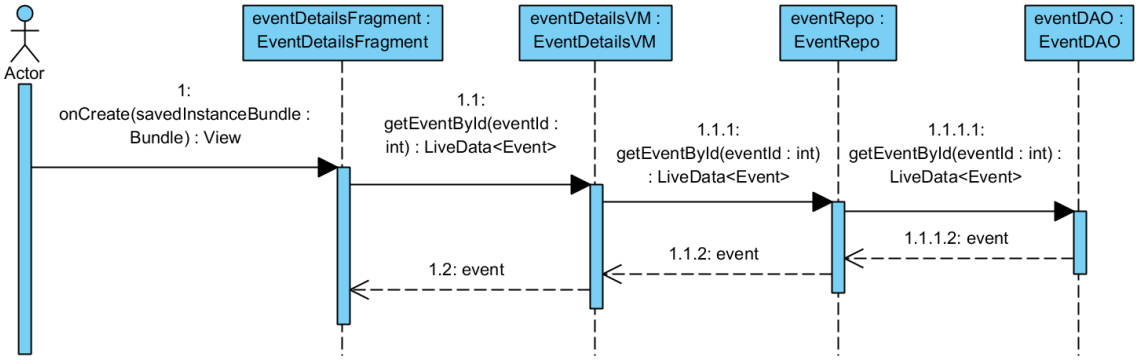

Figura 35 Diagrama de secuencia de consultar evento – Elaboración propia

CU3.5: Vincular combate al evento

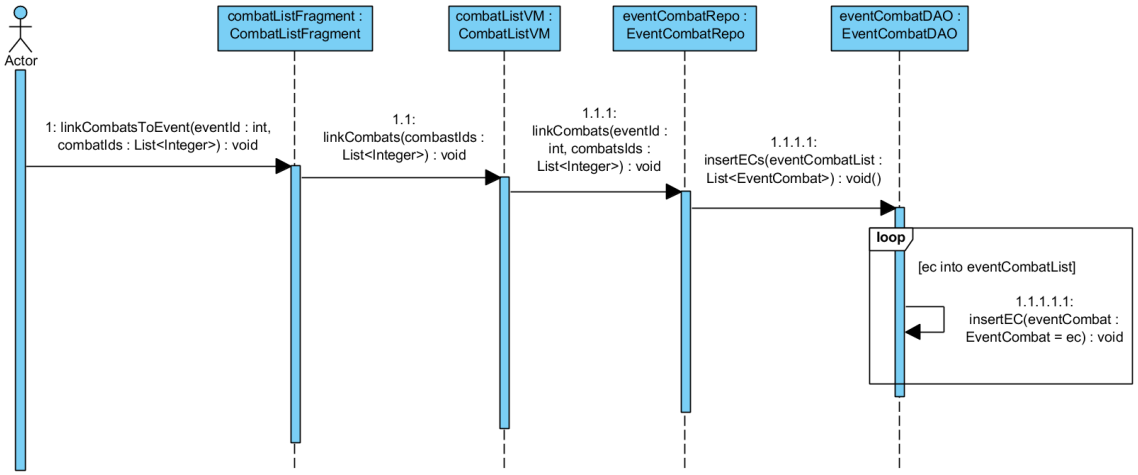

Figura 36 Diagrama de secuencia de vincular combate al evento – Elaboración propia

CU3.6: Desvincular combate del evento

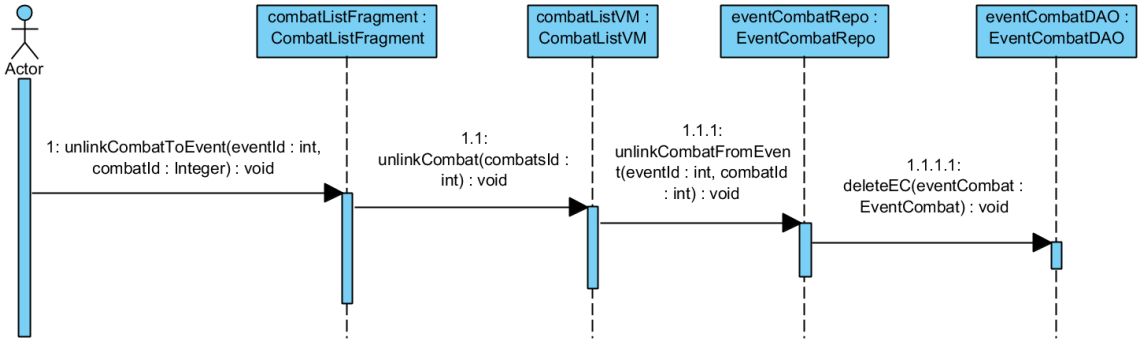

Figura 37 Diagrama de secuencia de desvincular combate del evento – Elaboración propia

CU3.7: Vincular personaje al evento

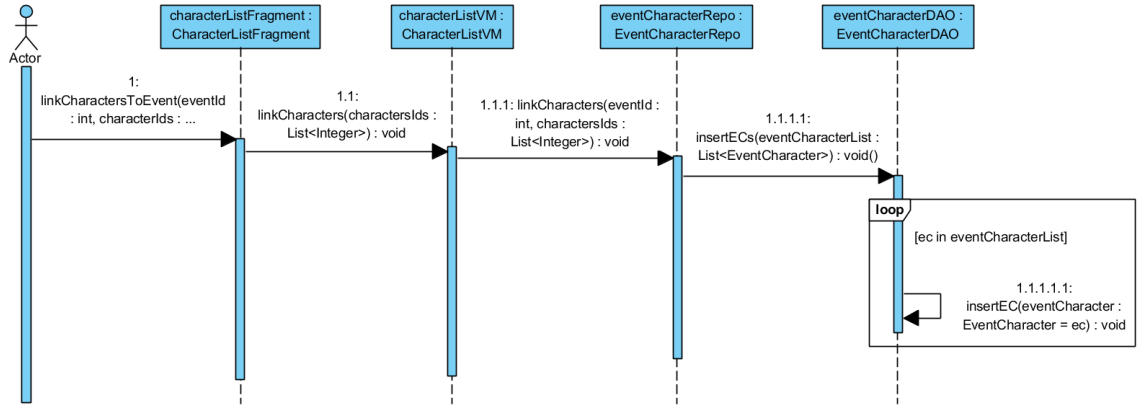

Figura 38 Diagrama de secuencia de vincular personaje al evento – Elaboración propia

CU3.8: Desvincular personaje del evento

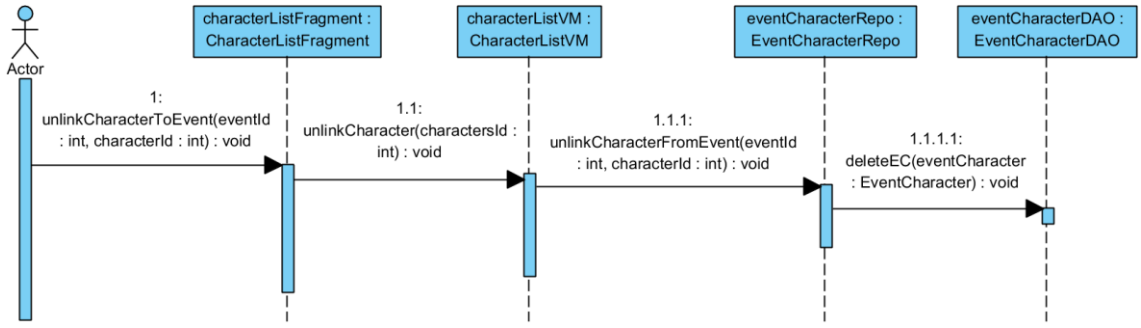

Figura 39 Diagrama de secuencia de desvincular personaje del evento – Elaboración propia

CU4.1: Crear combate

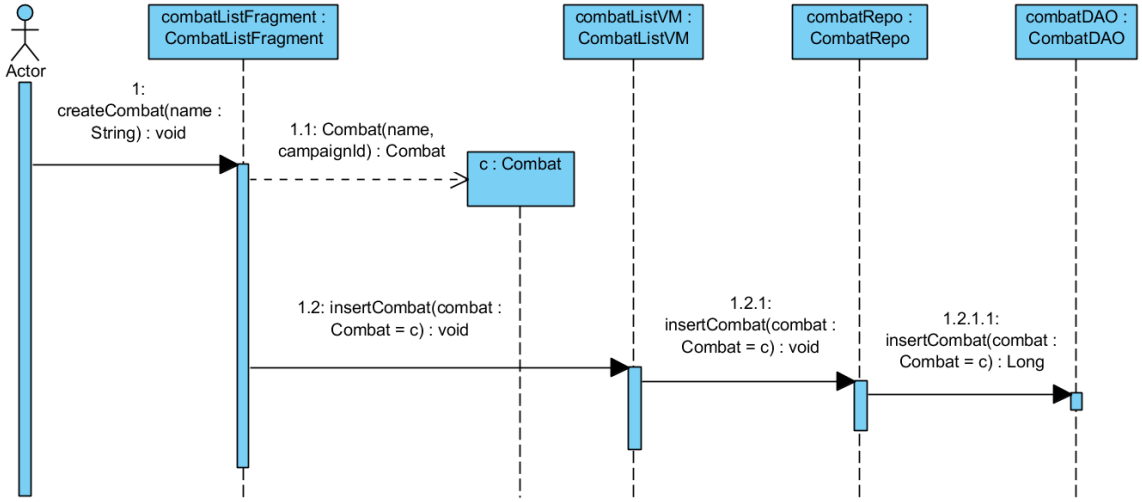

Figura 40 Diagrama de secuencia de crear combate – Elaboración propia

CU4.2: Eliminar combate

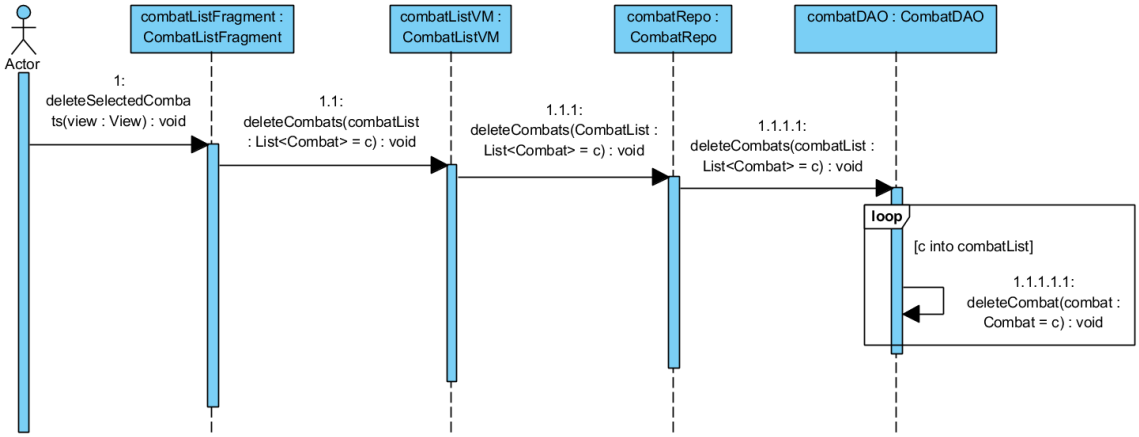

Figura 41 Diagrama de secuencia de eliminar combate – Elaboración propia

CU4.3: Modificar combate

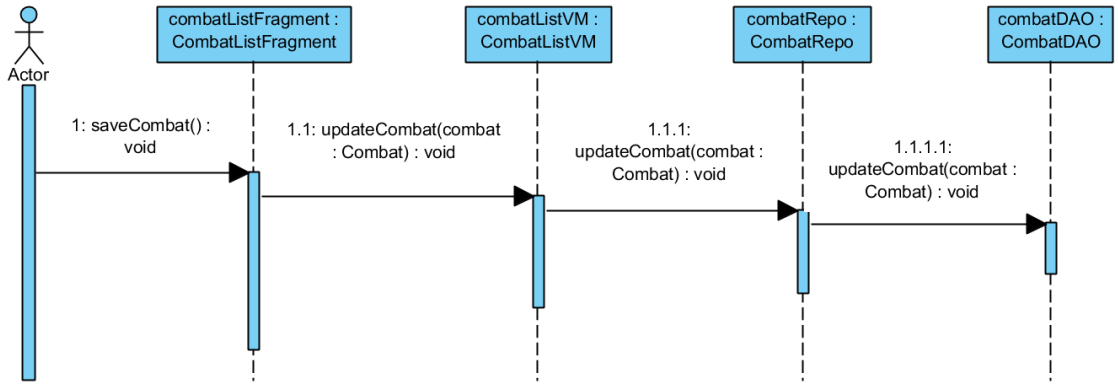

Figura 42 Diagrama de secuencia de modificar combate – Elaboración propia

CU4.4: Consultar combate

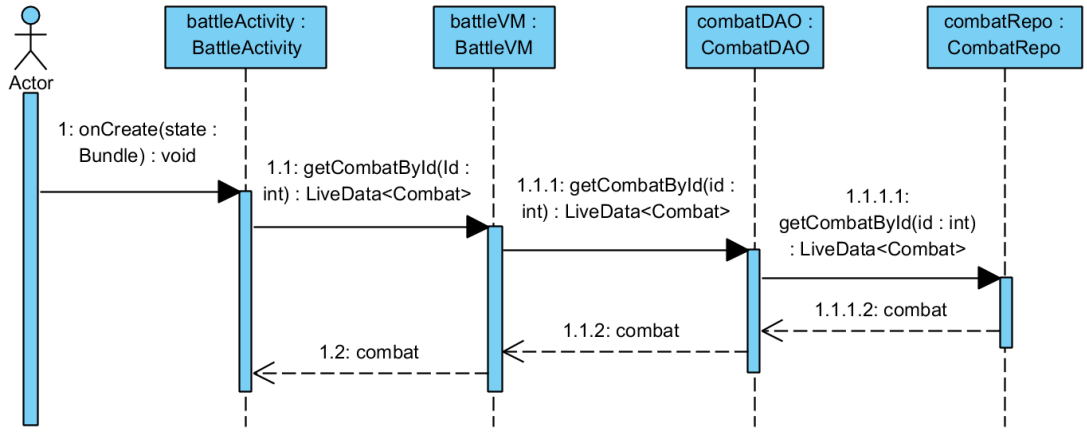

Figura 43 Diagrama de secuencia de consultar combate – Elaboración propia

CU4.5: Vincular personaje al combate

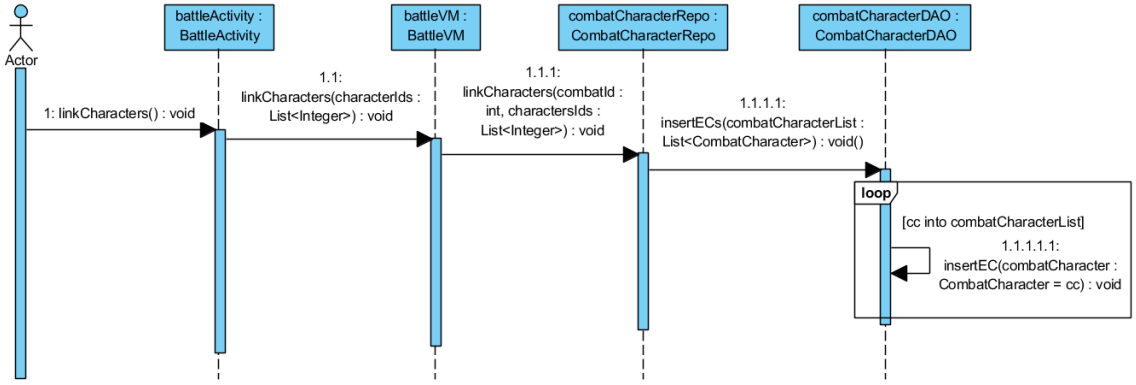

Figura 44 Diagrama de secuencia de vincular personaje al combate – Elaboración propia

CU4.6: Desvincular personaje del combate

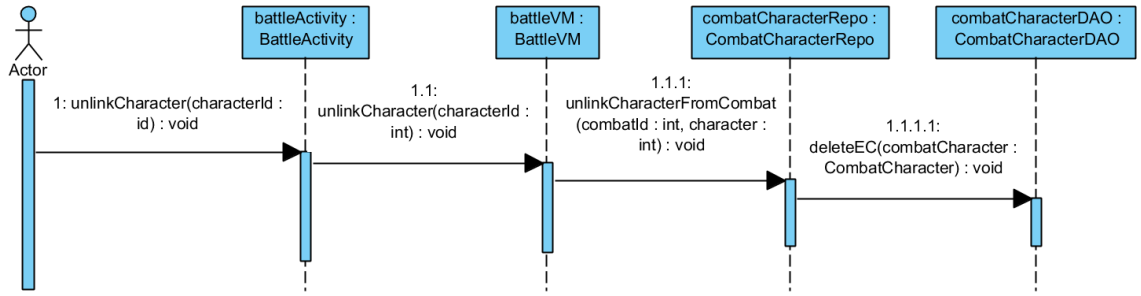

Figura 45 Diagrama de secuencia de desvincular personaje del combate – Elaboración propia

CU4.7: Resolver ataque

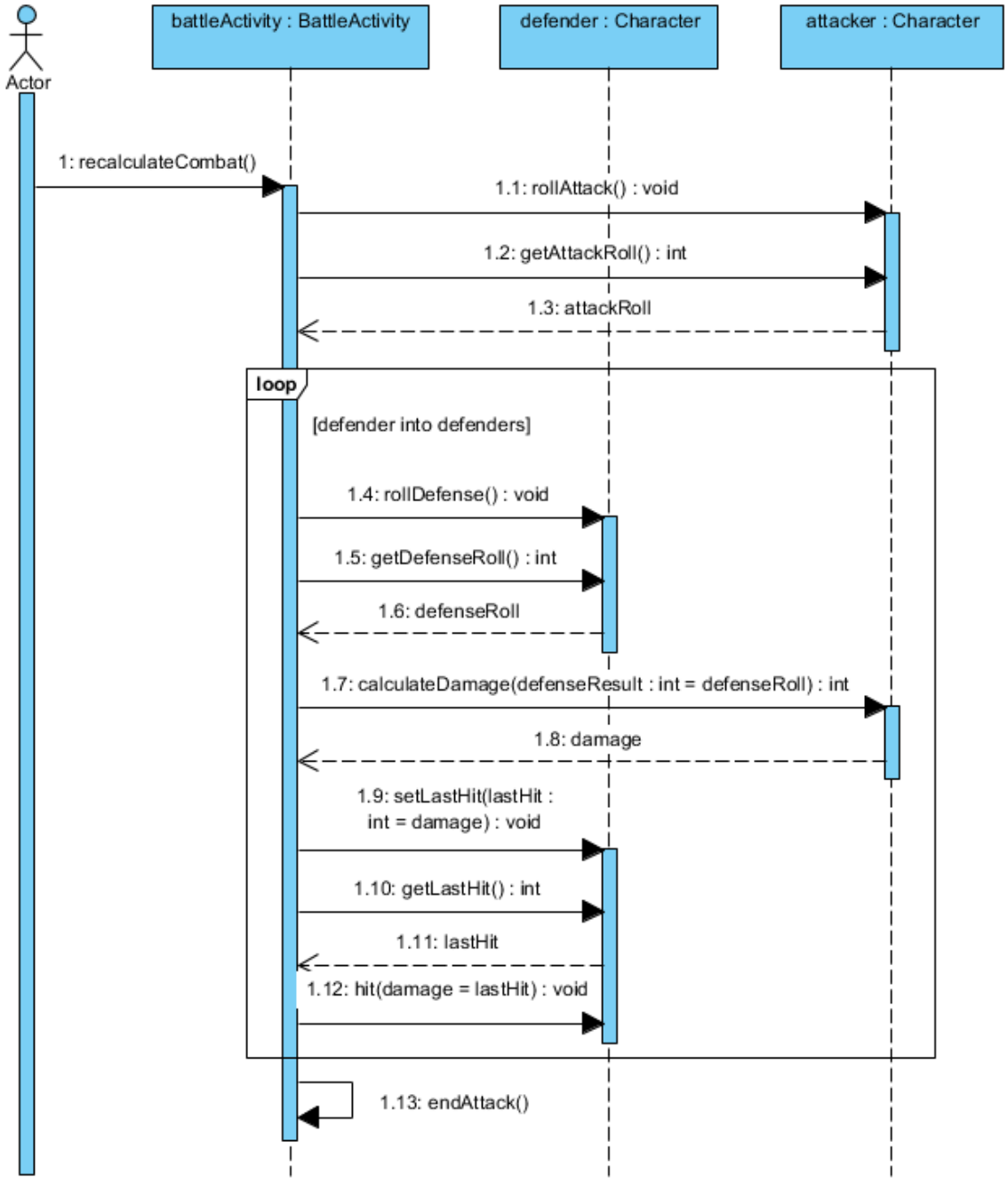

Figura 46 Diagrama de secuencia de resolver ataque – Elaboración propia

El diagrama anterior es el diagrama que representa el cálculo del combate, y es por eso que es el más complejo. Las acometidas en ánima se pueden dividir en 6 fases:

- **Cálculo de la tirada de ataque:** En esta fase se tira un dado de cien caras (dos dados de 10, uno para las decenas y otro para las unidades) y el resultado se suma a la habilidad de ataque del atacante. Esta cifra se llamará tirada de ataque.

- **Cálculo de la tirada de defensa:** Por lo general, al igual que con la tirada de ataque, se tira un dado de cien caras y se suma a la habilidad de defensa del defensor, sin embargo, en Anima existen ciertas criaturas que no se defienden de las acometidas, sino que simplemente tienen una cantidad de vida enorme y reciben el daño de lleno, en dichos casos su tirada de defensa es automáticamente cero.

- **Modificadores de ataque:** Dependiendo de una gran variedad de circunstancias, el atacante puede ver mejorada o empeorada su habilidad de ataque. Por ejemplo, si un arquero quiere acertar a alguien con niebla denas, le resultará mucho más complejo que hacerlo en un día claro y soleado. Es por ello que habrá que restar o sumar a la habilidad de ataque los bonificadores y penalizadores que correspondan a la situación en la que se encuentra.

- **Modificadores de defensa:** Al igual que con los de ataque, hay ciertas circunstancias que modifican la tirada de defensa, por lo que también se reducirá o aumentará dependiendo de la situación en la que se encuentre el defensor.

- **Cálculo del daño:** Una vez tenemos las tiradas de ataque y defensa con sus modificadores, se resta la tirada de defensa a la tirada de ataque y el resultado es el porcentaje de daño que recibe el defensor. Para calcular los puntos de vida que pierde es necesario multiplicar el daño del arma del atacante por el porcentaje. Por ejemplo si la diferencia entre ataque y defensa es 25, y el daño del arma es 40, el defensor recibiría 10 puntos de daño. Si la diferencia es cero o inferior, el defensor no sufre daño.

- **Reducción de vida:** Una vez tenemos el daño, solo queda restarlos a los puntos de vida actuales.

En nuestro caso, hemos agrupado el cálculo del ataque con los modificadores de ataque e igual con la defensa. En el diagrama, estas dos fases están representadas con `rollAtack()` y `rollDefense()` respectivamente. `calculateDamage(int defenseResult)` calcula el daño que luego se guarda con `setLastHit()`. El método `setLastHit()` está separado del método `hit(int damage)` para así poder separar la fase del cálculo del daño y la de reducción de vida. El método `hit(int damage)` reduce la vida del defensor.

CU5.1: Crear consumible

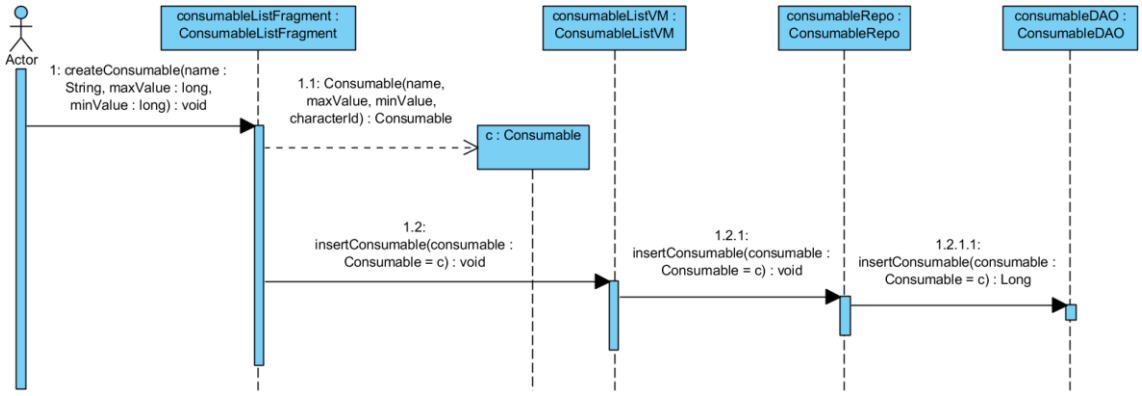

Figura 47 Diagrama de secuencia de crear consumible – Elaboración propia

CU5.2: Eliminar consumible

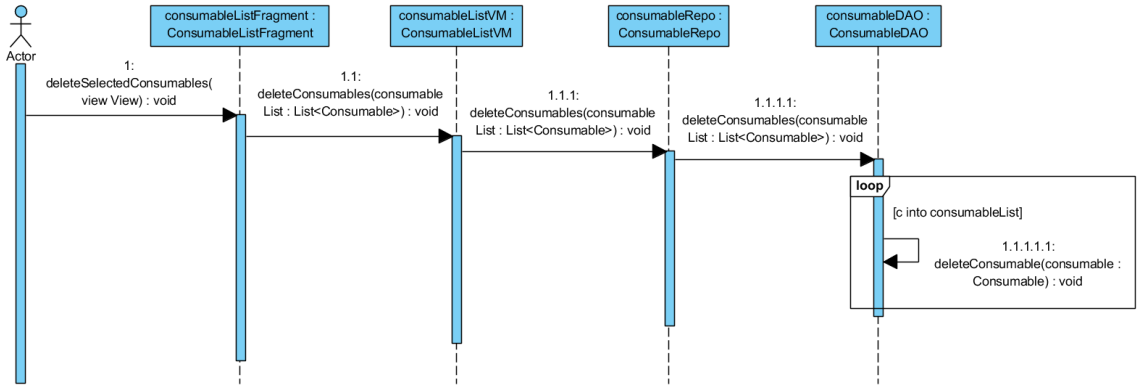

Figura 48 Diagrama de secuencia de eliminar consumible – Elaboración propia

CU5.3: Modificar datos del consumible

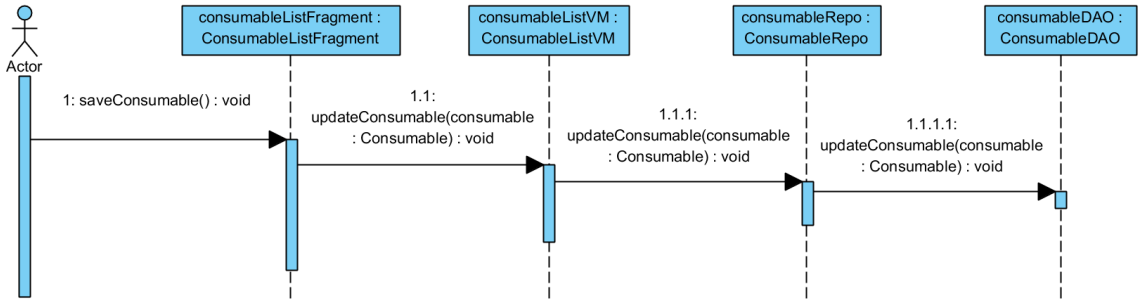

Figura 49 Diagrama de secuencia de modificar datos del consumible – Elaboración propia

CU5.4: Consultar datos del consumible

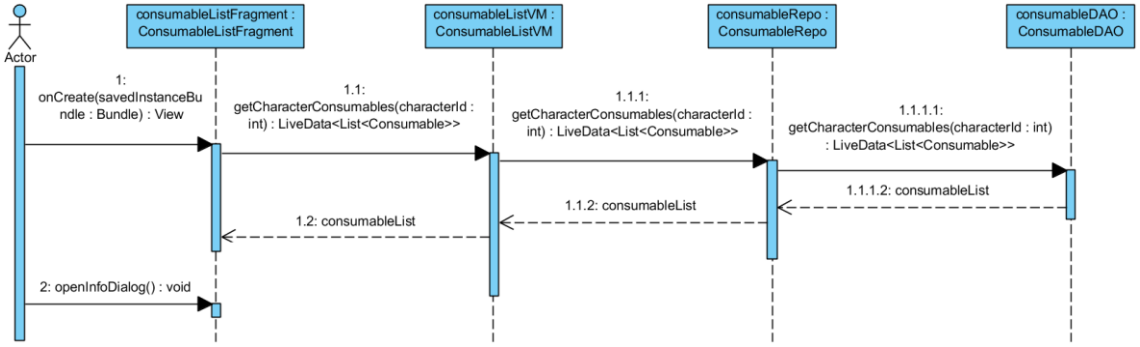

Figura 50 Diagrama de secuencia de consultar datos del consumible – Elaboración propia

4.3. Tecnología utilizada

A continuación, se listarán y explicarán las diferentes tecnologías y herramientas que se han utilizado durante el desarrollo de CHIMERA.

4.3.1. Android Studio

Sin lugar a dudas, Android Studio es el IDE más utilizado a la hora de desarrollar en Android. Basado en el potente IntelliJ, este IDE no tiene competidores, siendo el entorno oficial para desarrollo de aplicaciones Android.

4.3.2. Color Tool (Material.io)

Material.io es una página web centrada en el *Material Design*, en la cual se puede encontrar varios consejos y herramientas que se pueden necesitar a la hora de diseñar las interfaces de una aplicación. En concreto, se ha utilizado la herramienta *Color Tool*. Esta herramienta nos proporciona de forma rápida los diferentes colores recomendados para utilizar en una aplicación; con elegir los colores principales, *Color Tool* nos calcula automáticamente los colores derivados y los colores de texto que podamos necesitar en nuestra aplicación.

4.3.3. Draw.io

Se trata de una página web de dibujo de diagramas. No es tan potente como Visual Paradigm y no permite vincular los diferentes diagramas, sin embargo, si se quiere hacer un diagrama aislado rápidamente da muy buenos resultados. Además, estéticamente es simple y elegante. Es la herramienta que se ha utilizado para dibujar el diagrama de la arquitectura.

4.3.4. Genymotion

Genymotion es un emulador de Android multiplataforma. Debido a que el emulador que viene por defecto en Android Studio daba varios problemas a la hora de ejecutarse y que el rendimiento no era tan bueno como el de Genymotion, se optó por este emulador de la compañía Genymobile.

4.3.5. Gimp

Se trata de un programa de edición de imágenes es libre, de código abierto y tiene versión para todos los sistemas operativos. Se centra en la edición de imágenes rasterizadas, aunque también permite trabajar con gráficos vectoriales. Es el programa que se ha utilizado para realizar las imágenes que no son gráficos vectoriales.

4.3.6. Git

Git es un programa de control de versiones que nos permite llevar un seguimiento y gestionar los diferentes cambios que sufre nuestra aplicación. Es un programa indispensable en los entornos de desarrollo colaborativo; y en nuestro caso, aunque no haya más que un desarrollador, es una opción interesante para poder retroceder en las versiones de la aplicación.

4.3.7. GitHub

GitHub es una página web para alojar proyectos utilizando Git. Es una de las mejores opciones para poder guardar remotamente y de forma gratuita una aplicación. Es sin

duda la página web número uno a la hora de almacenar proyectos, siguiéndola de lejos Bitbucket y Gitlab.

4.3.8. GitKraken

GitKraken es una aplicación de escritorio que nos permite hacer uso del control de versiones de Git de forma gráfica. Además, recientemente han añadido una funcionalidad que permite crear tableros Kanban, por lo que se decidió utilizarla también para la gestión ágil del proyecto.

4.3.9. Inkscape

Se trata de un programa de edición de gráficos vectoriales que al igual que Gimp, es libre, de código abierto y, además, tiene versión para todos los sistemas operativos. Este es el programa que se ha utilizado para la creación de los iconos de la aplicación.

4.3.10. Java

Java es uno de los principales lenguajes orientados a objetos que existen actualmente y una de las opciones más extendidas a la hora de desarrollar para Android. Existen otras opciones que se han empezado a popularizar (p.ej. Kotlin), sin embargo, la familiaridad con el Java y la poca experiencia con el resto, hizo que fuera la opción a elegir.

4.3.11. JUnit

Se trata de una serie de bibliotecas que tienen como finalidad facilitar la creación de pruebas automatizadas para aplicaciones basadas en Java; forma parte de la familia de entornos de *testing* conocida como XUnit que se originó con JUnit. JUnit ha sido de gran relevancia para el desarrollo software dirigido por *tests*.

4.3.12. MockFlow

MockFlow es una página web para el diseño de prototipos que se centra sobre todo en los prototipos web. CHIMERA se trata de una aplicación Android, pero no hubo ningún problema para encontrar los elementos necesarios y adaptarlos a nuestro caso.

4.3.13. Office Timeline Online

Es una página web que permite la creación de líneas de tiempo y planes de trabajo. Es la herramienta que se utilizó para diseñar el diagrama de Gantt.

4.3.14. Room

Como se ha mencionado anteriormente, para la implementación de la base de datos se ha utilizado la librería Room. Se trata de una librería muy potente que permite montar una base de datos en cuestión de minutos. Para las aplicaciones que mantienen la base de datos en la memoria del dispositivo, es probablemente la opción más extendida.

4.3.15. Visual Paradigm (Community Edition)

Se trata de un programa para la creación de diagramas UML. Permite crear cualquier diagrama necesario para el desarrollo de una aplicación y permanecen vinculados entre sí para facilitar la modificación progresiva de los mismos. La versión *Community* de esta herramienta es la que se ha utilizado para la creación de los diagramas.

5. Desarrollo de la solución

5.1. Organización

Como se mencionó anteriormente, para organizar el proyecto se decidió utilizar Scrum, y los *sprints* fueron de dos semanas, sin embargo, hubo algunas excepciones. A continuación, se comentarán cada uno de los *sprints* y se expondrá el trabajo realizado en cada uno.

5.1.1. Sprint 0

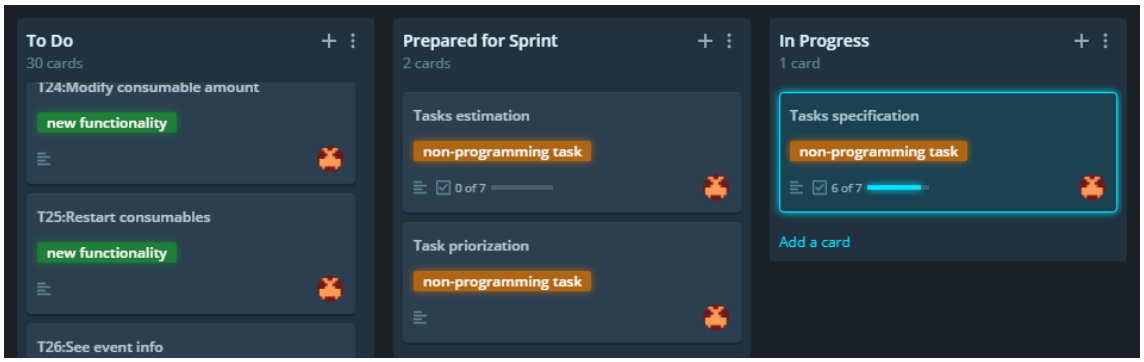

Figura 51 Sprint 0 – Elaboración propia

Antes de empezar con la implementación de las funcionalidades, se realizó un sprint cero en el que se preparó todo el entorno de desarrollo y en el que se introdujeron las tareas en el *backlog*. En nuestro caso utilizamos el tablero Kanban de GitKrakenGlo y creamos cuatro columnas: *To Do*, *Prepared for Sprint*, *In Progress* y *Done*.

La columna *To Do* hace la función de *backlog* y es donde se introducen todas las posibles tareas. Además, cuando se introducen en dicha columna, se les proporciona una primera estimación en *Story Points* y se ordenan por prioridad. Antes de empezar el sprint, se especifican en detalle las tareas que tienen más prioridad, se realiza una segunda estimación más precisa y se introducen en la columna *Prepared for Sprint*. Una vez empieza el *sprint* la forma de proceder es mover las tareas a la columna *In Progress* mientras se estén implementando y cuando se terminen, colocarlas en la columna *Done*.

5.1.2. Sprint 1

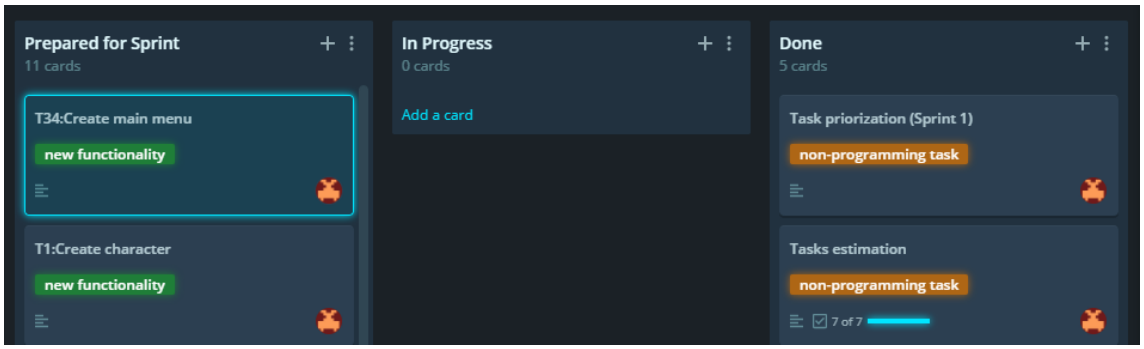

Figura 52 Sprint 1 – Elaboración propia

En el *sprint* 1 se montó la arquitectura de la aplicación, se creó la base de datos en Room y se implementaron las tareas relacionadas con la gestión de personajes y la gestión de consumibles. Una vez acabó el *sprint* los usuarios probaron la aplicación y proporcionaron retroalimentación, que usamos para especificar el siguiente *sprint*. Es importante remarcar que la elección de tareas permitió obtener un mínimo producto viable que permitía a los usuarios llevar la cuenta de los elementos consumibles de sus personajes.

5.1.3. Sprint 2

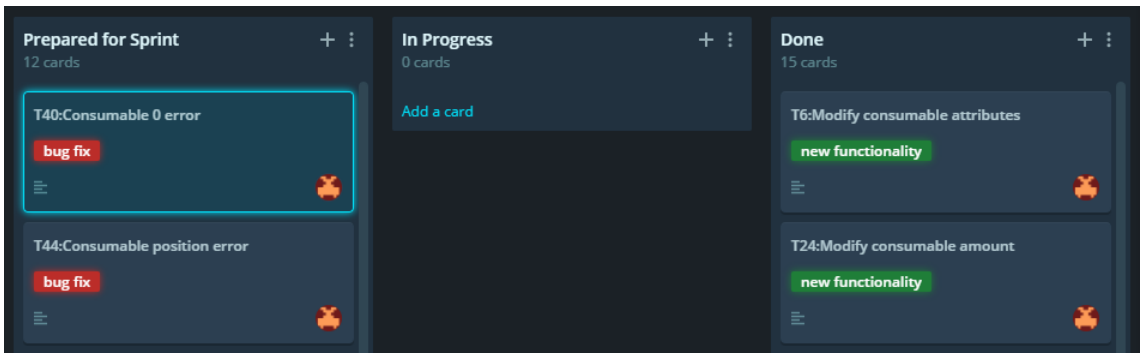

Figura 53 Sprint 2 – Elaboración propia

En el *sprint* 2, utilizando la retroalimentación obtenida en el sprint 1, se determinó que el combate sería el siguiente aspecto a implementar. Como se puede observar, el combate se adelantó hasta el *sprint* 2; esto fue así debido a que después de recoger la retroalimentación de los usuarios se determinó que el combate era más prioritario que el resto de los aspectos. Debido a que el combate de Anima es extenso y que surgieron algunos errores menores en el *sprint* 1, solo se pudo implementar el combate en este *sprint*. El mínimo producto viable obtenido, permitía a los usuarios calcular el orden de acción de los contendientes y calcular el resultado de una acometida de manera automática.

5.1.4. Sprint 3

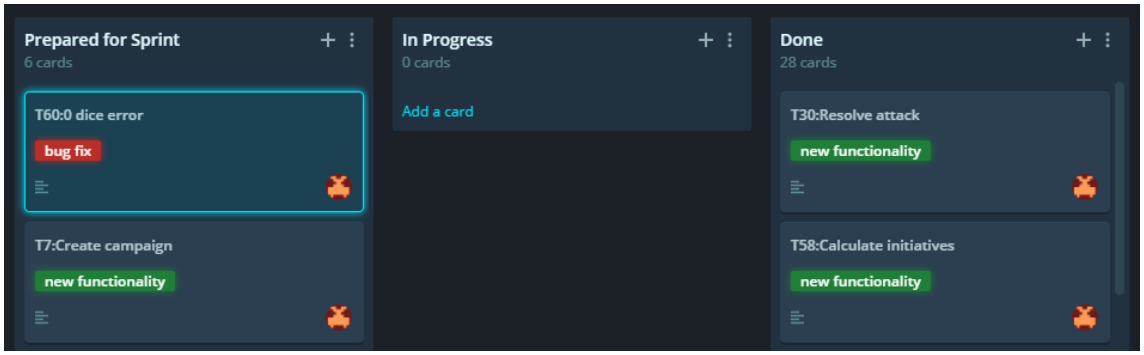

Figura 54 Sprint 3 – Elaboración propia

La retroalimentación del *sprint* 2, puso en evidencia varios problemas de usabilidad en la funcionalidad del combate y varios errores, por lo que gran parte del sprint se dedicó a resolverlos. También se decidió implementar pruebas unitarias para reducir la cantidad de errores de lógica. El resto del *sprint* se dedicó a una reestructuración de la base de datos para permitir la gestión de campañas.

5.1.5. Sprint 4

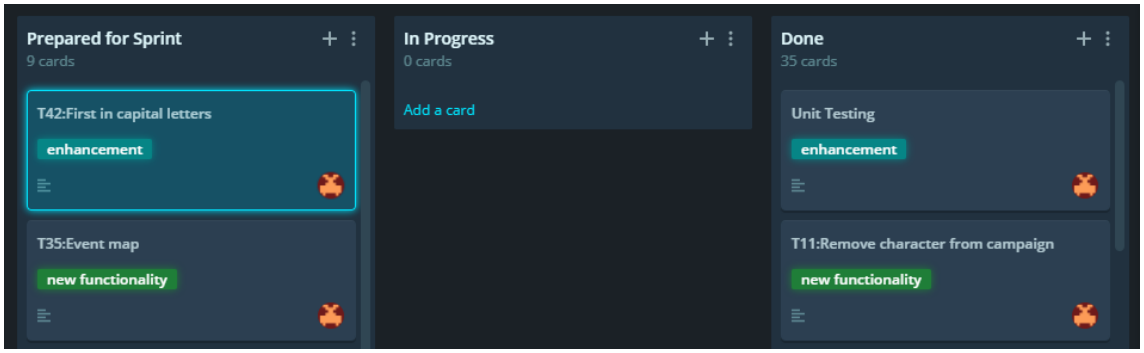

Figura 55 Sprint 4 – Elaboración propia

Por razones personales los usuarios con los que se hacían las pruebas y las revisiones no estuvieron disponibles la semana en la que finalizaba el *sprint*, por lo que tuvo que adelantarse una semana el desarrollo. Gracias a las pruebas unitarias, en este *sprint* no se tuvo que resolver ningún error, lo que permitió poder implementar toda la gestión de eventos en una semana. En este sprint se completó todo el desarrollo de nuevas funcionalidades que se había previsto, cumpliendo satisfactoriamente con los objetivos impuestos.

5.1.6. Sprint 5

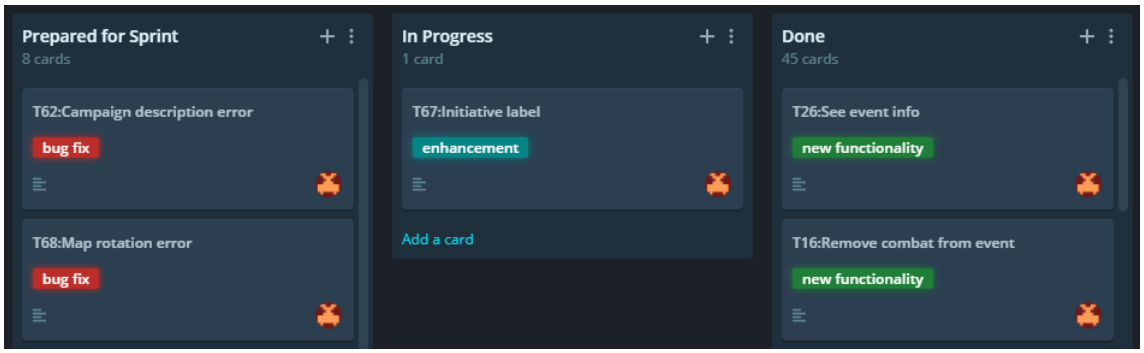

Figura 56 Sprint 5 – Elaboración propia

El *sprint* 5 finalizó también en una semana y se dedicó exclusivamente al *refactoring* y a la corrección de errores.

5.2. Problemas e implementaciones relevantes

Como se ha mencionado en la introducción, al empezar el proyecto, Android era un completo desconocido por lo que los problemas y las dificultades técnicas que se han sufrido son incontables. Sin embargo, a continuación se expondrán únicamente las que más relevancia han tenido en este proyecto:

5.2.1. Elección de la arquitectura

Debido a la inexperiencia en este tipo de tecnologías, en un principio se intentó simular la arquitectura clásica que presentan las aplicaciones de escritorio de java (presentación, lógica, persistencia), sin embargo, Android presentaba algunas peculiaridades que con una arquitectura tradicional no quedaban muy claras; estas peculiaridades se centraban sobre todo en el apartado de la presentación, ya que el ciclo de vida que seguían las `Activity` y los `Fragment` se desconocía.

Toda la información sobre la arquitectura se obtuvo de un canal de YouTube llamado Coding in Flow[18] que se centra en explicar distintos aspectos de Android. Concretamente, este canal tiene una serie de videos que explican paso a paso cada uno de los elementos de la arquitectura y como deben ser implementados, así como la puesta en marcha de la base de datos de Room.

5.2.2. Las listas

Uno de los aspectos más destacables de las aplicaciones para dispositivos móviles es la capacidad de detectar los gestos que hacen los usuarios sobre la pantalla, y algunas de las vistas que vienen por defecto ya gestionan dichos eventos automáticamente. Sin embargo, cuando se requiere poder detectar muchos gestos sobre el mismo elemento de la interfaz, las vistas por defecto ya no sirven y es necesario capturar manualmente dichos eventos.

En nuestro caso, el problema surgió al implementar las listas de la aplicación, donde teníamos que detectar cuatro eventos: el evento de pulsación simple para entrar en el perfil, el de pulsación larga para mostrar el menú contextual, el de arrastrar para hacer *scroll* de la lista y el de pulsación larga junto con arrastrar para cambiar de posición un elemento. Además, cuando cambias la posición de un elemento, debes poder seguir haciendo *scroll* por si la posición a la que quieres mover el objeto no está visible en ese momento.

Para diferenciar entre pulsación simple y pulsación larga, se utilizó un `GestureDetector` y extendimos la clase `SimpleOnGestureListener` sobrescribiendo los métodos `onSingleTapConfirmed(MotionEvent e)`, con el comportamiento deseado para la pulsación simple; y `onLongPress(MotionEvent e)` con el siguiente código:

```
@Override
public void onLongPress(MotionEvent e) {
    if (!getSelectModeEnabled()) {
        itemHolder.showPopup();
        itemHolder.setLongClicked(true);
    }
}
```

Dejando de lado la comprobación de si nos encontramos en el modo de selección, lo que indica el código es que se muestre el cuadro contextual con el método `showPopup()` (que es lo que queremos hacer si hay una pulsación larga) y que pongamos a *true* el valor de la variable `longClicked` del `ItemHolder`. Esta variable, se utiliza después para determinar si el usuario ha hecho una pulsación larga sobre el elemento y de ser afirmativo a partir de ese momento se comprobará si la posición inicial del dedo es muy diferente de la actual. Esta comprobación se realiza para saber si el usuario está intentando mover el elemento de posición y de ser así se empieza un evento de arrastre como se muestra en el siguiente código:

```
boolean dragOnX = Math.abs(downX - event.getX()) > DRAG_THRESHOLD;
boolean dragOnY = Math.abs(downY - event.getY()) > DRAG_THRESHOLD;

if (longClicked && (dragOnX || dragOnY)) {
    popup.dismiss();

    flyingItemPos = getAdapterPosition();

    final ClipData clipData = ClipData.newPlainText("", "");
```

```
    View.DragShadowBuilder shadowBuilder = new
        View.DragShadowBuilder(v);

    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
        v.startDragAndDrop(clipData, shadowBuilder, v, 0);
    } else {
        v.startDrag(clipData, shadowBuilder, v, 0);
    }

    v.setVisibility(View.INVISIBLE);
} else {
    return false;
}
```

Por último, la parte más compleja fue conseguir que los elementos cambiaran de posición cuando el usuario los arrastra por la pantalla. Esto está implementado por defecto en la clase *RecyclerView*, sin embargo, como debíamos capturar muchos más eventos aparte del arrastre, tuvimos que implementarlo por nuestra cuenta.

Lo primero fue implementar un método que, dependiendo de las coordenadas del dedo, calculara la posición donde debería caer el elemento. La mayoría de los problemas de dicho método surgieron a causa del *scroll*, ya que cuando llega al final no devuelve bien el desplazamiento vertical de la vista. Para solucionarlo se realizó lo siguiente:

```
float scrollOffset = recyclerView.computeVerticalScrollOffset();
int row = (int) Math.floor((y + scrollOffset) / cellHeight);
if (!recyclerView.canScrollVertically(1)
        && recyclerView.canScrollVertically(-1)) {
    row = recyclerRows - (int) Math.ceil((recyclerView.getBottom() -
        y) / cellHeight);
}
```

De esta forma si se ha llegado al final, en vez de calcular la posición usando de referencia la parte superior de la vista se utiliza de referencia la inferior.

El último de los problemas relacionado con las listas fue detectar que el usuario quería hacer *scroll* mientras arrastraba un elemento. Esto lo conseguimos definiendo zonas de desplazamiento. Si el usuario mantiene un elemento en dicha zona, en vez de calcular la posición donde debería caer el elemento, lo que hace es desplazar la vista en la dirección deseada. Es importante mencionar que la zona de desplazamiento inferior hubo que recortarla horizontalmente para que el usuario pudiera soltar los elementos en el icono de la papelera y así borrarlos. Esto lo conseguimos de la siguiente manera:

```
final int scrollY = recyclerView.getScrollY();

final boolean hoverScrollDown = event.getY() - scrollY >
        recyclerView.getBottom() - 250;
final boolean hoverScrollUp = event.getY() - scrollY <
```

```
        recyclerView.getTop() + 250;
final boolean hoverDeleteArea = event.getX() > recyclerView.getRight()
        - 250;

if (hoverScrollDown && !hoverDeleteArea
        && recyclerView.canScrollVertically(1)) {
    recyclerView.scrollBy(0, 30);
} else if (hoverScrollUp && recyclerView.canScrollVertically(-1)) {
    recyclerView.scrollBy(0, -30);
} else {
        //Cambio de posición…
}
```

5.2.3. El mapa de eventos

Esta parte de la aplicación también supuso un reto, ya que redimensionar una vista y todos los elementos que dependen de ella no es una tarea trivial. Este comportamiento ya está implementado en algunas librerías, sin embargo, es solo para imágenes y no vistas completas. Para conseguirlo se utilizó como base la clase ZoomableLayout[19] de un usuario de Github llamado anorth. Hubo que hacer varias modificaciones a la clase para conseguir el comportamiento deseado, pero los cambios más importantes fueron dos: modificar el método *onScale(ScaleGestureDetector scaleDetector)* y añadir *offsetX* y *offsetY*.

El método *onScale(ScaleGestureDetector scaleDetector)* se modificó para que el zoom lo realizara desde el punto en el que se hace el gesto de pinza.

```
@Override
public boolean onScale(ScaleGestureDetector scaleDetector) {
    float scaleFactor = scaleDetector.getScaleFactor();
    Log.i(TAG, "onScale" + scaleFactor);
    if (lastScaleFactor == 0 || (Math.signum(scaleFactor) ==
                Math.signum(lastScaleFactor))) {
        float prevScale = scale;
        scale *= scaleFactor;
        scale = Math.max(MIN_ZOOM, Math.min(scale, MAX_ZOOM));
        lastScaleFactor = scaleFactor;

        float adjustedScaleFactor = scale / prevScale;
        float focusX = scaleDetector.getFocusX();
        float focusY = scaleDetector.getFocusY();
        dx += (dx - focusX + this.getWidth() / 2.0) *
                (adjustedScaleFactor - 1);
        dy += (dy - focusY + this.getHeight() / 2.0) *
                (adjustedScaleFactor - 1);
    } else {
        lastScaleFactor = 0;
    }
    return true;
}
```

Se tuvo que añadir también, dos atributos adicionales y gestionar los bordes del mapa ya que, cuando el usuario selecciona una imagen para que sea el mapa de la campaña, esta

puede no encajar perfectamente con la pantalla del dispositivo. Esto puede parecer trivial, pero no lo es ya que no es posible obtener estos valores de la vista que contiene a la imagen al no tener las mismas proporciones. No guardar estos valores ocasionaría que al girar la pantalla los eventos no se mantuvieran en la misma posición y por tanto quedarían desplazados en el mapa.

5.2.4. Arrastrar y soltar en Android Pie

El último problema que comentar tiene que ver con los eventos de arrastre de Android Pie. En la última versión de Android, los eventos de arrastre no funcionaban o realizaban acciones extrañas. Esto era debido a que en Android Pie, el método `getLocalState()`, que se utilizaba para obtener la vista que estaba siendo arrastrada, devuelve siempre vacío, haciendo que la aplicación fallara la mayoría de veces. Esto fue reportado como un bug, pero a día de hoy todavía no ha sido solucionado. La forma de solucionarlo fue gestionando por nuestra cuenta la vista que era arrastrada en una variable aparte.

5.3. Ejemplo de funcionamiento

Para mostrar el funcionamiento de CHIMERA, a continuación se muestran una serie de capturas en las que se utilizó la aplicación para modelar una campaña real. La campaña relata una historia en la que los protagonistas acaban topándose de lleno con una serie de creaciones nigrománticas que amenazan con destruir la aldea en la que se encuentran. En un principio no son conscientes de lo que está pasando, pero a medida que se desarrolla la historia se hace más evidente que una organización secreta está utilizando los alrededores para sus macabros experimentos.

La pantalla que se muestra a la derecha, es la lista de campañas que se encuentra al entrar en la zona del director.

Figura 57 Lista de campañas – Elaboración propia

Lo primero que podemos ovserbar al seleccionar una campaña, es la pantalla de detalles de la campaña. En ella se encuentra una breve descripción y una imagen del mapa de la campaña. La descripción se trata de un breve texto que sirve a modo de introducción, donde también se pueden introducir las notas que necesitemos disponibles durante toda la campaña como información del clíma, economía del país, cultura...

En la parte superior se encuentra el mapa de la campaña, si pulsamos en la imagen podemos entrar en la pantalla del mapa pero, como todas las acciones de la aplicación, tambien se encuentra en el menú desplegable de la esquina superior derecha de la pantalla.

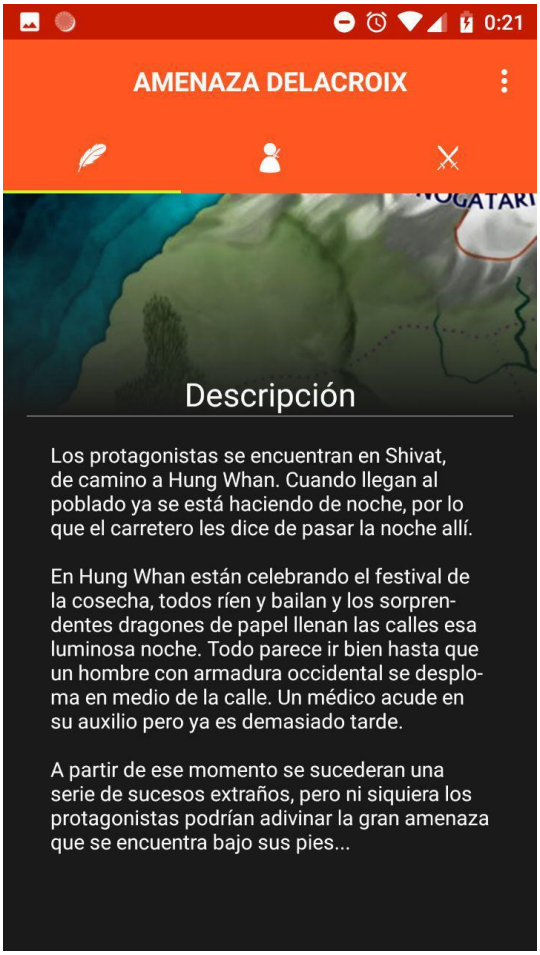

Figura 58 Detalles de la campaña – Elaboración propia

Si deslizamos el dedo hacia la izquierda, llegamos a la lista de personajes de la campaña. Esta pantalla se irá llenando de personajes a medida que los vayamos creando en los eventos. Adicionalmente se pueden crear desde esta pantalla para más tarde vincularlos a un evento.

Las listas mantienen la ordenación, por lo que es posible colocarlos en el orden que más convenga para la partida.

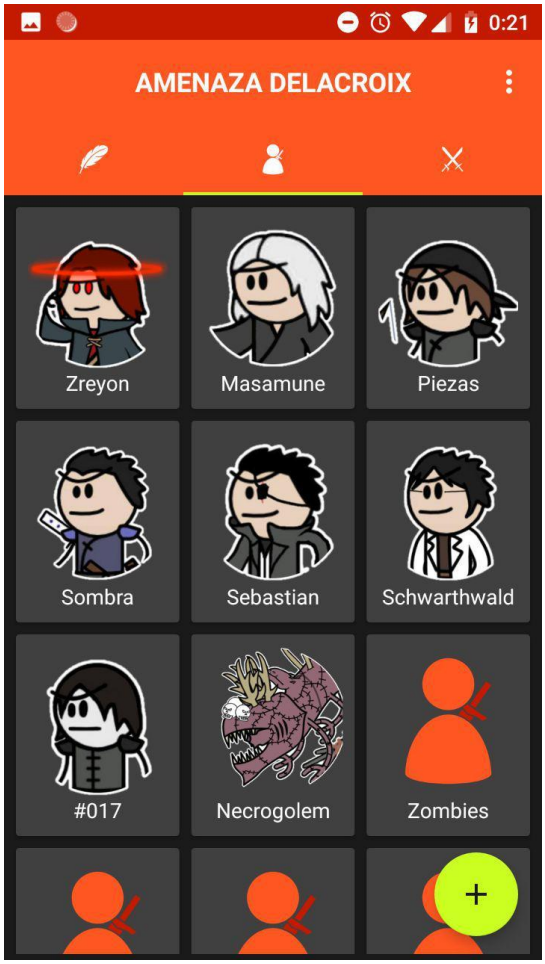

Figura 59 Lista de personajes de la campaña – Elaboración propia

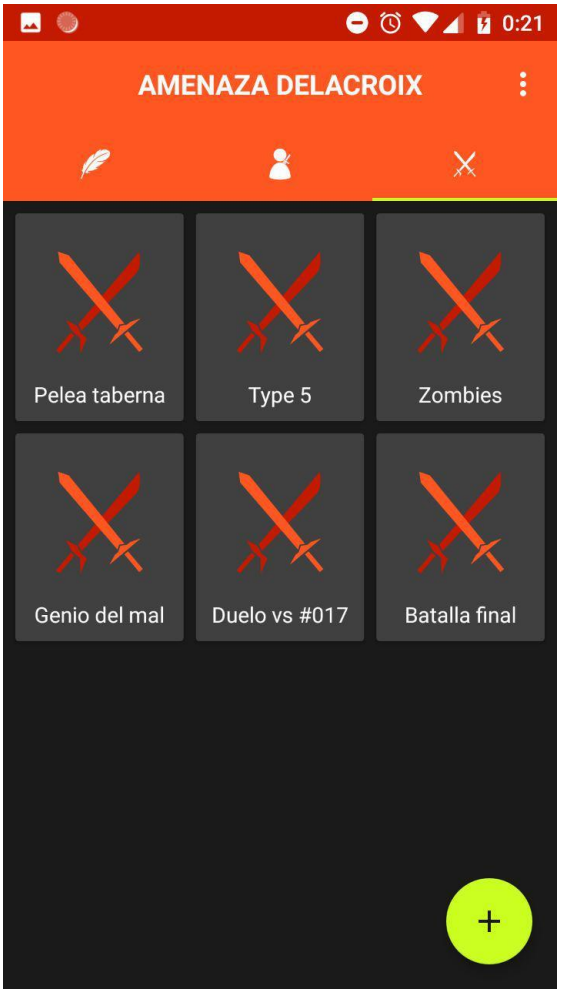

Figura 60 Lista de combates de la campaña – Elaboración propia

Si deslizamos una vez más a la izquierda, nos encontramos con la lista de combates. Al igual que los personajes, esta lista también se irá llenando de combates a medida que se creen en los eventos o se creen directamente desde esta pantalla.

Si volvemos a la pantalla de detalles y pulsamos la imagen de la parte superior, llegaremos a la pantalla del mapa de eventos de la campaña. En esta pantalla podemos colocar los diferentes sucesos que van a ocurrir durante la partida y así tener una representación más visual de lo que sucede.

El mapa es escalable, por lo que si tenemos una gran densidad de eventos podemos hacer *zoom* para observarlos mejor.

Figura 61 Mapa de eventos – Elaboración propia

Si pulsamos uno de los eventos, llegaremos a la pantalla de detalles del evento. En esta pantalla, al igual que la campaña, tenemos una zona para introducir todo lo que sucederá en dicho evento, desde descripciones del paisaje, hasta apariciones estelares de algún NPC.

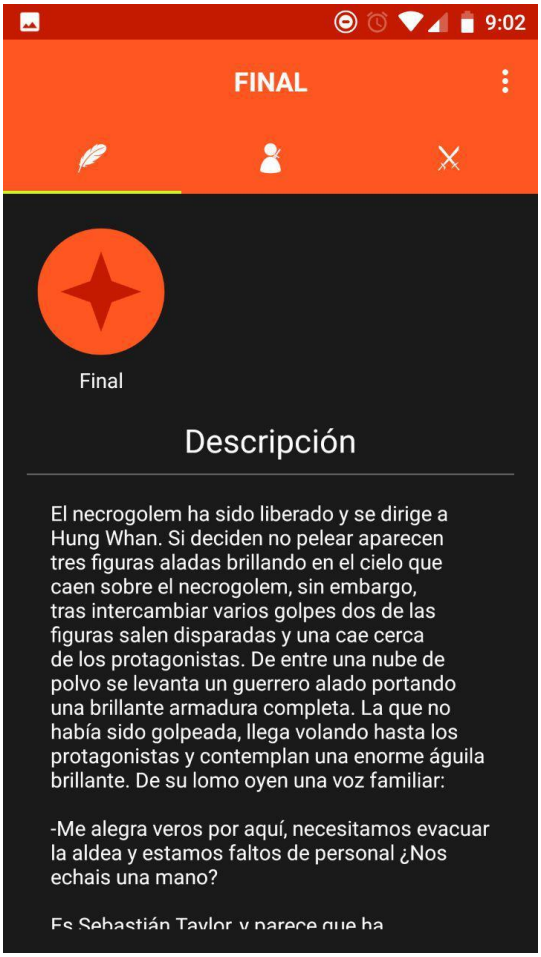

Figura 62 Descripción del evento – Elaboración propia

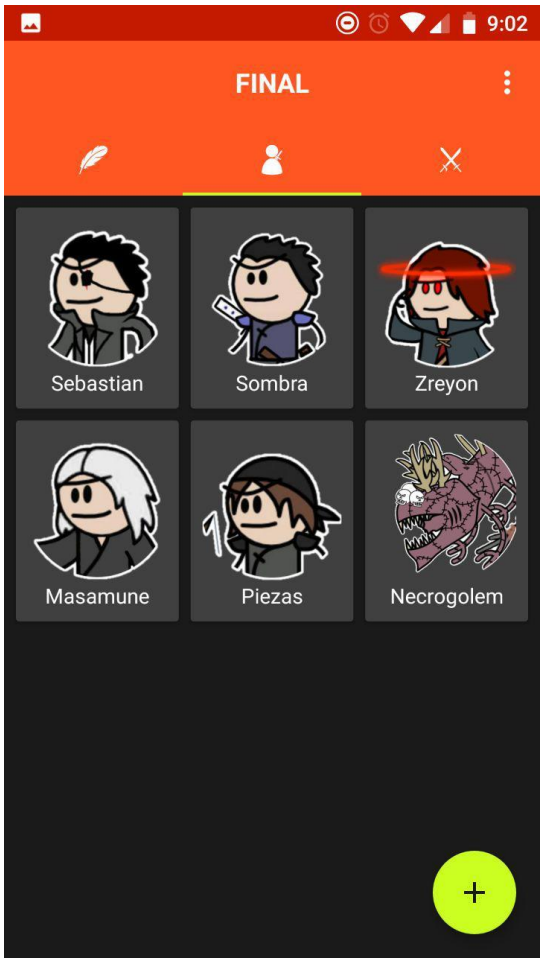

Figura 63 Lista de personajes del evento – Elaboración propia

Si deslizamos hacia la izquierda, como en la campaña, llegaremos a la lista de personajes que participan en el evento. Como se mencionó anteriormente, desde aquí es posible crear personajes y que automáticamente aparezcan en la lista de personajes de la campaña.

Si por el contrario se desea vincular un personaje ya existente a el evento, es posible hacerlo desde el menú desplegable que se encuentra en la parte superior derecha de la pantalla.

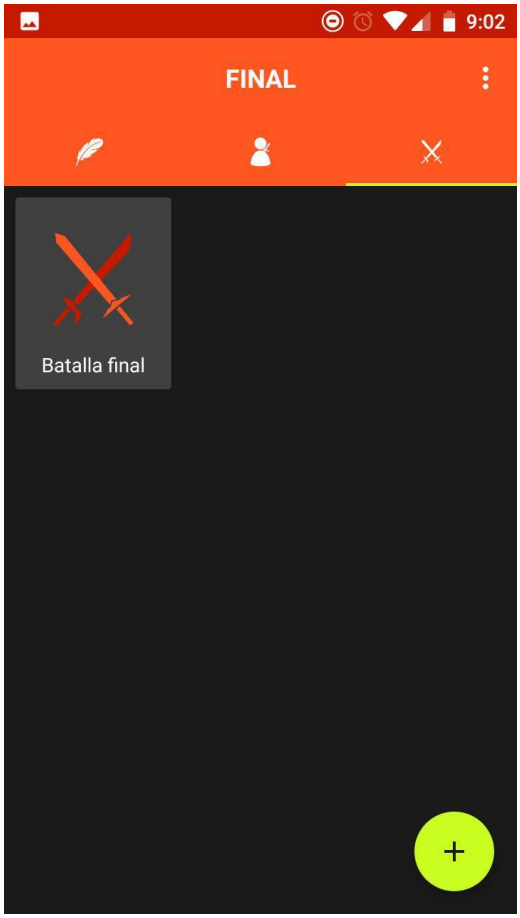

Figura 64 Lista de combates del evento – Elaboración propia

Por supuesto, al igual que la campaña, también hay una lista de combates en los eventos.

Si pulsamos sobre un combate, nos llevará a la pantalla de dicho combate, en la cual podremos calcular la iniciativa de los contendientes y calcular el resultado de los ataques. Para ello, primero deberemos vincular los personajes a un combate. Una vez vinculados, aparecerán en la columna de la iniciativa; a partir de este momento ya seremos capaces de calcular el orden de acción de los contendientes con pulsar en el botón de recalcular la iniciativa.

Una vez sepamos el orden de actuación, podemos seleccionar atacantes y defensores para calcular el resultado de una acometida. Para ello pulsamos sobre el personaje que queramos que sea el atacante y deslizamos hacia la derecha a los personajes que queramos que sean los defensores. El atacante aparecerá en el cuadrado rojo y los defensores aparecerán en la columna azul.

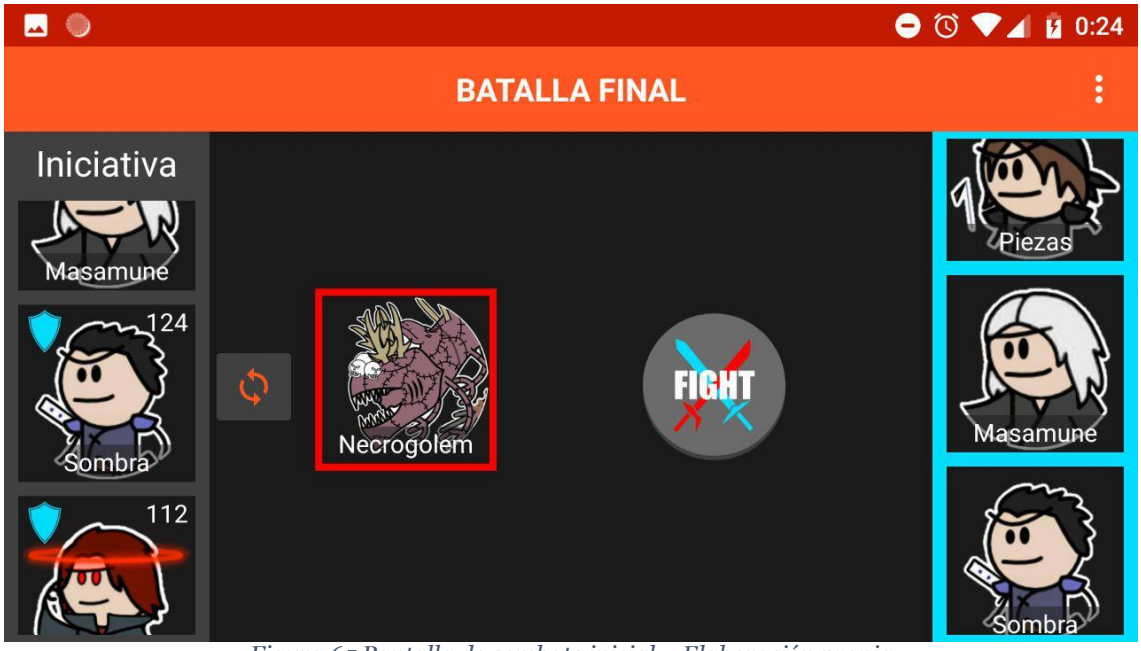

Figura 65 Pantalla de combate inicial – Elaboración propia

Cuando tenemos a todos los personajes en la posición que corresponde, ya podemos calcular la acometida. Siguiendo el sistema de Anima, el cálculo que realiza la aplicación consta de tres partes: cálculo de las habilidades de ataque y defensa, cálculo del daño y reducción de vida.

Cuando pulsamos el botón de combate por primera vez, realiza el cálculo del ataque y la defensa y cambia de forma para representar la siguiente acción a realizar. La habilidad de ataque final del atacante aparece en la esquina superior derecha de la celda del atacante y la habilidad de defensa final de los defensores, en la esquina superior izquierda.

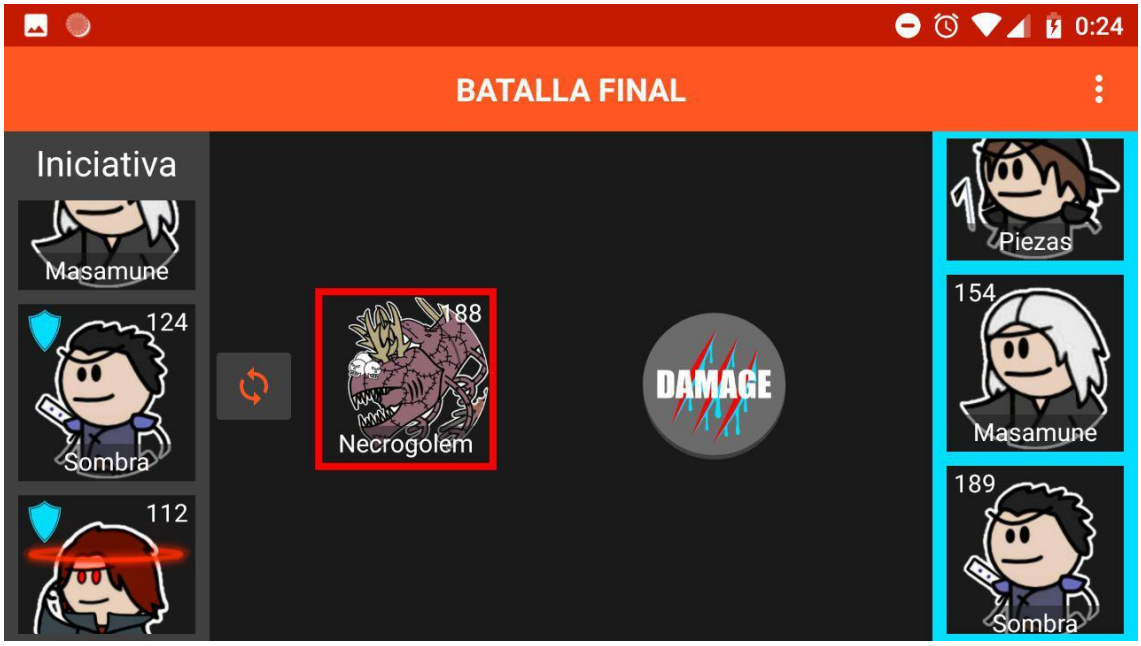

Figura 66 Pantalla de combate, primera pulsación – Elaboración propia

Si pulsamos sobre el botón nuevamente, realizará el cálculo del daño y sobre cada personaje aparecerá el daño que le va a causar el golpe. Es importante mencionar, que es posible retirar a un personaje de la lista de defensores en cualquier momento, por si por alguna razón no debería recibir el daño del ataque. Nuevamente cambia de forma para indicar la siguiente acción.

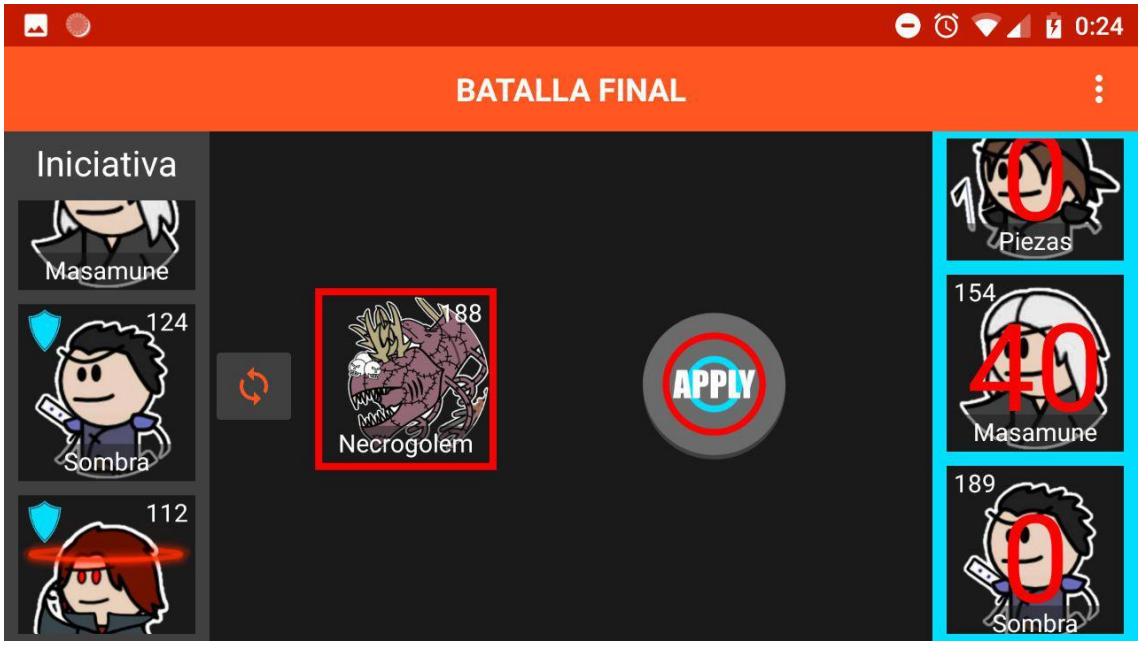

Figura 67 Pantalla de combate, segunda pulsación – Elaboración propia

Por último, si pulsamos una vez más sobre el botón, reducirá la vida de cada defensor en función al daño recibido.

Lo único que queda por mostrar es el apartado de los personajes, que es accesible tanto desde los propios personajes que se encuentran en las campañas del director como desde la zona del jugador que se encuentra en el menú principal de la aplicación.

Es importante mencionar que los personajes no son los mismos en la zona del director y en la zona del jugador, de hecho, tampoco son los mismos entre campañas. Esto es así debido a que no resulta lógico que los personajes de una campaña compartan la vida y los consumibles con los de otra campaña, ya que, en el caso de tener dos partidas simultaneas, los personajes no mantendrían estados consistentes.

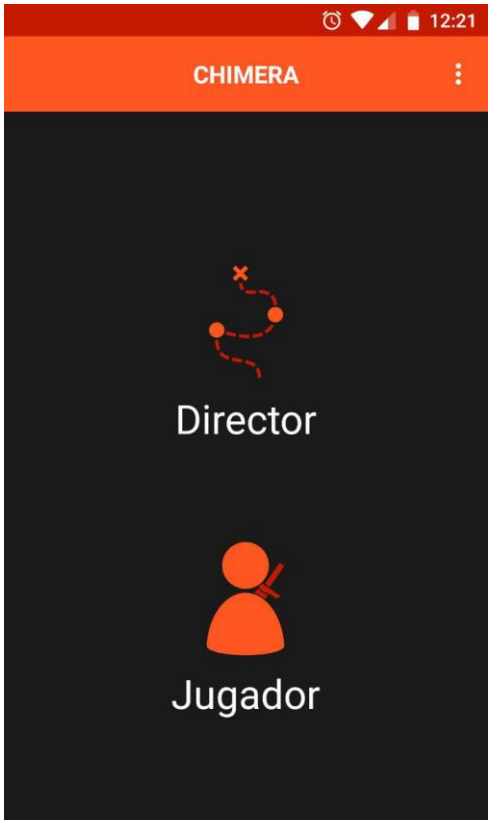

Figura 68 Menú principal – Elaboración propia

Si entramos en la zona del jugador, aparecerá la lista de personajes que tengamos para interpretar.

Esta lista funciona igual que las listas de la campaña y los eventos, por lo que también se pueden crear, borrar, modificar y ordenar los personajes.

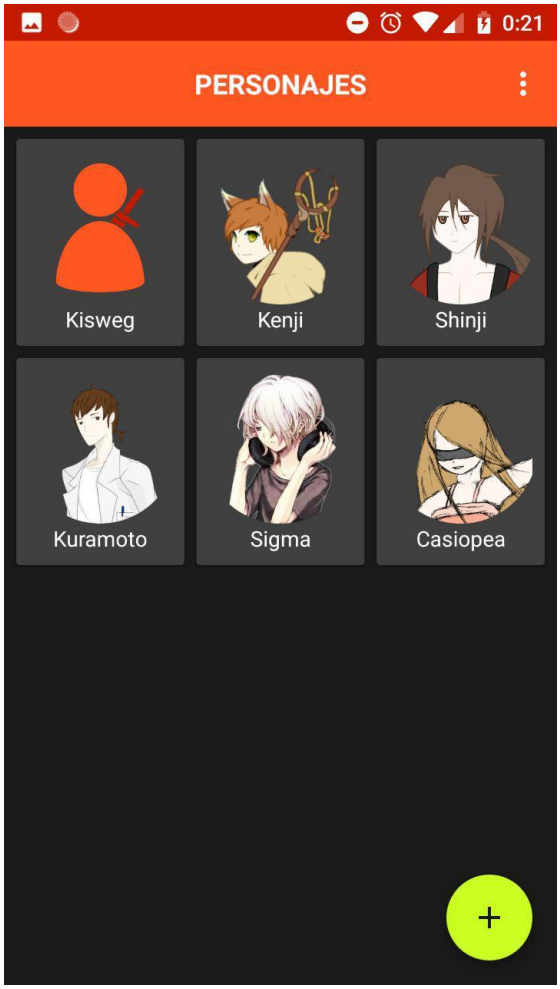

Figura 69 Personajes del jugador – Elaboración propia

Nuevamente, cuando pulsamos sobre un personaje aparece la pantalla de detalles en la que tenemos la foto de perfil, y una descripción donde se cuenta la biografía del individuo.

Figura 70 Detalles del personaje – Elaboración propia

Figura 71 Lista de consumibles – Elaboración propia

Si deslizamos a la izquierda, llegamos a la pantalla de consumibles. En esta pantalla podemos crear diferentes tipos de consumibles para representar elementos que iremos consumiendo y obteniendo durante la partida.

Los consumibles pueden representar una gran cantidad de cosas, desde el número de pociones curativas que poseemos, hasta la cantidad de magia de la que disponemos.

Por último, tenemos el perfil de combate, en el que un personaje puede definir los diferentes valores que serán utilizados para el cálculo del combate. También se podrán introducir modificadores temporales para no tener que modificar la base.

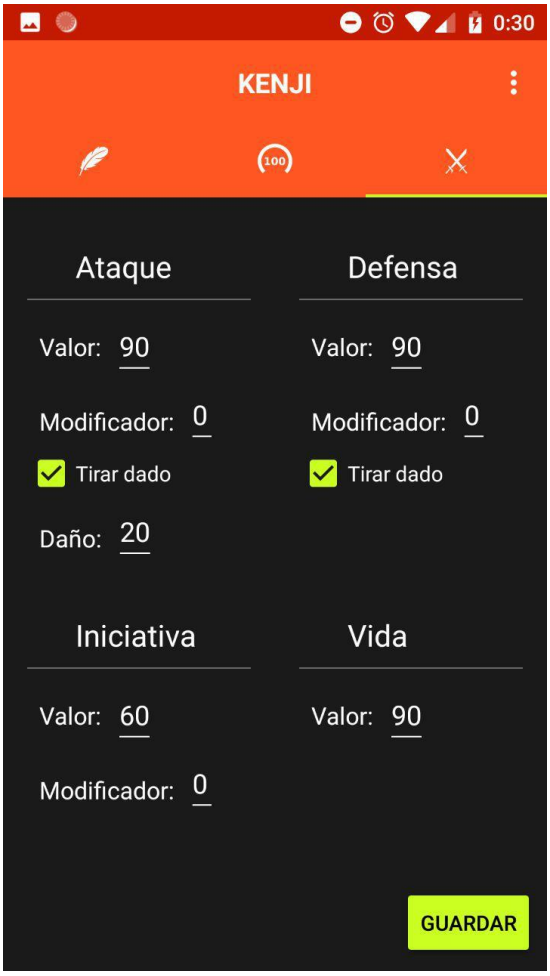

Figura 72 Perfil de combate – Elaboración propia

6. Pruebas

En cuanto a las pruebas de la aplicación se realizaron dos tipos de pruebas: pruebas unitarias y validación con usuarios reales.

6.1. Pruebas unitarias

Para las pruebas unitarias se decidió optar por hacer *tests* en JUnit. En nuestro caso no se ha realizado un desarrollo dirigido por pruebas, pero se consideró importante probar algunas de las clases más relevantes. JUnit permite la creación de pruebas de varios tipos, sin embargo, las pruebas unitarias eran suficientes para el propósito de este proyecto. En un principio también se planteó usar Jenkins para gestionar las pruebas, pero debido al gran número de problemas que presentó integrarlo con nuestro proyecto Android, se desechó esta opción.

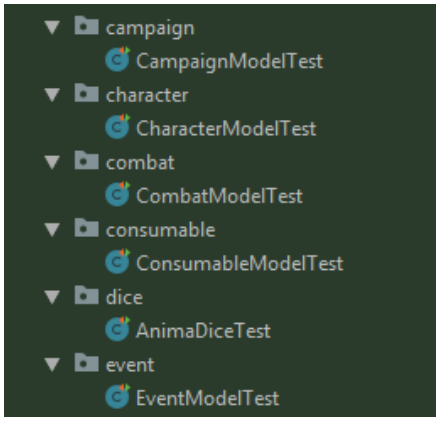

Figura 73 Pruebas JUnit – Elaboración propia

Las clases para las que se realizaron pruebas unitarias fueron los modelos de la lógica, los cuales son las clases con la lógica más crítica, sobre todo `Character`. Estas pruebas se empezaron a incluir a partir del *sprint* 2. Al principio del *sprint* 3 se implementaron 4 *tests* que probaban las clases `Character`, `Consumable`, `Combat` y `AnimaDice`. Al final del *sprint* 3, después de implementar las campañas, se implementaron las pruebas de la clase Campaign y se hicieron los ajustes pertinentes al resto. Por último, al final del sprint 4, se añadieron los *tests* de las funcionalidades de los eventos y se volvieron a hacer ajustes al resto de *tests*.

Test1: Campañas

Descripción	Las pruebas realizadas sobre las campañas se centraron en los métodos `equals(Object o)`, que comprueban si dos campañas son la misma (si comparten ID); y `contentsTheSame(Object o)`, que comprueba si todos y cada uno de los atributos de las campañas comparadas son los mismos.		
Sprint	3	Métodos	2

Test2: Personajes

Descripción	La clase `Character` es la clase con más métodos, y es la clase que mantiene más información, por lo que necesita también una mayor cantidad de pruebas. Los métodos probados están todos relacionados con las funcionalidades del combate a excepción de `contentsTheSame()` y `testEqualsSymetric()`. Este último es igual que el test que probaba `equals(Object o)`, a excepción de que también prueba el método `hashCode()`.		
Sprint	3	Métodos	7

Test3: Combates

Descripción	Para la clase `Combat` solamente es necesario probar los métodos `equals(Object o)` y `contentsTheSame(Object o)`, ya que todos los cálculos los realizan las clases de los jugadores.		
Sprint	3	Métodos	2

Test4: Consumibles

Descripción	Para los consumibles, a parte de los métodos `equals(Object o)`, `hashCode()` y `contentsTheSame(Object o)`, también hay que probar el cambio de los valores mínimo y máximo para que el valor actual no se quede en un estado inconsistente. Además, para el método `getValueFormated()` es necesario comprobar que el valor es devuelto con el formato correcto.		
Sprint	3	Métodos	5

Test5: AnimaDice

Descripción	Para probar la generación de tiradas de dados, se realizan miles de tiradas y se comprueba en cada una de ellas que el valor cumpla los requisitos. Estos *tests* pueden dar falsos positivos debido a que los métodos de `AnimaDice` dependen de números aleatorios, sin embargo, nunca dará falsos negativos.		
Sprint	3	Métodos	2

Test6: Eventos

Descripción	Igual que las campañas y los combates, para los eventos solamente se necesita probar los métodos `equals(Object o)` y `contentsTheSame(Object o)`.		
Sprint	4	Métodos	2

Concretamente, la prueba de la clase `AnimaDice` fue extremadamente útil para detectar errores, ya que, al tratarse de números aleatorios, surgían errores difíciles de reproducir que sin los *tests* hubiera sido complicado encontrar.

6.2. Validación con usuarios

Para las clases de la lógica se implementaron *tests* unitarios, sin embargo, para las clases de la presentación no tiene mucho valor implementar este tipo de pruebas, ya que lo más relevante en este tipo de clases es medir la usabilidad y la aceptación. Puesto que la usabilidad y la aceptación no son valores objetivos es necesario realizar pruebas con los usuarios para que sean ellos los que juzguen y aporten retroalimentación.

Nosotros optamos por el *test* SUS (System Usability Scale), que se trata de un formulario con diez preguntas que hay que contestar con un número del uno al cinco con una escala Linkert, siendo uno "Completamente en desacuerdo" y cinco "Completamente de acuerdo". Las preguntas impares son positivas mientras que las impares son negativas. Para facilitar la comprensión de las preguntas por parte de los usuarios, se tradujeron al castellano y se dispuso en forma de tabla (ver anexo).

Para calcular la puntuación del *test* hay que sumar las contribuciones de cada punto que puede ir del cero al cuatro. Para calcular la contribución de las preguntas impares es necesario restar uno al valor proporcionado por el usuario, mientras que, en las pares, la contribución es cinco menos el valor proporcionado por el usuario. Por ejemplo, en una pregunta par que se ha contestado con un 2 ("En desacuerdo") el resultado sería 5 – **2** = 3; y si en una pregunta impar se contesta con un 3 ("Neutral") el resultado sería **3** – 1 = 2.

Una vez tenemos la puntuación de cada *test* es necesario sumar los *tests* de cada sprint, hacer la media y sacar el porcentaje para que la puntuación sea sobre 100. Como la puntuación máxima es 40, para sacar el porcentaje es necesario multiplicar por 2,5. El resultado habrá que contrastarlo con la siguiente escala que nos indicará el estado de nuestra aplicación.

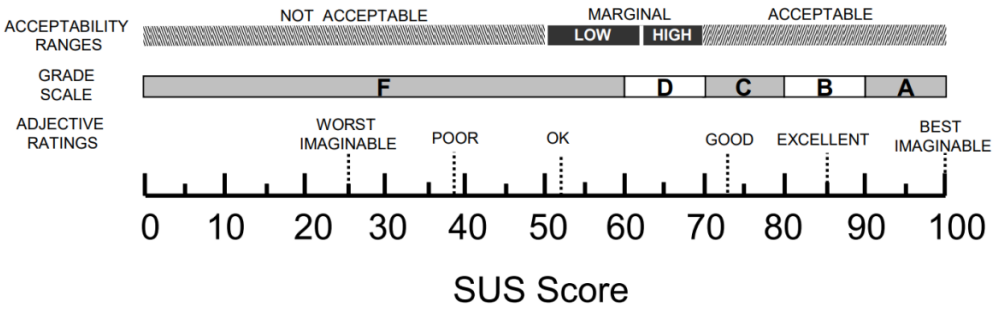

Figura 74 Escala SUS – Elaboración propia

Al final de cada sprint se subía una nueva versión de la aplicación a la tienda de Android y se les proporcionaba a los usuarios una hoja con la encuesta para obtener retroalimentación. En el sprint 5, puesto que se dedicó exclusivamente al *refactoring* y a

resolver errores, no se proporcionó a los usuarios ninguna encuesta. Los resultados por *sprint* fueron los siguientes:

	Sprint 1				Sprint 2				Sprint 3				Sprint 4			
Usuario / Pregunta	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	5	5	5	5	4	5	4	3	5	5	5	5	5	5	5	4
2	1	1	2	1	4	2	3	2	3	1	2	2	2	1	1	2
3	5	5	5	5	4	4	4	4	4	4	4	4	4	4	5	3
4	1	1	1	1	3	1	5	1	3	1	1	2	2	1	1	2
5	4	5	5	5	5	4	3	5	5	5	5	5	4	4	5	4
6	1	1	1	1	1	1	3	1	1	1	1	1	1	1	1	2
7	5	4	4	5	2	5	3	2	2	4	4	4	3	3	4	4
8	1	1	1	1	1	1	3	1	1	2	1	1	1	1	2	4
9	5	5	5	5	3	5	3	4	4	4	3	3	4	4	4	2
10	1	1	1	1	3	1	5	2	1	1	2	2	2	1	1	4
Total	39	39	38	40	26	37	18	31	31	36	34	33	32	35	37	23
Resultado	97,5				70				83,8				79,4			
Valoración	Insuperable				Bien				Excelente				Excelente			
Desviación	0,816				8,04				2,08				6,18			

Tabla 5 Resultados de las pruebas con usuarios (Usuarios) – Elaboración propia

En las casillas de la tabla queda representada la puntuación que le dio cada usuario a cada pregunta, siendo 5 "Completamente de acuerdo" y 1 "Completamente en desacuerdo".

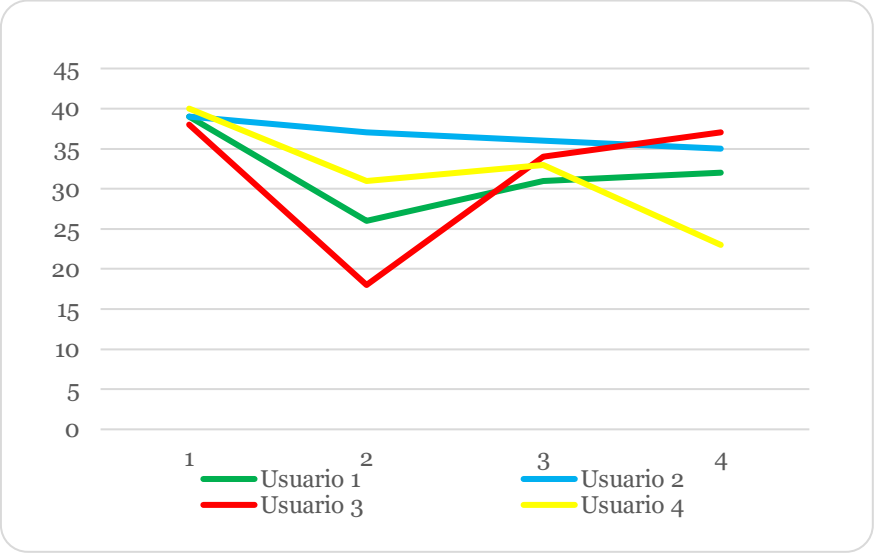

Figura 75 Gráfico con los resultados de las pruebas con usuarios – Elaboración propia

También se elaboró una tabla para mostrar el número de veces que cada pregunta recibió cada puntuación. La tabla es la siguiente:

	Sprint 1					Sprint 2					Sprint 3					Sprint 4				
Respuesta / Pregunta	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1					4			1	1	1					4					2
2	3	1					2	1			1	2	1			1	1			
3					4				3					4					2	
4	4					2				1	2	1	1			1	1			
5				1	3			1	1	1					4				2	
6	4					2		1			4					2				
7				2	2		1	1		1		1		3				2		
8	4					2		1			3	1				2				
9					4			1	1	1			2	2					2	
10	4					1	1			1	2	2				1	1			

Tabla 6 Resultados de las pruebas con usuarios (Respuestas) – Elaboración propia

Además de las encuestas, se realizaron dos sesiones de Anima de prueba en la que los usuarios comentaban en voz alta lo que opinaban de la aplicación durante la partida. Estos comentarios fueron vitales a la hora de recoger retroalimentación y muchos supusieron cambios importantes en la aplicación. Algunos de ellos se muestran a continuación:

La disposición con barras circulares para mostrar los consumibles fue un diseño acertado:

> *- "Mis dieces por eso." (haciendo referencia a la barra circular de los consumibles)*

La forma en la que se eliminaban lo enemigos no estaba lo suficiente claro y hubo que revisarla:

> *- "Uf, me ha costado. Pero como para no haberlo encontrado jamás si no me lo dices, eh" (haciendo referencia a eliminar combate)*

Hubo que replantearse la funcionalidad del combate para permitir retirarlos a mitad de una acometida:

> *- "No veo útil el botón de apply si no das la opción de no-apply."*

Mantener las campañas en la cima de la jerarquía de la arquitectura fue una decisión acertada:

> *"Me gusta que al crear personajes en un evento se añadan también a los de la campaña, pero no al revés."*

Permitir poner imágenes a los personajes causó buenas impresiones:

> *"Me gusta lo de cambiar los dibujos por fotos."*

Se retiró el botón para volver directamente al menú principal:

> *"¿Por qué está el botón de home en el menú principal?¡No hace nada!"*

La decisión de reorganizar todos los elementos del perfil de combate resultó en un buen diseño:

> *"Esto me gusta." (haciendo referencia al perfil de combate)*

7. Conclusiones

Este proyecto se inició con un objetivo claro, el de crear una aplicación que permitiera agilizar las partidas de rol, proporcionando, tanto al director como al jugador, herramientas para gestionar el estado y los cálculos de las mismas. Para ello se marcaron tres objetivos específicos: Permitir la gestión de personajes, proporcionar un mapa de eventos para organizar la información y disponer de una funcionalidad para el cálculo del resultado del combate. Como se ha podido observar en la memoria, los tres objetivos se han cumplido satisfactoriamente. Sin embargo, aunque la premisa fuese la misma, los detalles de los objetivos fueron cambiando durante el desarrollo del proyecto. Se tomaron en cuenta una serie de mejoras proporcionadas por los usuarios, y aunque la mayoría fueron cambios menores en la disposición de los elementos, hubo algunas que no fueron tan triviales; la más notable fue el cambio de las listas por defecto por listas customizadas. Estas listas son las que se menciona en el punto cinco y, aunque resultaron complicadas de implementar, el esfuerzo mereció la pena e incrementó considerablemente la usabilidad. Otra mejora que se propuso durante el desarrollo fue dividir el combate en fases para poder evitar que se aplique la reducción de vida y que quede más claro para el usuario lo que está haciendo la aplicación.

A nivel personal, este trabajo fue una gran oportunidad para formarme académica y profesionalmente. Aprender Android desde cero y realizar un proyecto como el que es CHIMERA en tres meses, me proporcionó experiencia tanto en los procedimientos y aproximaciones propias del desarrollo en Android, como en los generales que se presentan en cualquier proyecto de desarrollo software. Para ello fue indispensable el conocimiento aprendido en las asignaturas de la carrera; en concreto, a parte de las asignaturas que nos proporcionaban la base para el desarrollo software como Introducción a la Programación y similares, x fueron las más relevantes:

- **Diseño de Software:** En esta asignatura se explicaba el funcionamiento de algunos de los patrones de diseño más relevantes. Aunque en el proyecto no se crearan directamente muchos patrones de diseño, esta asignatura fue de gran ayuda para comprender el funcionamiento de una gran cantidad de clases que los utilizan en su implementación.

- **Análisis y Especificación de Requisitos:** Determinar y describir los requisitos correctamente es un aspecto indispensable en el desarrollo de software y gracias a esta asignatura se pudo completar el punto tres satisfactoriamente.

- **Mantenimiento y Evolución del Software:** Esta asignatura resultó de gran utilidad para crear código mantenible, sobre todo la realización de pruebas en JUnit.

- **Proceso de Software:** Aprender el desarrollo ágil del software fue de gran ayuda a la hora de gestionar el proyecto.

Para finalizar este apartado solo queda mencionar que la aplicación se puede encontrar en la tienda de Android[35]. A continuación, se mostrarán de nuevo los diagramas de radar de las aplicaciones del punto dos, en conjunto con el diagrama de radar de CHIMERA después de acabar el proyecto:

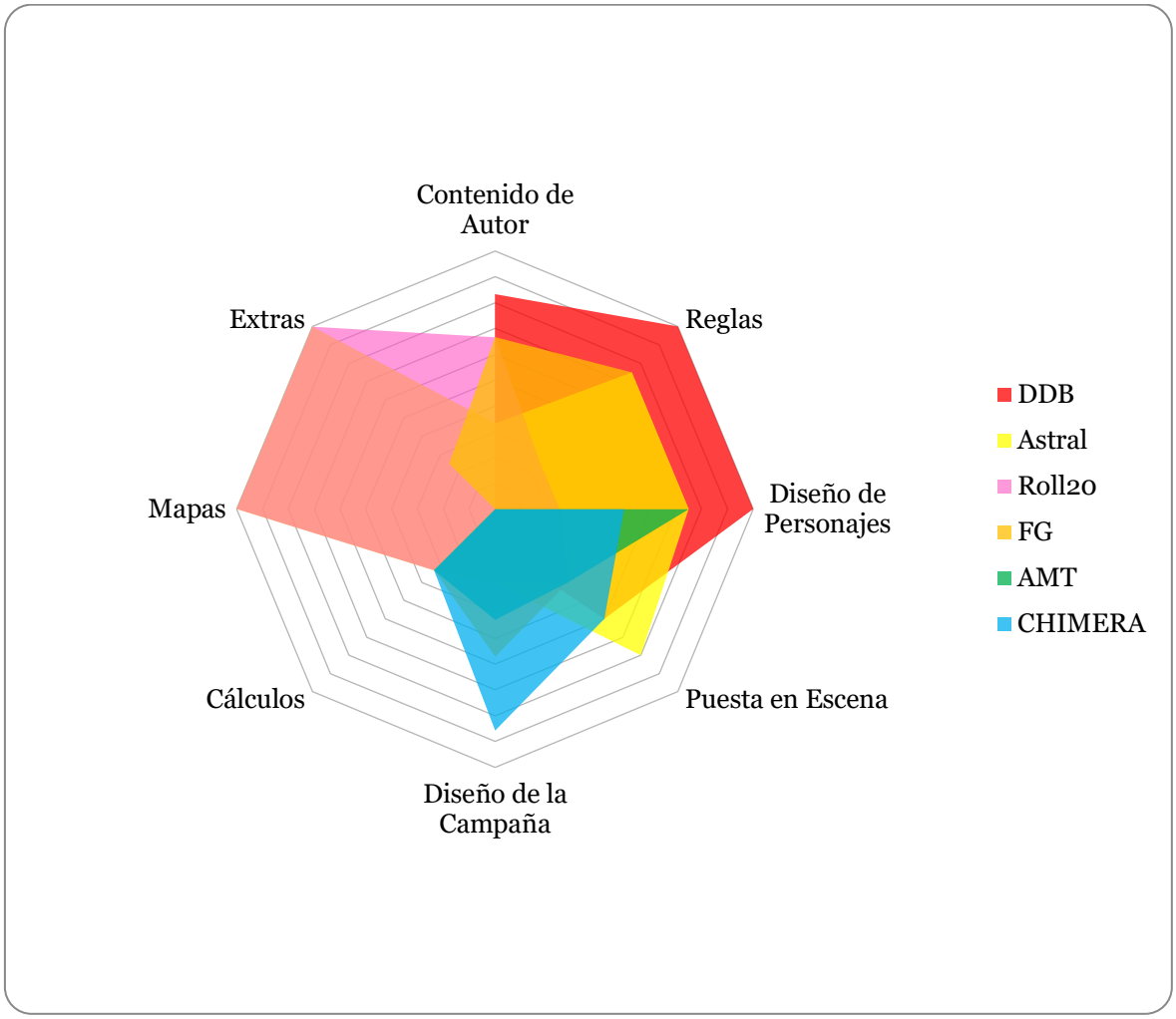

Figura 76 Comparativa final – Elaboración propia

8. Trabajos futuros

Como era de esperar, durante el desarrollo aparecieron muchas mejoras y funcionalidades interesantes que no se pudieron implementar en esta primera versión del producto. Por ello, después de un período de enfriamiento, se empezará a desarrollar una nueva versión. Durante este período, se obtendrá retroalimentación de los usuarios y se integrará con las funcionalidades que aparecieron durante este primer desarrollo que no pudieron implementarse. Algunas de las propuestas más prometedoras son las siguientes:

- Sustituir la descripción del jugador por un diario que se pueda separar por días.
- Añadir la posibilidad de tener submapas en el mapa de la campaña.
- Mostrar la barra de vida de los personajes en la pantalla de combate.
- Permitir cambiar los colores de los consumibles.
- Conectar al director con los jugadores para mantener actualizado el estado de los personajes en todos los dispositivos.
- Disponer de diferentes formas de ordenación para las listas.
- Poder separar personajes por tipos.
- Consumibles multivalor con varias barras.
- Añadir habilidades.
- Exportar campañas
- Exportar personajes
- Modificar los resultados de combate
- Reproducir música
- Vincular música a campañas, personajes, eventos y combates
- Masas de enemigos
- Umbrales en los consumibles.
- Creación de poderes/técnicas
- Poderes con gasto automático de consumibles.
- Lanzador de dados

- Manuales
- Imágenes por defecto de personajes
- Consumibles en el icono del personaje

9. Referencias

1. Historia de los juegos de rol [en línea] [11/4/2019]: https://rolosofo.com/2018/06/historia-de-los-juegos-de-rol-1973-2017/
2. WIZARDS OF THE COAST *Dungeons & Dragons Player's Handbook* Edición: 1 Wizards of the Coast, 29 de agosto de 2014 ISBN-13: 978-0786965601
3. Estado actual de D&D [en línea] [13/4/2019]: https://unpossiblejourneys.com/how-well-is-5th-edition-dungeons-and-dragons-selling/
4. Historia de los teléfonos móviles [en línea] [11/4/2019]: https://blogthinkbig.com/evolucion-del-movil-del-3310-al-iphone-x
5. JOHN HORTON *Android Programming for Beginners*. Packt Publishing, 31 de diciembre de 2015 ISBN-13: 978-1730928963.
6. Beneficios de los juegos de rol en la educación [en línea] [12/4/2019]: https://hipertextual.com/2015/07/juegos-de-rol-beneficios
7. Objetivos de Desarrollo Sostenible [en línea] [29/06/2019] https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/
8. JEFFREY RIES *Scrum: The Complete Step-By-Step Guide to Managing Product Development Using Agile Framework*. Publicación independiente, 30 de octubre de 2018 ISBN-13: 978-1730928963.
9. CARLOS B. GARCÍA APARÍCIO *Anima: Beyond Fantasy Core Exxet* Edición: 2 EDGE ENTERTAINMENT, 29 de octubre de 2010 ISBN: 978-84-96802-63-6
10. Aplicaciones para juegos de rol de sobremesa[en línea] [20/6/2019]: https://www.xatakandroid.com/aplicaciones-android/las-35-mejores-aplicaciones-para-masters-y-jugadores-de-rol
11. D&D Beyond [en línea] [5/4/2019]: https://www.dndbeyond.com/
12. Roll 20 [en línea] [5/4/2019]: https://roll20.net/
13. Fantasy Grounds [en línea] [8/4/2019] https://www.fantasygrounds.com/
14. Astral [en línea] [7/4/2019] https://www.astraltabletop.com/
15. Anima Master Toolkit [en línea] [13/4/2019] https://play.google.com/store/apps/details?id=com.nugganet.arget.mastertoolkit&hl=en_US
16. SHANE LACY HENSLEY *Savage Worlds* Studio 2 Publishing, Inc.,1 de marzo de 2005 ISBN-13: 978-0976360100
17. ROD STEPHENS *Beginning Software Engineering* Edición: 1, Sybex, 23 de marzo de 2015 ISBN-13: 978-8126555376
18. Coding in Flow [en línea] [24/06/2019] https://www.youtube.com/channel/UC_Fh8kvtkVPkeihBs42jGcA
19. ZoomableLayout [en línea] [24/06/2019] https://gist.github.com/anorth/9845602
20. Squire [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.herd.ddnext&hl=es_419

21. Fith Edition Character Sheet [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.wgkammerer.testgui.basiccharactersheet.app
22. Fight Club 5th Edition [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.lionsden.fightclub5
23. FaNG [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.crystalpeak.fantasynamegenerator
24. Dungeon Map Generator [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=externius.rdmg
25. Monster Factory [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=ca.splantio.splant.monsterfactory
26. Bag +5 [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.tartlegames.appofholding
27. Character Story Planner [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.lfantasia.android.outworld
28. Anima Magic Cards [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=fr.enlight.anima.animamagiccards
29. Cthulhu Sounds [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.antonioalarconmorales.cthulhu
30. Fantasy Insult Generator [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.ixiestudios.fantasyinsults
31. Fantasy Soundboard [en línea] [29/6/2019] https://play.google.com/store/apps/details?id=com.highfantasysoundboard
32. RPG Simple Dice [en línea] [1/7/2019] https://play.google.com/store/apps/details?id=com.ccp.rpgsimpledice&hl=es
33. The Legend of the Five Rings Dice [en línea] [1/7/2019] https://play.google.com/store/apps/details?id=com.FantasyFlightGames.DiceEngine
34. DnDice [en línea] [1/7/2019] https://play.google.com/store/apps/details?id=com.christian.bar.dndice&hl=es
35. CHIMERA [en línea] [1/7/2019] https://play.google.com/store/apps/details?id=com.aganchiran.chimera&gl=ES

Agradecimientos

Después de un breve pero intenso periodo de tres meses, escribo este apartado de agradecimientos terminando así mi trabajo de fin de grado. Han sido muchas las personas que me han hecho llegar hasta donde estoy hoy, cada una aportando su grano de arena, sin embargo, quiero agradecer especialmente a mi familia por darme apoyo todos estos años y por darme la oportunidad de estudiar en una de las mejores universidades de España. También quiero agradecer especialmente a Miguel, Jaume, Joaquim, Carles, Sergio, Raúl e Isabel por haber sido los *testers* durante el desarrollo del proyecto. Sin ellos no hubiera sido posible crear CHIMERA. Por último, pero no por ello menos importante, quiero dar las gracias a mi tutor, por ayudarme tanto durante el desarrollo del proyecto, sin su ayuda no habría sido capaz de haber redactado esta memoria ni haber llegado tan lejos con CHIMERA.

Muchas gracias a todos.

Glosario

Backlog – En el ámbito de Scrum hace referencia a la columna en la que se introducen las tareas por primera vez. Esta columna forma una lista con las tareas que pueden ser implementadas en el proyecto.

Brainstorming – Tormenta de ideas. Hace referencia a una reunión o fase del desarrollo en la que se proponen ideas sin pararse a pensar si son viables o no. De esta manera pueden surgir propuestas que, aunque en un principio puedan parecer descabelladas, después de un refinamiento o de la aportación de otras personas pueden resultar vitales para el proyecto.

Campaña – Historia que puede ser jugada usando las reglas de uno o varios juegos de rol. Una campaña puede entenderse como el equivalente en juegos de rol a una temporada en una serie de televisión.

Gráficos Vectoriales – Es un formato de imagen que representa las imágenes mediante vectores en vez de mediante mapas de bits. Este formato permite escalar las imágenes sin ningún tipo de perdida en la calidad, sin embargo, no son eficientes si la imagen es compleja. Es el formato más extendido a la hora de diseñar iconos.

Grimorio – Libro de conjuros. Se tratan de libros en los que están explicadas las fórmulas mágicas para ser capaz de lanzar ciertos hechizos. Suelen ser un elemento común en las historias de los juegos de rol.

IDE – Son las siglas en inglés de Entorno de Desarrollo Integrado (*Integrated Development Environment*). Se trata de un programa que proporciona diversas facilidades a la hora de desarrollar una aplicación (p.ej. autocompletado, esquema del proyecto o importación de paquetes automática).

Iniciativa – En el ámbito de los juegos de rol se trata de un valor numérico que representa los reflejos de un personaje. Este valor se utiliza para determinar el orden de actuación de los personajes en un combate, cuanto mayor sea la iniciativa, antes podrá actuar.

Kanban – Metodología para el desarrollo software que consiste en introducir las tareas a realizar en un tablero. El tablero estará dividido en una serie de columnas que indican el estado de las tareas (p.ej. Especificar, Programar, Probar, etc.). Las tareas deberán moverse de una columna a otra una vez se finalice la actividad en cuestión.

Material Design – Se trata de un lenguaje visual que sintetiza los principios clásicos de un buen diseño con la innovación de las tecnologías. Está muy extendido en el desarrollo de aplicaciones móviles y se caracteriza por ser un diseño minimalista que favorece la usabilidad.

Nigromancia – En los universos de fantasía, hace referencia a la magia relacionada con el alzamiento de los no-muertos. Sus practicantes utilizan hechizos y rituales para hacer que los cadáveres se muevan a voluntad.

NPC – Son las siglas en inglés de Personaje No Jugador (*Non-Player Character*). En los juegos de rol de sobremesa, hace referencia a todo aquel personaje que no es interpretado por un jugador, es decir que esté controlado por el director de juego.

Refactoring – Se trata de una técnica del ámbito de la ingeniería del software que consiste en cambiar la estructura interna del código sin cambiar el comportamiento externo, con el objetivo de hacer el software más mantenible y/o reusable.

Scrum – Metodología para el desarrollo software que consiste en crear incrementalmente el software en vez de planificar y ejecutar completamente el producto. Esto se consigue mediante la creación de mínimos productos viables durante el desarrollo para poder validar poco a poco las funcionalidades que se van implementando. Además, esta metodología quita importancia al procedimiento y se la da al equipo de desarrollo.

Sesión – Intervalo de tiempo de duración variable en el que todos los participantes se reúnen para jugar la campaña que haya preparado el director.

Sprint – Término utilizado en Scrum para definir el periodo durante el cual el equipo desarrolla un incremento del producto. La duración de los *sprints* es variable, siendo habitual que duren dos o tres semanas.

Tester – En el ámbito de la informática, hace referencia a la persona que se encarga de probar una aplicación exhaustivamente y notificar de los errores al equipo de desarrollo.

Anexo

Encuesta de usabilidad de CHIMERA

Responda a cada afirmación marcando con una X en el cuadro que corresponda de acuerdo con su experiencia al utilizar la aplicación, siendo:

- Completamente de acuerdo
- De acuerdo
- Neutral
- En desacuerdo
- Completamente en desacuerdo

					
1. Creo que me gustará utilizar con frecuencia esta aplicación					
2. Encontré la aplicación innecesariamente compleja					
3. Pensé que era fácil utilizar la aplicación					
4. Creo que necesitaría del apoyo de un experto para recorrer la aplicación					
5. Encontré las diversas posibilidades de la aplicación bastante bien integradas					
6. Pensé que había demasiada inconsistencia en la aplicación					
7. Creo que la mayoría de las personas aprenderían rápidamente a utilizar la aplicación					
8. Encontré la aplicación muy grande al recorrerla					
9. Me sentí muy confiad@ en el manejo de la aplicación					
10. Necesito aprender muchas cosas antes de manejarme en la aplicación					