# SCRUM RPG

Realizado por Isabel Arrans Vega y Matthew Bwye Lera

## Trabajo Fin de Grado

### Grado en Ingeniería Informática - Ingeniería del Software

Supervisado por Dr. Carlos Guillermo Müller Cejas
Dr. José Antonio Parejo Maestre


Universidad de Sevilla

Tercera Convocatoria

Curso 2021/2022

Publicado en abril 2024 por
Isabel Arrans Vega y Matthew Bwye Lera

isaarrveg@alum.us.es
maybwyler@alum.us.es

Nosotros, Isabel Arrans Vega y Matthew Bwye Lera con DNI 29584706J y 20229732J respectivamente,

**DECLARAMOS**

nuestra autoría del trabajo presentado en la disertación de este trabajo de final de carrera:

*Scrum RPG*

firmado,

Firma: Srs. Isabel Arrans Vega y Matthew Bwye Lera
en Universidad de Sevilla
a 11/04/2024

*A todos aquellos que han contribuido a nuestras andanzas en la US*

# Agradecimientos

Queremos mencionar en esta sección a nuestros amigos y compañeros, por inspirarnos con todas las vivencias que hemos tenido en la ETSII y sus anécdotas; a nuestros familiares, especialmente nuestros padres, por tratar de entender qué es Scrum cada vez que les explicamos el tema de nuestro proyecto y el uno al otro, ya que no hubiera sido lo mismo con solo uno de los dos en este "mini Scrum".

También expresar nuestro más sincero agradecimiento a nuestros tutores, Carlos y José Antonio, por habernos impulsado a hacer de nuestra idea original un trabajo de fin de grado conjunto y por las reuniones en las que nos han motivado y dado ideas. También al creador de ORKFramework, GamingIsLove, por las dudas que nos ha solucionado.

Sin todas estas personas, este Trabajo de Fin de Grado no existiría en la manera que lo hace ahora.

# Abstract

Scrum es una serie de buenas prácticas de trabajo cada vez más utilizadas en la ingeniería del software, y fácilmente extrapolables a otros ámbitos de trabajo. Se basa en gestionar el trabajo en equipo a través de una serie de ceremonias que tienen propósitos distintos.

Nosotros perseguimos exponer una nueva forma de enseñar estas técnicas y conceptos de una forma interactiva y divertida, pudiendo hacer que el usuario ya tenga experiencia ficticia en aplicar estas técnicas, adquiriendo una capa más de aprendizaje que la que obtendría leyendo acerca de Scrum.

Con este objetivo, creamos un videojuego que educase en los principios de Scrum. El sistema de juego está basado en los RPG por turnos, donde el jugador toma control de un equipo de Scrum, y toma decisiones globales (es decir, el jugador no es un personaje particular, sino que controla a todos). Si el jugador aplica Scrum correctamente, debería poder superar los obstáculos que se le presenten con facilidad, y entenderá la importancia de sus acciones, y su paralelismo con la realidad.

Scrum se divide en 4 ceremonias: Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective. Cada una de estas ceremonias tiene su propio propósito, y completarlas todas da por terminado el Sprint actual. En nuestro proyecto se recorren todas, exponiendo las principales funciones de cada una.

Durante el Sprint Planning realizamos un esbozo de lo que se hará en el Sprint, en las Daily Scrums se planean avances diarios en el Sprint, durante la Sprint Review se revisa el resultado del Sprint y la Sprint Retrospective nos ayuda a tomar decisiones para el futuro Sprint en base a un análisis del Sprint previo.

Para la realización de este proyecto, hicimos uso de Unity, un motor de desarrollo de videojuegos, y usamos un framework para facilitar la implementación del tipo de juego que diseñamos. El framework nos daba una base de datos con tablas creadas para los distintos tipos de elementos que necesitaríamos, y tenía funcionalidades básicas relativas al *combate* de los juegos RPG implementadas.

# Índice general

# Índice de figuras

# Índice de tablas

# Índice de Códigos

# Parte I

# Introducción

# Contexto

*En este capítulo se realizará una exposición del contexto y motivación del proyecto.*

## 2.1 SCRUM

Scrum es un conjunto de prácticas que se usa en proyectos software que hace uso de las Metodologías Ágiles para facilitar la resolución de problemas complejos a través de soluciones adaptativas, haciendo especial énfasis en prácticas relacionadas con el trabajo en equipo y cómo mejorar la eficacia del trabajo.

Para completar un proyecto lo dividimos en Sprints. Un Sprint es un periodo acotado en el que se realizan tareas relativas al proyecto. El trabajo total está recogido en el *Product Backlog (PB)*.

Cada Sprint consiste de 4 reuniones:

1. Sprint Planning: Marca el comienzo de un Sprint. Se seleccionan las tareas a realizar durante el Sprint, duración del Sprint y plan de ejecución inicial de estas tareas. La duración de un Sprint debe estar comprendido entre 1 y 4 semanas, lo más frecuente son 2 semanas.

2. Daily Scrum: Al principio de cada día, el equipo debe reunirse para especificar qué hicieron el día anterior, qué problemas encontraron y qué tarea harán ese mismo día. Esta parte es esencial, ya que es donde más se ve la flexibilidad que Scrum ofrece. En función de los avances diarios de cada miembro, el equipo debe decidir qué rumbo coger, ayudar a miembros que hayan tenido problemas, modificar el ritmo, etc. Estas reuniones no deben durar más de 15 minutos, y se deben hacer de pie para incentivar el que dure lo menos posible.

3. Sprint Review: Esta reunión se hace al final del sprint. Se muestran los resultados al *Product Owner (PO)*, y se evalúan los requisitos del PB, actualizándolo acordemente.

4. Sprint Retrospective: La última reunión de un Sprint. Los miembros del grupo deben reunirse para discutir acerca de qué aspectos del Sprint salieron bien, cuáles fueron mejorables y cuáles salieron mal. Tras hacer un listado de cada uno de estos aspectos, el grupo debe determinar qué hacer en el próximo Sprint para reducir los aspectos a mejorar y malos e incrementar y mantener los que salieron bien.

La clave de Scrum reside en la existencia del Scrum Master, cuya principal responsabilidad es asegurarse de que el equipo haga correcto uso de la metodología Scrum. El entorno de trabajo debe seguir estos 4 pasos:

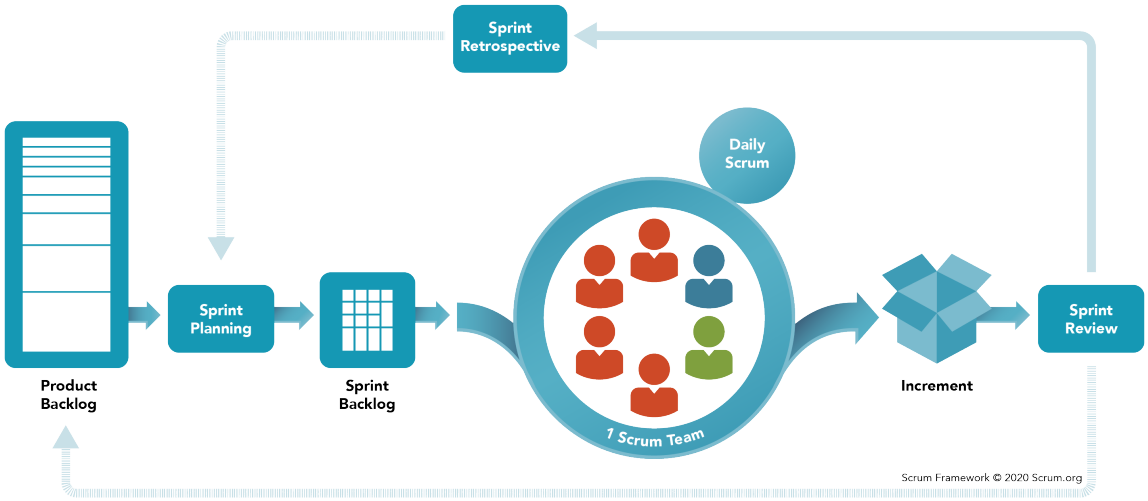



Figura 2.1: Estructura del Sprint

1. El PO exigirá la resolución de problemas complejos que estarán reflejados en el PB.
2. El equipo de Scrum añade una selección de las exigencias del PO al *Sprint Backlog (SB)*, convirtiendo ese trabajo en incremento de valor durante el Sprint.
3. El equipo de Scrum y los interesados analizan los resultados y los reflejan en el próximo Sprint.
4. Repetir desde el primer paso.

## 2.2 MOTIVACIÓN PARA SCRUM RPG

A pesar de parecer sencillo, Scrum es difícil de amaestrar[20]. Hay varias dificultades que distintos equipos encuentran a la hora de aplicar scrum[16]:

1. "Entregas de baja calidad"
2. "Problemas con integración de módulos"
3. "Código de baja calidad"
4. **"Interrupciones del equipo de trabajo"**

5. **“Equipos nuevos a Scrum tienden a tener más problemas”**
6. **“Duración de los Sprints”**
7. **“Falta de formación en Scrum”**
8. “Proceso de despliegue”
9. “Gestión de Product Backlog y Sprint Backlog”
10. “Falta de prácticas técnicamente superiores”
11. “Equipos múltiples (Scrum de Scrums)”
12. “Falta de entendimiento y profundidad de las métricas”
13. “Falta de gestión de riesgos”
14. “Falta de documentación”
15. “Demasiado idealista”
16. “Demasiada comunicación del equipo”

En el mismo artículo[16] que lista estas dificultades se listan soluciones propuestas para cada una. Algunas de estas propuestas mencionan una mejor formación, madurez del equipo en la aplicación de Scrum o más conocimiento práctico, concretamente 4, 5, 6 y 7.

Scrum RPG es nuestra respuesta a algunos de estos problemas. La motivación de nuestro producto es crear un entorno virtual que emula un equipo Scrum que tiene que resolver tareas para un proyecto de una determinada duración. El usuario controla a los personajes del equipo, y haciendo uso de las mecánicas ofrecidas debe completar las tareas siguiendo las prácticas de Scrum.

# Objetivos

*En este capítulo se detallarán los objetivos que se pretenden conseguir con este proyecto.*

## 3.1 Creación de una emulación virtual de Scrum

Nuestro trabajo tiene como fin analizar, diseñar e implementar una emulación virtual en formato de juego que imite cualidades propias de trabajos vinculados a Scrum cuyo propósito es introducir los conceptos básicos de Scrum de forma práctica e interactiva a equipos y particulares que no estén del todo familiarizados con ellos. Las limitaciones que tenemos nos impiden cubrir todos y cada uno de los casos que nos gustaría cubrir, por lo que no esperamos una panacea ideal. Tampoco esperamos pruebas automáticas por el tipo de proyecto que es.

Los primeros pasos serán analizar qué tipo de entorno podría emular mejor el tipo de contexto que queremos representar y eligiremos qué tecnologías favorecen el tipo de proyecto que abarcaremos. Una vez todo esté elegido y estemos familiarizados con las teconologías a usar, pasaremos a diseñar la arquitectura del software para finalmente implementar nuestro proyecto. Una vez el proyecto esté terminado, se enviará a distintas personas acompañado de un cuestionario de feedback, y semanalmente haremos actualizaciones siguiendo el feedback recibido, haciendo así una fase de mantenimiento hasta el momento de la entrega.

Nuestro producto debe contar con una serie de conceptos elementales, como son Scrum Master, Sprint y las ceremonias de Scrum (Sprint Planning/Review/Retrospective y Daily Scrum) que hagan que el usuario entienda el contexto en el que se le sitúa.

Las 4 dificultades que hemos decidido enfrentar son *Interrupciones del equipo de trabajo* [4], *Equipos nuevos a Scrum tienden a tener más problemas* [5], *Duración de los Sprints* [6] y *Falta de formación en Scrum* [7]. Nuestro producto debe ayudar a combatir cada una de estas dificultades.

Lo que esperamos es que con este producto el usuario reciba una buena introducción a Scrum, que le dé un punto de vista distinto al que le daría un manual o un vídeo ya que nuestro producto incluye un componente interactivo que daría un mínimo de experiencia práctica. Los equipos Scrum más maduros no cuentan con los problemas mencionados, y haciendo uso de nuestro producto sus equipos empezarán con un mayor nivel de madurez, ya que entenderán la utilidad del Scrum Master, de cada una de las ceremonias, entenderán donde reside la fortaleza de la flexibilidad que ofrece Scrum y entenderán la importancia de medir bien el tiempo de un Sprint. Cabe destacar que nuestro producto no busca formar a alguien para ser Scrum Master.

## 3.2 Objetivos personales

Además de conseguir hacer el proyecto propuesto, nosotros mismos buscamos también unas metas personales con este trabajo.

- **Aprender a usar Unity**: A pesar de que Matthew contaba con experiencia previa en Unity, Isabel apenas había realizado proyectos en este motor. Realizar este trabajo de fin de grado significa adquirir un nivel en el manejo de Unity, lo cual era algo que Isabel llevaba queriendo de hacía tiempo.

- **Completar un proyecto RPG**: Desde hace mucho tiempo, ambos hemos usado herramientas como RPG Maker para hacer juegos, pero hasta hace poco no logramos considerar terminado ninguno, y este es el primero que consideramos terminado que sea de género RPG.

- **Crear una herramienta educativa**: En segundo de carrera empezamos a cuestionarnos qué tipo de proyectos nos gustaría hacer en el futuro cuando estuviéramos trabajando. Ambos pensamos que proyectos de gamificación orientados a la educación o formación era un tipo de proyecto que funcionaría para ambos.

  Tenemos esperanzas en que este proyecto pueda inclinar nuestra carrera profesional en esta dirección, y que este sea el primero de muchos proyectos de este tipo en el que nos veamos envueltos.

- **Poner en práctica los conocimientos de la carrera**: Sin duda el TFG nos ha obligado a hacer uso de todos los conceptos relacionados con la Ingeniería del Software que hayamos adquirido durante nuestros estudios en el grado. Ha sido muy interesante, e incluso diría que nostálgico, el tener que recuperar conocimientos que nos fueron impartidos hasta hace 3 años y ponerlos en práctica.

  El hecho de que fuéramos dos personas, que encima han estado trabajando juntos desde el principio del grado, claramente a ayudado mucho para hacer memoria o recuperar apuntes o proyectos pasados para referencias.

- **Hacer nuestro último proyecto del grado juntos**: Esto es algo muy personal para ambos. Desde el principio, ambos estábamos en los mismos equipos y trabajábamos en los mismos proyectos, pero en 4º Matthew hizo una estancia Erasmus mientras que Isabel se quedó en la US. Esto quería decir que posiblemente el último proyecto que hiciéramos juntos era DP2, ya que en 4º cada uno estaría en una universidad distinta.

Sin embargo, planeamos hacer el TFG juntos, haciendo que nuestro último proyecto juntos fuera el proyecto con el que nos despediríamos del grado.

# Análisis de mercado

*En este capítulo se explicará la comparativa de nuestro proyecto con alternativas ya existentes.*

## 4.1 Opciones ya existentes

Mirando en fuentes fiables[32], podemos ver que las formas más recomendadas de aprender Scrum son las siguientes:

- Leer material relacionado con Scrum (guías, foros, artículos...).
- Realizar cursos de formación.
- Realizar prácticas de evaluación para practicar Scrum.
- Aplicar Scrum en espacios de trabajo.

En resumen, se recomienda estudiar Scrum o practicarlo.

Aunque las fuentes mencionadas no hagan referencia a aplicaciones didácticas, debido a la semejanza con nuestro proyecto, los revisaremos también.

Los resultados más destacables son[19]:

- **Scrum Pocket Training**[12]: Scrum Pocket Training es una aplicación para dispositivos móviles que enfrenta al usuario con unos cuestionarios que miden su conocimiento de Scrum.
- **Lego4Scrum**[21]: En este juego varios equipos deben aplicar Scrum para crear un proyecto usando piezas de Lego, y el objetivo es que todos los equipos juntos sean capaces de crear una ciudad de Lego. De esta forma, los distintos participantes aplican Scrum a una situación práctica, donde además deben coordinarse con otros equipos de Scrum.
- **The Scrum Game**[29]: The Scrum Game es un juego de mesa para 2-6 personas. Los distintos jugadores forman parte de un único equipo Scrum. El juego simula un Sprint, y el objetivo es que los jugadores cooperen para completar sus objetivos en un tiempo dado. El juego hace uso de mecánicas que incentiva a los jugadores a tomar decisiones rápidas para solucionar problemas comunes.
- **Scrum Knowsy**[11]: El objetivo de Scrum Knowsy es unificar la idea de Scrum de los distintos miembros del equipo. El objetivo del juego es adivinar lo que los demás miembros del equipo piensan acerca de diversos aspectos de Scrum.

- **The Product Owner Game**: Existe relativamente poco material acerca de cómo mejorar como Product Owner. Este juego intenta atacar a ese punto flaco de las guías y métodos de aprendizaje de Scrum, tratando de dar una formación específica para los Product Owners. Hasta donde sabemos, este juego no está disponible, por lo que no haremos un análisis del mismo.

Analicemos estas opciones. Los aspectos que estudiaremos serán su coste, su objetivo e impacto, tiempo requerido y obtención de certificados.

### 4.1.1 Coste

El material de lectura puede encontrarse gratuitamente, aunque habrá material de pago, pero en general los cursos de formación y las prácticas de evaluación son de pago, siendo aquellos que otorguen certificación los más caros.

Scrum Pocket Training cuesta entre 2 y 3 euros, Lego4Scrum cuesta el precio de la guía y el precio de las piezas de Lego (que puede tener un coste muy alto, dependiendo del número de participantes), The Scrum Game es imprimible (cuesta lo que cueste imprimirlo), Scrum Knowsy cuesta 1 euro.

### 4.1.2 Objetivo e impacto

El objetivo de las distintas guías depende en gran medida de la guía que estemos estudiando en cuestión, pero su impacto estará siempre limitado por la falta de aplicación práctica.

Los cursos de formación tienen como objetivo general preparar completamente a los estudiantes, el éxito del curso depende de la calidad del personal docente y sus técnicas.

El éxito de la experiencia práctica depende en gran medida del conocimiento teórico previo, y su objetivo es mejorar la aplicación de las técnicas de Scrum.

Los distintos juegos de Scrum tienen objetivos distintos.

Scrum Pocket Training tiene como objetivo evaluar y mejorar el conocimiento teórico del usuario, pero el impacto es mínimo tan limitado como el de las guías, aunque tener un componente lúdico ayuda a mantener la concentración durante el aprendizaje.

El objetivo de Lego4Scrum es que los jugadores entiendan con experiencia práctica qué es Scrum, por qué es útil y cómo aplicarlo. Sin un conocimiento teórico que acompañe al juego, es posible que los jugadores no aprendar todos los conceptos de Scrum, pero como mínimo entenderán su utilidad y cómo se aplica a grandes rasgos.

De forma muy similar a Lego4Scrum, The Scrum Game puede ayudar a comprender los aspectos positivos de Scrum, pero sin conocimiento teórico de base, no ayudará mucho.

Scrum Knowsy es distinto a los demás ya que su objetivo no es educar Scrum sino a mejorar la eficacia de su implementación en un equipo determinado. El impacto de esta herramienta depende en gran medida de la actitud del equipo, ya que espera flexibilizar los cimientos de los conceptos de Scrum del equipo.

### 4.1.3 Tiempo requerido

Las guías requieren que el usuario invierta tiempo estudiándolas. El objetivo que tenga el usuario para estas guías variará el tiempo invertido en su estudio.

Los cursos de formación de Scrum requieren un mayor grado de implicación, y tienen una duración de al menos 14 horas[4].

El tiempo que el usuario quiera invertir en experiencia práctica varía en función de cómo aplicarla, pero podemos asumir que como mínimo un par de horas han de aplicarse.

Los juegos de Scrum intentan ofrecer una experiencia mucho más rápida que los cursos o guías. Aún así, en general los juegos necesitan un poco de trasfondo teórico. Scrum Pocket Training intenta enseñar algunos conceptos teóricos con cada respuesta del jugador, por lo que igual el conocimiento teórico necesario para este juego es menor.

### 4.1.4 Obtención de certificados

Sólo a través de cursos certificados podremos obtener certificados. Aún así, aprender Scrum de otras fuentes puede agilizar el proceso de obtención de certificados. Generalmente, el conocimiento teórico que ofrecen los juegos es menor que el que ofrecen las guías, que es lo que normalmente es evaluado para la obtención de certificados.

## 4.2 Análisis de nuestro producto

Nuestra alternativa intenta ofrecer una buena oportunidad para una introducción a Scrum. Si lo analizamos punto a punto:

- **Coste:** En nuestro TFG no hemos estudiado opciones de comercialización del producto, pero haciendo un análisis del mercado actual, podemos asumir que el juego podría estar disponible por un precio menor a 5 euros.
- **Objetivo e impacto**: Nuestro producto intenta ser una introducción a Scrum. No asumimos que el usuario tenga ningún conocimiento teórico previo relativo a Scrum, aunque sí un poco relativo a la Ingeniería del Software y metodologías Ágiles. Ofrecemos un glosario en el que se explican conceptos clave de Scrum, justamente para ofrecer esa base teórica que el usuario va a tener que adquirir para poder aplicar Scrum apropiadamente.

  Debido a que presentamos al usuario casos prácticos en las 4 ceremonias fundamentales de Scrum, consideramos que el usuario habrá entendido el propósito fundamental de cada una de ellas al terminar de usar nuestro producto.

  El sistema de varios Sprints se implementó con el propósito de que el jugador presente una curva de aprendizaje, donde cada Sprint lo hará mejor que el anterior, aprendiendo de sus errores y aplicando una mejor versión de Scrum.
- **Tiempo invertido**: El juego es algo largo de completar (alrededor de 15-30 minutos), pero no hay necesidad de completarlo para adquirir todo lo que el juego pretende ofrecer. Justamente esto se hizo para que cada jugador pueda marcar su propio ritmo de aprendizaje, y el juego no resulte ni demasiado largo para unos ni demasiado corto para otros.
- **Obtención de certificados**: Nuestro producto se aleja de este objetivo. Pretendemos introducir Scrum, no hacer que el usuario se vuelva experto en su uso, por lo que utilizar nuestro producto no preparará al usuario para obtener un certificado de Scrum.

## 4.3 Conclusiones

Habiendo visto las opciones favoritas y comparándolas con nuestro producto podemos deducir que nuestro producto presenta los siguientes componentes únicos:

1. Nuestro producto ofrece una buena introducción de Scrum para absolutos principiantes. Ninguna de las opciones lúdicas que hemos expuesto ofrece esta alternativa, por lo que si alguien intentase aprender Scrum sin conocimiento previo, necesitaría empezar estudiando.

2. Hereda componentes propios de videojuegos, al contrario que las opciones expuestas, que hacen uso de componentes propios de juegos de mesa, juguetes analógicos o cuestionarios.

3. Ofrece un ritmo de aprendizaje personalizable, de tal forma que el usuario puede adaptar su experiencia a su preferencia.

4. Al igual que el material de lectura, la experiencia es individual. Esto podría verse como una contra ya que Scrum hace uso de trabajo en equipo para funcionar, pero justamente esta cualidad agiliza el proceso de aprendizaje, y permite personalizar la experiencia sin tener que hacer que otros se adapten.

En resumen, nuestro producto es más comparable a las guías y otros materiales de lectura que a cursos de formación o las demás opciones gamificadas. Al contrario que el material de estudi, nuestra propuesta ofrece una alternativa más interactiva que tiene componentes de experiencia práctica. Consume poco tiempo y permite construir unos cimientos como ninguna de las demás opciones ofrece.

# Parte II

# Organización

# Metodología

*En este capítulo, se explicará la metodología aplicada durante el transcurso del proyecto y las herramientas empleadas para poder llevarla a cabo.*

## 5.1 Metodología del proyecto

La metodología por emplear en el proyecto se ve obligada a ser diferente respecto a lo que se puede esperar de un trabajo de fin de grado corriente, ya que se trata de un caso particular en el que lo realizaremos dos alumnos, Isabel Arrans y Matthew Bwye, bajo la tutela de dos profesores, Carlos Müller y José Antonio Parejo. De esta manera, la comunicación se dará en dos situaciones diferentes: tanto de los alumnos a tutores como de alumno a alumna. Ocasionalmente, se contactará con el creador del framework rpg usado en Unity, ORKFramework.

A continuación, detallaremos la metodología a seguir durante el transcurso de este proyecto.

### 5.1.1 La metodología

Al tratarse este proyecto de un trabajo de fin de grado con más alumnos, se da una buen ocasión para poder trabajar con metodologías ágiles. En este caso, además de coincidir con la idea del propio videojuego a desarrollar, aplicaremos la metodología Scrum[24] con algunas variantes para adaptarlo a las condiciones de nuestro proyecto. En esta variante, los alumnos seremos el equipo de desarrollo y Scrum Master y los tutores cumplirán el rol de Product Owner.

Para el Sprint Planning, dados los bloques de trabajo, se reunirán los dos alumnos para identificar las tareas del sprint, estimarlas, asignarlas y establecer una duración del sprint, generalmente de un mes. Esta serie de decisiones serán comunicadas a ambos tutores para mantener constancia de los distintos hitos del proyecto, además de una petición de fecha de reunión para, en el final del sprint, realizar la Review de este.

Durante el desarrollo, los dos alumnos tendrán reuniones para informarse el uno al otro de los avances que se van haciendo en las tareas del sprint, pudiéndose consultar pequeñas dudas o plantear ayudas en tareas más complejas. El plan inicial será llevar a cabo dos reuniones de este tipo a la semana, para poder conciliar los diferentes horarios de los dos alumnos. La frecuencia de las reuniones podrá aumentar hacia el final de los sprint y siempre se trabajará con ellas de manera flexible.

Al finalizar el Sprint, en la fecha escogida por alumnos y tutores, se realizará la Sprint Review, en la que los tutores, al actuar también de Product Owners, valorarán el trabajo realizado. Durante esta reunión, se propondrán mejoras para el trabajo mostrado, que se aplicarán en el siguiente sprint, junto a los bloques de tareas ya identificados

para este. Finalmente, los alumnos se reunirán y realizarán la retrospectiva del Sprint, y darán paso al siguiente sprint.

Respecto al rol Scrum Master, realizaremos una rotación del mismo entre los alumnos. De esta manera, podremos aportar mejor los diferentes puntos de vista y procurar la mejor aplicación de Scrum durante el desarrollo. Para hacerlo de la mejor manera posible, rotaremos el rol de Scrum Master cada Sprint, procurando que, el alumno que deba cumplir con el rol durante el sprint tenga una cantidad de puntos de historia por cumplir algo menor que el otro. Así, trataremos de igualar la carga de trabajo de ambos.

Dividiremos el desarrollo del proyecto en diferentes bloques, ordenados debido a sus dependencias y complejidades. De esta manera, los diferentes sprints a realizar serán iteraciones incrementales de los diferentes bloques previstos. En el primero de ellos, analizaremos los requisitos respecto a las tecnologías elegidas para, en el segundo bloque, hacer un buen diseño del proyecto, previo a la implementación. El tercer bloque se dedicará a la familiarización con el framework y el motor gráfico que utilizaremos para el correcto funcionamiento del juego, además de configuraciones básicas de este. El cuarto bloque, también el más extenso, consistirá en el desarrollo de una primera versión completa de las mecánicas principales del videojuego, es decir, las fases de Sprint Planning y Combate (Desarrollo), ya que contiene la principal jugabilidad y conforman la base sobre la que se debe construir el resto del proyecto. El quinto bloque consistirá en conectar y pulir las mecánicas desarrolladas en el anterior bloque y en añadir una primera versión las fases de Sprint Review y Sprint Retrospective. A continuación, en el sexto bloque consistirá en unificar y ajustar la jugabilidad y finalizar el desarrollo del prototipo. Por último, en el séptimo bloque, gestionamos el mantenimiento de nuestro software dando a diferentes usuarios a probar el juego y analizando su feedback mediante un formulario, para luego aplicar los cambios que veamos oportunos en tramos semanales.

### 5.1.2 Herramientas de comunicación y organización

Para la comunicación con los tutores empleamos principalmente Microsoft Teams, gracias a las licencias gratuitas que nos han sido concedidas por ser estudiantes de la Universidad de Sevilla. De esta manera, son posibles las reuniones no presenciales mediante videollamadas, pudiendo compartir la pantalla o enviar archivos para mostrar los avances de proyecto en las diferentes Sprint Review. También utilizamos el correo electrónico para convocar reuniones o consultar dudas puntuales a los tutores.

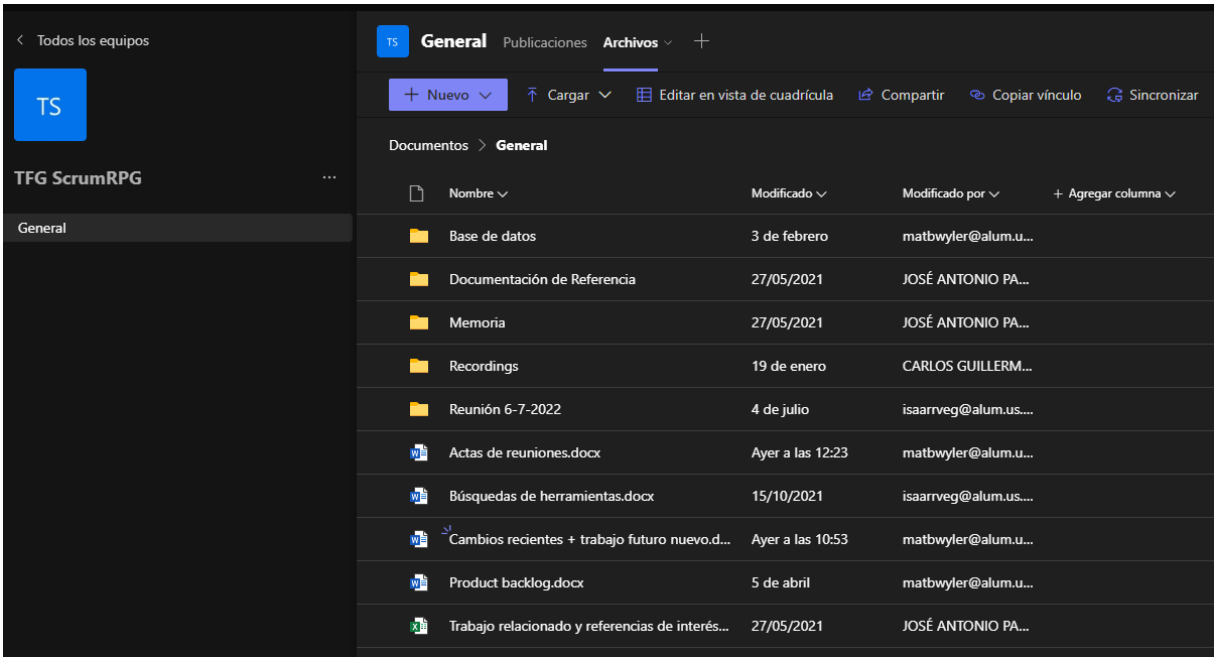



Figura 5.1: Microsoft Teams

Para la comunicación entre alumnos, se ha utilizado en gran medida Discord, ya que es una herramienta actual a la que estamos acostumbrados, debido a que la empleamos con mucha frecuencia y que resulta muy cómoda para dividir la conversación en diferentes canales, compartir rápidamente archivos y hacer videollamadas para las diferentes reuniones.

Respecto a la gestión de tareas, del Product Backlog y Sprint Backlog, se ha creado un proyecto de Github del que hemos utilizado su tablero Kanban, herramienta con la que estamos familiarizados, configurándolo para que sus columnas fueran adecuadas para el uso de Scrum. Inicialmente organizaríamos las tareas en este proyecto. Sin embargo, encontramos más rápido una manera más flexible de llevar este control, aplicable solo porque tan solo somos dos personas realizando este proyecto: en el mismo Sprint Backlog, recogido en un acta, escribiríamos la asignación de tareas. Debido a la constante comunicación entre alumnos y a que las tareas no estaban demasiado divididas, era suficiente para poder saber el estado de las tareas en todo momento.

Para la gestión de código fuente y otros archivos, inicialmente utilizamos el repositorio en el que se había creado el proyecto con el tablero Kanban, pero diversos problemas hicieron necesario que cambiáramos de herramienta en varias ocasiones durante el transcurso del proyecto.

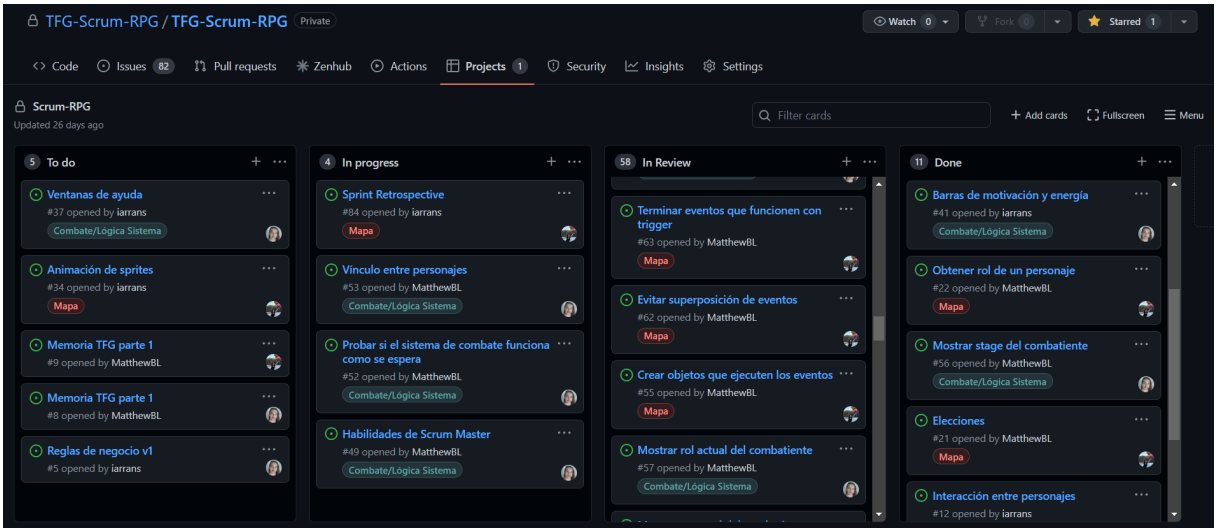



Figura 5.2: Tablero Kanban de Github

El primer problema surgió debido a que Github ofrece un límite de subida de archivos de 2 GB y algunos ficheros necesarios para el uso del framework dentro de Unity excedía más del doble de este límite. Por esto, añadimos la extensión Git Large File Storage (LFS). Sin embargo, descubrimos mediante su uso que LFS tenía un límite de 12 GB al mes, cantidad que excedíamos con tan solo 4 subidas al repositorio principal. Como en este proyecto somos dos alumnos desarrollando, no nos podíamos permitir hacer tan solo 4 subidas en todo un mes.

El siguiente paso fue buscar un repositorio propio de la empresa del motor gráfico, Unity Collab, que no presentaba límites de tamaño y, al estar pensado para Unity, es mucho más práctico y cómodo para el trabajo con este. De esta manera, ya podíamos hacer todos los cambios que quisiéramos. No obstante, al tiempo de emplear esta herramienta, Unity cambió sus términos y migró automáticamente todos los repositorios a PlasticSCM, por lo que hubo que cambiar algunos parámetros de configuración y usar una nueva UI para acceder al repositorio.

PlasticSCM es una herramienta de gestión distribuida similar a Github, pensada para proyectos de Unity que permite mediante su interfaz de usuario especial trabajar con archivos de gran tamaño y binarios. Permite las mismas funcionalidades que otros gestores que hemos aprendido a lo largo del grado, de forma que nos acostumbramos rápidamente a la herramienta y nos permitió organizar el repositorio mejor que con las anteriores herramientas.

Para la comunicación con el creador del Framework utilizado en Unity, ORKFrame-

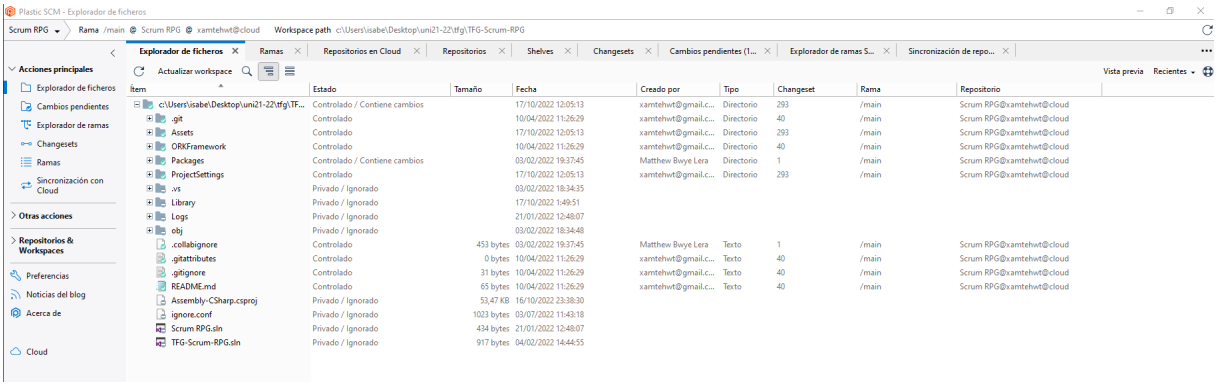



Figura 5.3: PlasticSCM

work, debido a que este es específico, poco conocido y la documentación es bastante pobre, hemos usado el foro específico de su comunidad.

Como herramienta de gestión de tiempo, hemos empleado Clockify, ya que lo hemos empleado para más proyectos y el uso que le íbamos a dar entraba dentro de su versión gratuita. Además de contador de tiempo, permite crear proyectos, tareas para poder asociar los tiempos y generar gráficas adecuadas.

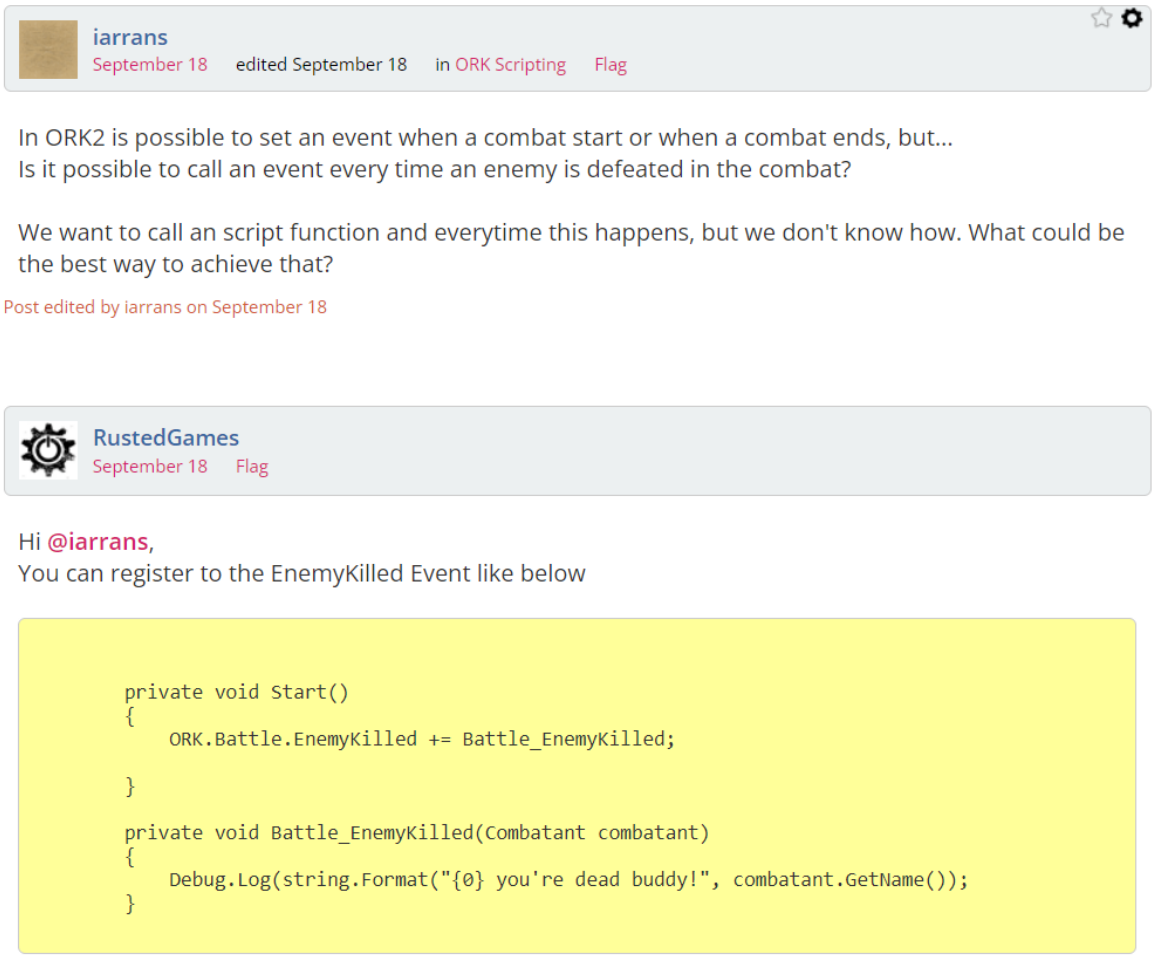
Por último, para comunicación con usuarios piloto, en los últimos sprints, utilizamos formularios de la herramienta Google Forms para poder analizar sus distintas opiniones del juego de manera más cómoda y eficaz y poder determinar cómo es la percepción general de los resultados del proyecto.

## 5.2 Documentación

Para la creación del modelo de dominio y de otros diagramas, hemos utilizado la herramienta de Diagrams.net, anteriormente conocida como Draw.io, ya que lo hemos empleado más veces a lo largo del grado y tiene recursos para todo tipo de diagramas UML y organigramas, además de estar en la nube y permitir la exportación de los diagramas generados tanto a pdf como a imagen.

Para la recolección inicial de requisitos, así como para las actas de reunión, se ha utilizado Microsoft Word, bajo la misma licencia de la Universidad de Sevilla que nos proporcionó Microsoft Teams. Esto nos ha permitido hacer anotaciones rápidas, cómodas y compartir rápidamente los documentos que íbamos generando.

Finalmente, para la escritura de la memoria, se ha empleado LaTex desde la plata-



iarrans
September 18 edited September 18 in ORK Scripting Flag

In ORK2 is possible to set an event when a combat start or when a combat ends, but...
Is it possible to call an event every time an enemy is defeated in the combat?

We want to call an script function and everytime this happens, but we don't know how. What could be the best way to achieve that?

Post edited by iarrans on September 18

RustedGames
September 18 Flag

Hi @iarrans,
You can register to the EnemyKilled Event like below

```
private void Start()
{
    ORK.Battle.EnemyKilled += Battle_EnemyKilled;

}

private void Battle_EnemyKilled(Combatant combatant)
{
    Debug.Log(string.Format("{0} you're dead buddy!", combatant.GetName());
}
```

Figura 5.4: ORK Forum

forma Overleaf. Esta herramienta nos permite agilizar el proceso de dar estilos al documento, inserción de imágenes y código. Hemos empleado la plantilla proporcionada por los tutores, por lo que hemos podido partir de un estilo ya definido. Otra ventaja que nos ofrece Overleaf es que, al ser un editor en la nube, nos facilita la compartición de la memoria tanto entre los alumnos como con los tutores.

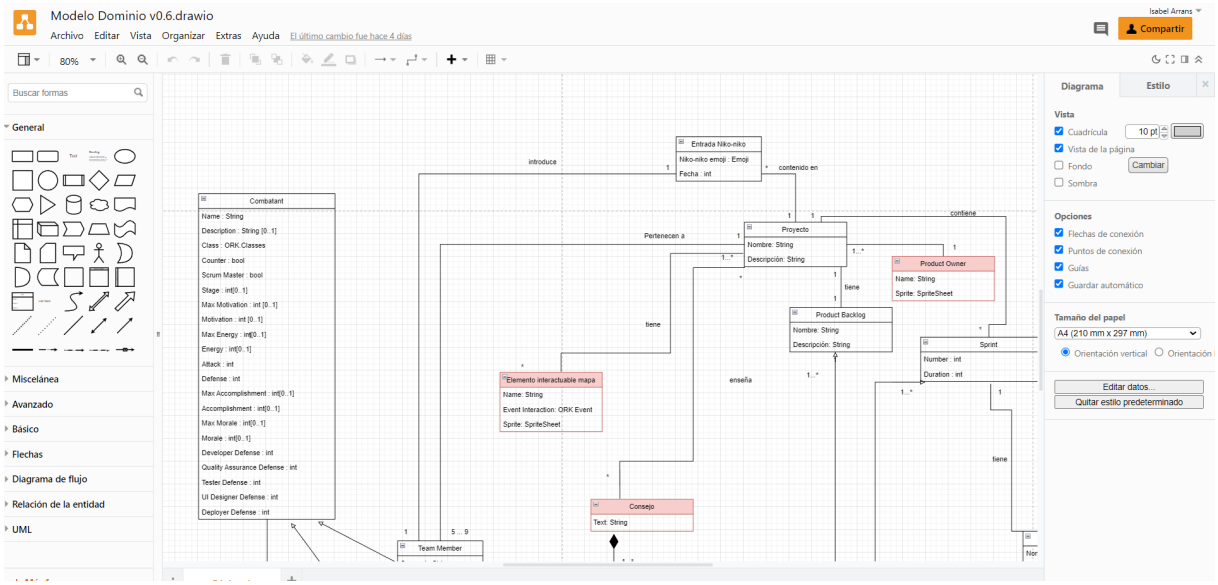



Figura 5.5: Diagrams.net

| Resumen de herramientas empleadas en el proyecto | | |
|---|---|---|
| **Herramienta** | **Versión** | **Link** |
| **Microsoft Teams** | 4.12 | Plataforma |
| **Discord** | Stable 104967 | Página de descarga |
| **Github** | | Plataforma |
| **Plastic SCM** | 11.0.16.7626 | Página de descarga |
| **ORK Forum** | | Página del foro |
| **Google Forms** | | Plataforma |
| **Overleaf** | | Plataforma |
| **Diagrams.net** | | Plataforma |
| **Clockify** | | Plataforma |

Tabla 5.1: Herramientas usadas en el proyecto

# Planificación

*En este capítulo veremos la planificación temporal del proyecto y sus desviaciones.*

## 6.1 Planificación inicial

La idea de este trabajo surgió a finales del curso 2020 - 2021, de manera que, al iniciar el curso siguiente, ya teníamos planteados qué hitos seguiríamos y los tiempos que les daríamos a cada uno para ser llevados a cabo, teniendo en cuenta que Isabel cursaba todas las asignaturas del primer cuatrimestre de cuarto y que Matthew se encontraba de Erasmus en Noruega, lo que hizo que planteáramos entregar el proyecto en septiembre del curso que empezaba.

El proyecto ha sido dividido, inicialmente, en 7 hitos:

| Hitos | |
|---|---|
| **Análisis de los requisitos respecto a las tecnologías escogidas** | 01/09/2021 a 31/10/2021 |
| **Diseño del proyecto** | 31/10/2021 a 30/11/2021 |
| **Familiarización con el Motor Gráfico y el Framework escogidos** | 01/12/2021 a 30/01/2022 |
| **Primera versión fases Sprint Planning y Desarrollo (combate)** | 01/02/2022 a 30/04/2022 |
| **Pulir mecánicas y fases Sprint Review y Sprint Retrospective** | 01/05/2022 a 30/06/2022 |
| **Finalizar el desarrollo del prototipo** | 01/07/2022 a 31/07/2022 |
| **Análisis y aplicación del feedback obtenido por usuarios piloto** | 01/08/2022 a 31/08/2022 |

Tabla 6.1: Planificación temporal de los hitos

- **Análisis de los requisitos respecto a las tecnologías escogidas:** Para este bloque se estimó un mes, ya que queríamos tener tiempo para poder investigar bien las tecnologías a utilizar y redactar la base de requisitos lo más fielmente posible al juego deseado.
- **Diseño del proyecto:** Para tener un buen diseño sólido del proyecto y evitar errores a posteriori en el desarrollo o funciones no necesarias de este, era fundamental pensar en los distintos detalles y pensar cómo los distintos componentes del proyecto debían comunicarse con el Game Engine y con el framework. Por ello, estimamos un mes.
- **Familiarización con el Motor Gráfico y el Framework escogidos:** Se estimaron dos meses para este hito debido a varios motivos. Era la primera vez que trabajábamos con el framework y pensamos hacer los tutoriales propuestos para un futuro desarrollo más fluido. Otra de las razones fue la situación académica en

la que transcurriría, ya que entre diciembre y enero transcurren las vacaciones de Navidad y también las últimas semanas del primer cuatrimestre, donde la exigencia por entregas y exámenes de diversas asignaturas aumenta.

- **Primera versión fases Sprint Planning y Daily Scrum:** Debido a que, tal como planteamos el videojuego, la mayor parte del desarrollo y de las distintas mecánicas se encontraban en las dos primeras fases del juego, además de ser algunas de ellas bastante complejas y requerir de funciones muy específicas del nuevo framework. Por este motivo, decidimos hacer una estimación de 3 meses para la primera versión de estas.

- **Pulir mecánicas y fases Sprint Review y Sprint Retrospective:** Para este bloque, se estimaron 2 meses porque las mecánicas dependían en parte de funciones ya desarrolladas en el bloque anterior y su diseño era más simple, aún teniendo cierta complejidad. Por este motivo, determinamos que podríamos tenerlas en este tiempo.

- **Finalizar el desarrollo del prototipo:** Para este bloque estimamos tan solo un mes porque estaba centrado en unificar y pulir las funcionalidades ya existentes y no se iban a implementar cambios demasiado grandes si no surgían complicaciones. Además, transcurriría en julio y el hecho de que en esa fecha ya no se impartieran clases era beneficioso.

- **Análisis y aplicación del feedback obtenido por usuarios piloto:** debido a que la idea inicial era entregar el trabajo de fin de grado para la convocatoria de septiembre, decidimos dejar el último mes para pulir el juego con pequeños ajustes extraídos de encuestas a los usuarios piloto y detección de bugs..

## 6.2 Informe temporal

El cuarto hito del proyecto, es decir, el hito de desarrollo de una primera versión de las fases de Sprint Planning y de Desarrollo (Combate), sufrió un retraso de 4 meses. Esto hizo que se cambiara la fecha de todos los hitos y la entrega de trabajo pasase a diciembre. A los problemas mencionados anteriormente por lo que se hizo una estimación al alza, se le suman dos factores:

- **Problemas con el Framework seleccionado:** muchas de las funcionalidades del sprint principal alargaron su tiempo de desarrollo debido a la inexperiencia con

el framework y a problemas internos de este, como la integración con scripts externos o los problemas para acceder a su base de datos por la pobre documentación. Más adelante, detallaremos varios de sus problemas y las soluciones que implementamos para avanzar en el desarrollo.

- **Prácticas de empresa de mayor extensión que las previstas:** Isabel inició las prácticas curriculares a mitad de febrero, de mes y medio de duración, que al final se alargaron más de lo inicialmente previsto debido a un convenio con la empresa.

| Resumen temporal del proyecto | |
|---|---|
| **Fecha de inicio** | 01/09/2021 |
| **Fecha de finalización prevista** | 01/09/2022 |
| **Fecha de finalización** | 28/11/2022 |
| **Carga de trabajo semanal estimada** | 10 horas |
| **Total planificadas** | 600 horas |
| **Total horas Matthew** | 300 horas |
| **Total horas Isabel** | 300 horas |
| **Total real** | 600 horas |

Tabla 6.2: Tiempo y planificación

## 6.3 Desglose de tareas

En la tabla tabla 6.3 A continuación, hacemos un desglose temporal de las tareas realizadas a lo largo del proyecto. Debido a la extensión del proyecto y al notable número de bloques en el que se han dividido, hemos decidido hacer este desglose en tareas grandes.

Para cada tarea del desglose, tenemos en cuenta el número de horas totales destinadas a ella, diferenciando qué alumno se las dedicó.

| Tareas y tiempo (Parte 1/3) |
|---|
| **Análisis de los requisitos respecto a las tecnologías escogidas** |
| Búsqueda de tecnologías |
| Matthew 5h |
| Isabel 5h |
| Análisis de los requisitos |
| Matthew 10h |
| Isabel 10h |
| **SUB-TOTAL** 30h |
| **Diseño del proyecto** |
| Modelo de dominio |
| Matthew 10h |
| Isabel 10h |
| **SUB-TOTAL** 20h |
| **Familiarización con el Motor Gráfico y el Framework escogidos** |
| Tutoriales en ORKFramework |
| Matthew 10h |
| Isabel 10h |
| Ajustes básicos de ORKFramework |
| Matthew 20h |
| Isabel 20h |
| **SUB-TOTAL** 60h |
| **Primera versión fases Sprint Planning y Daily Scrum (combate)** |
| Desarrollo de Sprint Planning |
| Matthew 5h |
| Isabel 65h |
| Desarrollo del Daily Scrum |
| Matthew 55h |
| Isabel 0h |
| **SUB-TOTAL** 125h |

Tabla 6.3: Tiempo dedicado para cada tarea (1/3)

| Tareas y tiempo (Parte 2/3) | |
|---|---|
| **Pulir mecánicas y fases Sprint Review y Sprint Retrospective** | |
| Sprint Review | |
| Matthew | 5h |
| Isabel | 20h |
| Sprint Retrospective | |
| Matthew | 5h |
| Isabel | 25h |
| Pulir mecánicas Sprint Planning y Daily Scrum | |
| Matthew | 30h |
| Isabel | 0h |
| **SUB-TOTAL** | 85h |
| **Finalizar el desarrollo del prototipo** | |
| Unificar las fases del Gameflow | |
| Matthew | 0h |
| Isabel | 5h |
| Solución de bugs | |
| Matthew | 10h |
| Isabel | 0h |
| **SUB-TOTAL** | 15h |

Tabla 6.4: Tiempo dedicado para cada tarea (2/3)

| Tareas y tiempo (Parte 3/3) | |
|---|---|
| **Análisis y aplicación del feedback obtenido por usuarios piloto** | |
| Matthew | 15h |
| Isabel | 5h |
| **SUB-TOTAL** | 20h |
| **Otras** | |
| Documentación de la memoria | |
| Matthew | 65h |
| Isabel | 70h |
| Reuniones | |
| Matthew | 40h |
| Isabel | 40h |
| Preparación de la exposición | |
| Matthew | 15h |
| Isabel | 15h |
| **SUB-TOTAL** | 240h |
| **TOTAL** | 600h |

Tabla 6.5: Tiempo dedicado para cada tarea (3/3)

# Costes

*En esta sección vamos a detallar los costes del proyecto.*

## 7.1 Resumen de costes del proyecto

| Resumen de costes | |
|---|---|
| **Costes del personal** | 4.568,57 € |
| Sueldo bruto | 3.484,8 € |
| Costes sociales | 1.083,77 € |
| **Costes indirectos** | 360 € |
| Amortización | 130 € |
| Otros costes | 230 € |
| **SUB-TOTAL** | 4.928,57 € |
| IVA (21 %) | 1.035 € |
| **TOTAL** | 5.963,57 € |

Tabla 7.1: Resumen de costes

## 7.2 Costes del personal

Los costes de personal para la empresa, se componen del sueldo bruto y los costes de seguridad social, que se calculan a partir del sueldo bruto. Debido a que somos dos alumnos en el proyecto, todos los cálculos los realizaremos para dos trabajadores.

Para hacer el cálculo del sueldo bruto del proyecto, debemos determinar tanto el número de horas empleadas en el desarrollo del mismo como el sueldo bruto a percibir por hora. A la hora de calcular las horas, debemos tener en cuenta que cada crédito del grado corresponde a 25 horas de trabajo. Sabiendo que el trabajo de fin de grado es equivalente a 12 créditos, podemos determinar que serán 300 horas programador teniendo en cuenta el trabajo de un alumno. Partiendo de la base de que, además, este proyecto se ha realizado entre dos alumnos, este número ascendería a 600 horas programador.

Por otra parte, para el cálculo del sueldo bruto, tendremos en cuenta que la media se sitúa en 14,10 € por hora para un programador promedio en España. Conociendo estas cifras, el resultado del cálculo se puede ver en la tabla 7.2.

Los gastos de la Seguridad Social se situa entre el 30 % y el 35 % del salario bruto que recibe trabajador y se compone de varias categorías, cuyo porcentaje queda distribuído de manera diferente dependiendo del tipo de contrato. Para nuestros cálculos,

| Cálculo del sueldo bruto a percibir por el proyecto | | | |
|---|---|---|---|
| **Salario anual** | **Salario por hora** | **Horas del proyecto** | **Sueldo bruto (proyecto)** |
| 25.746,6 € | 14,10 € | 600 | 8460 € |

Tabla 7.2: Sueldo bruto a percibir por el proyecto

tomaremos la aproximación de 31,1 % del salario bruto. El desglose por categorías se puede consultar en tabla 7.3.

| Costes sociales | | | |
|---|---|---|---|
| **Grupo** | **A percibir por el trabajador (€)** | **%** | **Cantidad (€)** |
| Contingencias comunes | 8460 | 23.6 | 1996,56 |
| Desempleo | 8460 | 6.7 | 566,83 |
| F.O.G.A.S.A. | 8460 | 0.2 | 16,92 |
| Formación profesional | 8460 | 0.6 | 50,76 |
| **TOTAL** | | 31.1 % | **2.634,06 €** |

Tabla 7.3: Desglose de costes sociales

## 7.3 Costes indirectos

### 7.3.1 Amortización

A los costes de personal, debemos añadirle los costes derivados del hardware y del software empleados en el proyecto. En cuanto a software, debemos tener en cuenta el precio de la adquisición de las licencias del Framework empleado: ORKFramework, cuyo precio fue de 50 euros por una licencia. Es decir, 100 euros para las dos licencias. La amortización máxima anual de un programa informático es del 33 %. Teniendo en cuenta estos datos, el cálculo de la amortización del software se ha desglosado en la tabla 7.4.

Respecto al hardware, tenemos que tener en cuenta el valor del equipo y la amortización máxima anual de un ordenador, que es el 26 %. Como en el proyecto hemos empleado dos equipos de 1000 y 700 euros respectivamente, hablamos de un valor del equipo de 1700 euros en total. Encontramos el desglose de este cálculo en la tabla 7.5.

| Coste del software | |
|---|---|
| **Valor del software** | 100 € |
| **Amortización máxima anual** | 33 % |
| **Años** | 1 |
| **Amortización** | 33 % |
| **Valor de amortización** | 33 € |

Tabla 7.4: Coste del software utilizado

| Coste del hardware | |
|---|---|
| **Valor del equipo** | 1.000 € |
| **Amortización máxima anual** | 26 % |
| **Años** | 1 |
| **Amortización** | 26 % |
| **Valor de amortización** | 260 € |

Tabla 7.5: Coste del equipo utilizado

### 7.3.2 Otros costes indirectos

Otra parte de los costes indirectos la conforman la electricidad y el acceso a Internet, necesarios para el correcto funcionamiento de los equipos. Debemos considerar la actual crisis del gas, que ha producido un notable ascenso del precio de la luz durante el desarrollo del proyecto, aunque se ha experimentado un descenso en el último mes. Estimaremos un coste de electricidad medio 0,35 € y una tarifa de internet mensual de 27 €, desglosamos los gastos en la tabla 7.6.

| Otos costes indirectos | | | | |
|---|---|---|---|---|
| | **Costes estimados** | **Unidad** | **Cantidad** | **Cantidad (€)** |
| **Electricidad** | 0,35 | €/hora | 600 horas | 210 € |
| **Acceso a internet** | 27 | €/mes | 12 meses | 324 € |
| **TOTAL** | | | | 534 € |

Tabla 7.6: Coste de la electricidad y el acceso a internet

## 7.4 Reservas de contingencia

Es necesario, para el proyecto, trazar un plan de contingencia económico para poder afrontar los desvíos temporales de nuestro desarrollo. Para ello, tendremos en cuenta una posible variación del 5% del tiempo, es decir 30 horas de trabajo de las 600 planificadas originalmente.

| Reservas | | | | |
|---|---|---|---|---|
| | **Coste** | **Unidad** | **Cantidad reservada** | **Cantidad** |
| **Salario del desarrollador** | 14,10 | €/hora | 30 horas | 423 € |
| **Costes sociales** | 31,1 | % | 423 € | 131,55 € |
| **TOTAL** | | | | 554,55€ |

Tabla 7.7: Reservas de contingencia

## 7.5 Beneficio industrial

Para poder calcular un presupuesto completo, es también necesario determinar el beneficio industrial, es decir, el beneficio que obtendría la empresa que llevase a cabo este proyecto. Para calcularlo, tendremos en cuenta que suele rondar el 20% de los costes del proyecto.

| Beneficio industrial | | |
|---|---|---|
| **Coste** | **Porcentaje** | **Beneficio** |
| 5.963,57€ | 20% | 1.192,71€ |

Tabla 7.8: Beneficio industrial

## 7.6 Presupuesto del proyecto

El presupuesto del proyecto es la suma de sus costes, su reserva de contingencia y su beneficio industrial. A continuación, realizamos el cálculo con las 3 cifras que hemos calculado previamente en esta sección.

| Presupuesto | |
|---|---|
| **Costes** | 5.963,57 € |
| **Reserva de contingencia** | 554,55€ |
| **Beneficio industrial** | 1.192,71€ |
| **TOTAL** | 7.710,83€ |

Tabla 7.9: Presupuesto del proyecto

# Parte III

# Análisis

# Requisitos del producto a construir

array

*En esta sección listaremos los requisitos de nuestra aplicación.*

## 8.1 Requisitos Generales

| **RG-000 Aprendizaje de SCRUM** |
| --- |
| Como desarrollador,<br>quiero que el objetivo principal al que debe de estar enfocado el juego sea el aprendizaje de la metodología SCRUM y, por lo tanto, refleje elementos propios de la misma,<br>para que se trate de una gamificación. |

| **RG-001 RPG por turnos** |
| --- |
| Como desarrollador,<br>quiero que el juego se oriente al género rpg por turnos,<br>para poder facilitar el símil con scrum. |

| **RG-002 Tipos de jugabilidad** |
| --- |
| Como desarrollador,<br>quiero que el juego tenga dos tipos de jugabilidad: combate (trabajo) e interfaces de usuario (reuniones de scrum),<br>para poder facilitar el símil con scrum. |

| **RG-003 Secuencias similares a SCRUM** |
| --- |
| Como desarrollador,<br>quiero que el juego refleje las distintas reuniones para llevar a cabo un proyecto con scrum: Sprint planning (interfaz) -> Daily Scrum (combate) -> Sprint Review(interfaz) -> Sprint Retrospective(interfaz),<br>para que la dinámica del juego se asemeje a scrum lo mejor posible. |

| **RG-004 Turnos de trabajo SCRUM** |
| --- |
| Como desarrollador,<br>quiero que los turnos de trabajo o combates reflejen métricas y elementos propios de SCRUM, como los calendarios Niko-Niko o los tableros Kanban y se vean afectados por ellos,<br>para poder facilitar el aprendizaje de scrum. |

## 8.2 Requisitos de Información

| **RI-000 Información sobre los roles** |
|---|
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre los roles: nombre (Developer, Quality assurance, Tester, UI Designer o Deployment), descripción y la lista de habilidades que se aprenden con ese rol,<br>para que distintos personajes con distintos roles puedan realizar distintas acciones. |

| **RI-001 Información sobre los miembros del equipo** |
|---|
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre personaje combatiente: nombre, su superrol (rol asignado), rol actual (clase), nivel que tiene en cada rol, cantidad de motivación, cantidad de energía, ataque, ánimo, ánimo máximo, estabilidad, productividad, el estado (stage) en el que se encuentra, si es o no Scrum Master, su estado Niko-Niko y el sprite que lo representa en pantalla,<br>para que cada personaje sea único respecto a los demás. |

| **RI-002 Información sobre enemigos (tareas)** |
|---|
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre las tareas: nombre, rol asociado (rol al que es vulnerable), completitud, ataque, puntos de historia y sprite,<br>para poder representar correctamente las tareas que se realizan en un proyecto software. |

| **RI-003 Información sobre los estados de ánimo de un miembro del equipo** |
|---|
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre los estados de un personaje: Número (“1: Life Sucks”, “2: My life sucks”, “3: I’m great and you’re not”, “4: We’re great”, “5: Life is great”), cara con la que se le representará en el calendario Niko Niko, el porcentaje de ataque que modifica,<br>para poder representar correctamente los posibles estados de ánimo y coordinación del equipo durante el desarrollo del combate. |

| **RI-004 Información sobre las acciones (habilidades)** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre las acciones: nombre, descripción, rol asociado (tipo de daño), daño base, tipo de objetivo, radio objetivo, coste, si permite hacer más acciones en ese mismo turno y si la acción se aplica inmediatamente,<br>para que el usuario comprenda qué hace cada habilidad correctamente. |

| **RI-005 Información sobre el Sprint (combate)** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre un sprint: duración (n º de turnos), turno actual, tareas asociadas (Sprint Backlog) y gráfica Burndown del Sprint,<br>para poder procesar y mostrar al usuario de manera correcta las estadísticas del sprint actual en el que se encuentra. |

| **RI-006 Información sobre el proyecto** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre un proyecto: lista de tareas originales, lista de tareas asociadas (Product Backlog), duración (nº de turnos totales) y turno actual del proyecto,<br>para poder determinar el avance del proyecto. |

| **RI-007 Información sobre el calendario Niko-Niko** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre el calendario Niko-Niko: las entradas del calendario para cada miembro del equipo,<br>para que se pueda representar correctamente el calendario Niko-Niko y hacer el seguimiento de los estados de ánimo de los personajes. |

| **RI-008 Información sobre las entradas sobre el calendario Niko-Niko** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre cada entrada del calendario Niko-Niko: personaje al que corresponde la entrada, el emoticono asociado a la entrada y el turno del proyecto en el que se puso ese emoticono,<br>para que se pueda representar correctamente el calendario Niko-niko y hacer el seguimiento de los estados de ánimo de los personajes. |

| **RI-009 Información sobre la gráfica Burndown** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre la gráfica Burndown: sprint al que pertenece y entradas de la gráfica Burndown,<br>para que se pueda representar correctamente la gráfica y se pueda hacer un mejor seguimiento de la cantidad de trabajo realizada y la pendiente. |

| **RI-010 Información sobre las entradas de la gráfica Burndown** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre las entradas de la gráfica Burndown: número de puntos de historia restantes en ese punto y turno en el que se creó la entrada,<br>para que se pueda representar correctamente la gráfica y se pueda hacer un mejor seguimiento de la cantidad de trabajo realizada y la pendiente. |

| **RI-011 Información sobre las misiones** |
| --- |
| Como desarrollador,<br>quiero que se almacene la siguiente información sobre las misiones: tipo de misión, nombre, descripción, medida, objetivos y premio por cumplirla,<br>para representar posibles objetivos que el jugador puede proponerse durante la Sprint Retrospective. |

| **RI-012 Información sobre las misiones II** |
| --- |
| Como desarrollador,<br>quiero que los tipos de las misiones puedan ser los siguientes: misión de tiempo, misión de puntos de historia,<br>para poder clasificar las misiones en estos tipos. |

## 8.3 Requisitos Funcionales

| **RF-000 Reuniones y escenas** |
| --- |
| Como desarrollador,<br>quiero que las 4 reuniones de Scrum: sprint planning, daily scrum, sprint review y sprint restrospective, existan en estas dos escenas del juego, de tal forma que el sprint planning, review y retrospective se jugarán desde una escena de interfaz de usuario y las daily meetings tendrán lugar en la escena de combate,<br>para hacer un paralelismo evidente con Scrum. |

| **RF-001 Manejo de un equipo** |
| --- |
| Como desarrollador,<br>quiero que el jugador maneje un equipo de desarrollo cuyo propósito es el de completar tareas,<br>para hacer de punto de vista del jugador en el juego. |

| **RF-002 Tamaño del equipo de desarrollo** |
| --- |
| Como desarrollador,<br>quiero que el equipo de desarrollo tenga 5 miembros,<br>para simular el tamaño típico de un equipo de desarrollo Scrum. |

| **RF-003 Sistema de combate** |
| --- |
| Como desarrollador,<br>quiero que la escena de tareas sea un sistema rpg por fases, donde el jugador, durante su fase, decide qué hacen sus personajes y en qué orden ejecutarán sus acciones y, una vez todos hayan actuado, se cambia a la fase de la CPU, y una vez ella haya terminado sus acciones, volverá a ser la fase del jugador,<br>para tener un sistema de juego estratégico que permita al jugador planear qué acciones tomar en cada turno y en qué orden. |

| **RF-004 Daily Scrum** |
| --- |
| Como desarrollador,<br>quiero que, al comenzar un turno, el jugador decida qué acciones cada personaje debe tomar, que estarán determinadas por su rol,<br>para simular el hecho de que las daily scrums son reuniones destinadas a hablar de qué se ha hecho el día anterior (en este caso, analizamos la situación del "combate" tras el turno previo), y decidimos qué hacer en base a ello (en este caso, elegimos qué acciones realizará cada personaje). |

| **RF-005 Contador de tiempo Daily Scrum** |
| --- |
| Como desarrollador,<br>quiero que, durante la elección de acciones durante la escena de combate, el jugador tenga un contador de tiempo para la toma de decisión de las acciones donde, si supera cierto umbral, se reduce la productividad de estas acciones,<br>para representar el hecho de que las daily scrums deben ser cortas, y si se rompe esta regla, puede afectar a la productividad del equipo, además de que así creamos más la sensación de tensión que queremos crear durante el trabajo. |

| **RF-006 Motivación de un personaje** |
| --- |
| Como desarrollador,<br>quiero que la motivación de cada personaje determine su rendimiento a la hora de completar tareas, de tal forma que cuanta menos motivación tenga, menos es capaz de hacer,<br>para que el jugador trate de mantener la motivación de su equipo lo más alta posible en todo momento. |

| **RF-007 Energía de un personaje** |
| --- |
| Como desarrollador,<br>quiero que cada vez que un personaje realiza una acción, esta consuma energía, y que el personaje no pueda realizar acciones que le consuman más energía de la que le queda,<br>para evitar que pocos personajes muy buenos hagan toda la tarea del proyecto. |

| **RF-008 Acciones de un personaje** |
| --- |
| Como desarrollador,<br>quiero que los distintos personajes que el jugador manipula puedan hacer diversas acciones,<br>para interactuar en combate. |

| **RF-009 Cálculo de daño: Fórmula de daño** |
| --- |
| Como desarrollador,<br>quiero que, al realizar una acción ofensiva, la realización de la tarea objetivo se reduzca siguiendo la siguiente fórmula:<br>daño = (u.atk * (1 + (u.mot/u.maxMot)) * penalizacionDaily * u.productividad/100 * atkType,<br>donde u.atk es el ataque del usuario, u.mot es la motivación actual del usuario, u.maxMot es la motivación máxima del personaje, penalizacionDaily es la penalización de haber hecho una Daily Scrum demasiado larga, u.productividad es la productividad del usuario y atkType es la efectividad que tiene el rol de la acción sobre el enemigo,<br>para que las acciones de los distintos personajes completen las tareas en función del tipo de acción y el tipo de tarea. |

| **RF-010 Realización de una tarea** |
| --- |
| Como desarrollador,<br>quiero que, si la realización de un enemigo llega a 0, este desaparezca y se marque como completado,<br>para indicar cuando las tareas se terminan. |

| **RF-011 Contraataque de los enemigos** |
| --- |
| Como desarrollador,<br>quiero que los enemigos puedan realizar contraataques al recibir daño, y que estos contraataques reduzcan la motivación de los personajes del jugador,<br>para que el jugador se vea expuesto a inesperadas situaciones donde sus personajes puedan encontrarse bajos de motivación por constantes situaciones difíciles y se represente mejor el cansancio tras realizar tareas. |

| **RF-012 Cambiar de rol** |
| --- |
| Como desarrollador,<br>quiero que los personajes del jugador puedan rotar sus roles inmediatamente durante la Daily Scrum sin ningún tipo de coste en cualquier momento para realizar acciones de distintos roles rápidamente,<br>para dar versatilidad y flexibilidad al juego, dejando a los jugadores hacer uso de estrategias creativas, y además así se representa el que una persona con determinado rol no tiene su conocimiento necesariamente limitado al relacionado con dicho rol. |

| **RF-013 Nivel de un rol** |
| --- |
| Como desarrollador,<br>quiero que los roles tengan distintos niveles para cada personaje, de tal forma que un personaje sea más productivo ejerciendo de un rol en el que tenga mayor nivel,<br>para que el jugador pueda especializar a sus personajes en los roles que vea oportuno. |

| **RF-014 Productividad de acciones del rol** |
| --- |
| Como desarrollador,<br>quiero que las acciones asociadas a cada rol realicen una carga de productividad relativa al nivel del rol del personaje,<br>para representar de esta forma el que el nivel de un rol representa la maestría que tiene el personaje sobre ese rol, que un personaje no es bueno globalmente, sino que es bueno relativo a ciertos roles. |

| **RF-015 Bonus de superrol** |
| --- |
| Como desarrollador,<br>quiero que, al cambiar de rol, si el rol no coincide con el superrol asociado al personaje, las estadísticas se vean reducidas en un 20%,<br>para penalizar a jugadores que no decidan adecuadamente los roles principales de sus miembros durante el sprint planning. |

| **RF-016 Cambio de estados Niko-Niko** |
| --- |
| Como desarrollador,<br>quiero que los miembros del equipo expresen estados de ánimo al principio de cada Daily Scrum, en forma de un emoticono Niko-Niko, de tal forma que el estado Niko-Niko es determinado de la siguiente forma:<br>NikoNiko = Suelo([0..motivación/motivaciónMax] + [-1..1])<br>donde NikoNiko = -1 indica tristeza, 0 indica neutralidad, 1 indica felicidad y 2 indica máxima felicidad,<br>para reflejar cómo el ánimo de un grupo varía de forma constante en función de una serie de parámetros. |

| **RF-017 Relevancia de los estados de Niko-Niko** |
| --- |
| Como desarrollador,<br>quiero que los estados de ánimo afecten al rendimiento y motivación (triste -10% de motivación, neutral no aplica cambios, feliz +10% y muy feliz +20%) de los distintos miembros, teniendo en cuenta que cada miembro tiene una estadística estabilidad que determina la relevancia del estado Niko-Niko,<br>para incentivar el que el jugador persiga estados de mayor ánimo y además reflejar los principios de Scrum en este apartado. |

| **RF-018 Scrum master** |
| --- |
| Como desarrollador,<br>quiero que el scrum master del equipo tenga un conjunto de acciones adicionales independientes de su rol asociado, que consisten en ser acciones que aumentan la motivación de los demás miembros del equipo y ayudar a crear situaciones positivas en equipos scrum,<br>para hacer entender al jugador la relevancia del scrum master en un equipo de desarrollo software. |

| **RF-019 Calendario Niko Niko** |
| --- |
| Como desarrollador,<br>quiero que exista un calendario Niko-Niko que se actualiza con cada Daily Scrum que permite ver los estados Niko-Niko que los distintos miembros han tenido a lo largo del proyecto,<br>para dar un historial de los estados Niko-Niko de cada miembro del grupo al jugador. |

| **RF-020 Gráfico Burndown** |
| --- |
| Como desarrollador,<br>quiero que el jugador pueda acceder a un gráfico Burndown que muestra el avance actual del sprint para que en todo momento el jugador pueda ver si el avance que está haciendo en el sprint es adecuado o no. |

| **RF-021 Derrota** |
| --- |
| Como desarrollador,<br>quiero que, si el proyecto no se da por terminado en el número de turnos esperado, se considera que el jugador ha perdido,<br>para poner una meta mínima para el jugador. |

| **RF-022 Victoria** |
| --- |
| Como desarrollador,<br>Quiero que, si el proyecto se da por terminado antes de que se termine el número de turnos esperado, se considera que el jugador ha ganado,<br>para poner una meta mínima para el jugador. |

| **RF-023 Sprint planning** |
| --- |
| Como desarrollador,<br>quiero que, al comienzo de un sprint, aparezca una interfaz gráfica para realizar el Sprint Planning, donde, a grandes rasgos, se eligen las tareas a realizar, cómo realizarlas, y en qué tiempo,<br>para dar al jugador una opción de planificación previa a comenzar a realizar tareas. |

| **RF-024 Interfaces en Sprint Planning** |
| --- |
| Como desarrollador,<br>quiero que el Sprint Planning se realice mediante una interfaz de usuario con la información necesaria disponible,<br>para que el jugador pueda organizarlo más cómoda y rápidamente. |

| **RF-25 Sprint planning: Elaborar Sprint Backlog** |
| --- |
| Como desarrollador,<br>quiero que el jugador pueda seleccionar componentes del Product Backlog para realizarlos en el sprint (decidir a qué enemigos se enfrentará en el combate) ,<br>para que el jugador elija qué hacer en el próximo sprint, además de simular este proceso en la situación real de desarrollo software. |

| **RF-26 Sprint planning: Creación de enemigos** |
| --- |
| Como desarrollador,<br>quiero que, tras seleccionar los componentes que van a formar el sprint backlog, se creen distintas tareas basadas en cada uno de esos componentes (donde, en el formato rpg, cada una de esas tareas es un enemigo),<br>para que el jugador decida a qué fuerza enfrentarse en el próximo Sprint, basándose en sus recursos. |

| **RF-27 Sprint planning: Estimación de tareas** |
| --- |
| Como desarrollador,<br>quiero que las para cada tarea se indiquen sus Puntos de Historia, que determinan la dificultad de la tarea,<br>para dar al jugador la capacidad de determinar la dificultad de las tareas a las que se enfrenta. |

| **RF-28 Sprint planning: Elección de superroles** |
| --- |
| Como desarrollador,<br>quiero que el jugador elija para cada personaje, durante un sprint planning, un superrol que se mantendrá constante durante todo el sprint,<br>para elegir en qué tipo de acciones debería especializarse cada personaje para el sprint. |

| **RF-29 División de tareas** |
| --- |
| Como desarrollador,<br>quiero que el jugador pueda dividir las tareas que considere más duras en tareas más pequeñas,<br>para recompensar una buena estimación de las tareas, y mostrar la importancia de la división de estas. |

| **RF-30 Sprint Review: Bonificaciones** |
| --- |
| Como desarrollador,<br>quiero que durante el Sprint Review, se dé feedback acerca del avance del proyecto en forma de estadísticas del Sprint (combate), y que, si este es positivo, aumente la motivación de los miembros del equipo; consecuentemente, si el feedback es negativo, la motivación bajará,<br>para premiar al jugador que lleva una gestión que le permita maximizar la productividad. |

| **RF-31 Sprint Review: Bonificaciones II** |
| --- |
| Como desarrollador,<br>quiero que haya recompensas en motivación si se han superado los objetivos escogidos durante el Sprint Retrospective de un Sprint anterior (o penalizaciones si no se completan),<br>para premiar al jugador que hace uso correcto del Sprint Retrospective. |

| **RF-32 Sprint Retrospective: Misiones** |
| --- |
| Como desarrollador,<br>quiero que, al elegir qué hacer durante el Sprint Retrospective, se puedan escoger misiones de una lista para hacer en el siguiente sprint,<br>para que el jugador sea premiado si toma decisiones basándose en el progreso del Sprint pasado. |

| **RF-33 Sprint Retrospective: Misiones II** |
| --- |
| Como desarrollador,<br>quiero que, al elegir misiones durante el Sprint Retrospective, solo se permita escoger una misión de cada tipo,<br>para que no puedan completarse dos misiones del mismo carácter de forma simultánea. |

| **RF-034 Interfaces gráficas** |
| --- |
| Como desarrollador,<br>quiero que las distintas opciones y configuraciones de las distintas fases de scrum puedan realizarse mediante una interfaz gráfica,<br>Para que el jugador tenga una experiencia más agradable y pueda centrarse en aprender. |

**RF-035 Cambiar entre stages**

Como desarrollador,
quiero que los miembros del equipo cambien su stage en función de su ánimo siguiendo la siguiente norma:

- ánimo aumenta en 1 si motivación >= 50 %
- ánimo disminuye en 1 si motivación < 50 %
- si ánimo < 2, stage = 1
- si 1 < ánimo < 5, stage = 2
- si ánimo > 5, stage = 3,

para que los miembros del equipo vayan variando su stage en base a su motivación.

**RF-036 Alcanzar stages 4 y 5**

Como desarrollador,
quiero que los miembros del equipo puedan acceder a stages 4 y 5 siguiendo la siguiente norma:

- si ánimo de todo el equipo >= 7, stage = 4
- si ánimo de todo el equipo >= 9, stage = 5
- si el equipo se mantiene en stage 5 durante 3 turnos, volverá a stage 4
- el equipo no puede volver a stage 5 hasta que no pasen 3 turnos después de haber salido de dicho estado,

para que los stages con mayor beneficio tengan acceso difícil y representemos que son estados que requieren buena sintonía en todo el grupo.

| **RF-037 Relevancia de los Stages de los miembros del equipo** |
| --- |
| Como desarrollador,<br>quiero que, dependiendo del stage en el que se encuentra un miembro del equipo, su ataque se modifique de la siguiente manera:<br>Stage 1 reduce la estadística de ataque en un 50%, Stage 2 reduce el ataque en un 25%, Stage 3 no aplica modificaciones a las estadísticas, Stage 4 aumenta el ataque un 25% y Stage 5 aumenta el ataque un 50%,<br>para que los stages de los personajes tengan un efecto en la productividad del equipo. |

| **RF-038 Acciones consecutivas sobre una tarea** |
| --- |
| Como desarrollador,<br>quiero que una tarea que es objetivo de acciones consecutivamente en un mismo turno reciba la mitad de daño con cada acción consecutiva,<br>para representar que tener a varias personas trabajando en una sola tarea dificulta la productividad. |

| **RF-039 Realizar varias acciones en un mismo turno** |
| --- |
| Como desarrollador,<br>quiero que las acciones de cambio de rol y las acciones de Scrum Master no impidan al usuario hacer más acciones en ese turno,<br>para permitir turnos más dinámicos. |

| **RF-040 Límite de acciones de Scrum Master** |
| --- |
| Como desarrollador,<br>quiero que sólo pueda hacerse una acción de Scrum Master por turno,<br>para poner un límite al uso excesivo de las acciones de Scrum Master, además de añadir un incentivo de hacer uso de estas acciones en distintos turnos. |

## 8.4 Requisitos No Funcionales

| **RNF-000 Juego para Android** |
| --- |
| Como desarrollador,<br>quiero que el juego esté disponible para sistemas Android,<br>para poder jugar cómodamente. |

| **RNF-001 Interfaz intuitiva** |
| --- |
| Como desarrollador,<br>quiero que la interfaz del juego sea intuitiva,<br>para que el usuario pueda acostumbrarse rápidamente a su uso. |

| **RNF-002 Controles sencillos** |
| --- |
| Como desarrollador,<br>quiero que los controles del juego resulten sencillos,<br>para que el usuario pueda acostumbrarse rápidamente a su uso y se encuentre cómodo. |

| **RNF-003 Resolución de pantalla** |
| --- |
| Como desarrollador,<br>quiero que el juego esté disponible en resoluciones de Smartphone y de Tablet,<br>para que el usuario pueda ver todo el juego correctamente independientemente del tamaño de su pantalla. |

# Análisis de Scrum

*En esta sección describiremos las dificultades de Scrum y cómo planeamos enfrentarnos a ellas.*

## 9.1 Dificultades de Scrum

La principal dificultad que tiene Srum se resume en que es fácil de entender pero difícil de amaestrar[20]. El motivo es porque las ceremonias y prácticas que propone Scrum no son aplicadas correctamente si el equipo no es experto en Scrum, y no se consiguen los resultados que se pretende conseguir.

Esto se describe en 16 problemas[16], donde algunos son efectivamente por equipos inmaduros, pero otros son propios de metodologías ágiles.

Si Scrum se aplica en equipos que lo conozcan bien y en entornos adecuados donde realmente la solución sea aplicar Scrum, estos 16 problemas no deberían econtrarse.

### 9.1.1 Madurez del equipo

Analizando los problemas de Scrum y viendo casos donde Scrum fracasa o no funciona de forma efectiva, vemos que en gran medida esto ocurre porque el equipo y/o el Scrum Master no tienen conocimiento suficiente de Scrum y no lo aplican correctamente. En otras palabras, el equipo de Scrum no es maduro. Se puede dividir el aprendizaje de Scrum (y realmente cualquier competencia) en 4 pasos[26]:

1. **Incompetencia inconsciente:** El usuario no es competente, y no entiende qué debe hacer para mejor su capacidad.

2. **Incompetencia consciente:** El usuario no es competente, pero entiende por qué, y podrá avanzar en una dirección.

3. **Competencia consciente:** El usuario es competente siempre y cuando sea consciente de qué tiene que hacer para serlo.

4. **Competencia inconsciente:** El usuario es competente en toda circunstancia.

Un equipo que sea nuevo a Scrum probablemente sea **inconscientemente incompetente**[1], y necesitará a un Scrum Master al menos **conscientemente competente**[3] para que se asegure de que se aplica Scrum correctamente y su equipo empiece a ser **conscientemente incompetente**[2].

Esto quiere decir que hay una curva de aprendizaje en el equipo, que supone una demora a la productividad y el equipo tardará más en conseguir los incrementos que

se espera de ellos. Si esta es la primera vez que los miembros del equipo trabajan juntos, entonces se tendrán que enfrentar a una dificultad añadida hasta que la curva de evolución del equipo no mejore[28].

Aproximadamente el 50 % de los equipos de Scrum carecen de formación adecuada[16]. Esto es porque Scrum es una forma de trabajo emergente, y muchas empresas que han migrado a Scrum en los últimos años.

Las dificultades que se encuentran en equipos de Scrum inmaduros son algunos de los siguientes:

- **Interferencia en el trabajo:**

  Scrum incentiva que el equipo tenga capacidad de auto gestión y autonomía[25] porque facilita la adaptabilidad del equipo. Se sabe que los Scrum Masters, managers y *Product Owner (PO)* tienden a interesarse por el estado del equipo, y les preguntan con frecuencia, interrumpiendo su actividad, pero esto es una mala práctica que va en contra de la auto gestión y autonomía del equipo.

  Además se sabe que es muy frencuente que el PO modifique los requisitos, ampliando la cantidad de trabajo o aumentando exigencias, interfiriendo en la gestión del equipo. Scrum (y en general las metodologías ágiles) facilita la resolución de este tipo de situaciones por su adaptabilidad, pero no quita el hecho de que la creación de una interferencia entorpece al equipo y reduce la efectividad de Scrum.

  En equipos maduros, el Scrum Master debería no interferir en el trabajo de sus compañeros, y al mismo debería ser él el que lidie con las gestiones del PO, evitando que incremente el Product Backlog y con el manager u otros miembros de la organización.

- **Duración del Sprint:**

  La duración del Sprint puede variar mucho el número de incidencias que se encuentre el equipo. Se sabe que tienden a haber muchos menos problemas con sprints de 2 y 5 semanas que con los sprints de otras duraciones, siendo los sprints de 1 semana los más problemáticos[16].

  Equipos más maduros tienen la experencia para determinar cómo de largo deberían ser sus sprints en función de las tareas que hayan planificadas, del estado de su equipo en ese momento, de las exigencias del PO y de otros factores como la Ley de Parkinson[23].

### 9.1.2 Dificultades derivadas de las metodologías ágiles

Algunas de las dificultades de Scrum son consecuencia de cómo las metodologías ágiles gestionan el trabajo:

- **Necesidad de aplicar aumentos al final de cada Sprint**

  Se espera que en poco tiempo el equipo sea capaz de realizar entrega de trabajo que muestre diferencias sustanciales que satisfagan al PO.

  Esto tiende a hacer que el equipo priorice conseguir resultados, tal y como se indica en el manifiesto ágil[15], pero puede acarrear algunas consecuencias negativas:

  1. Falta de calidad de las entregas
  2. Problemas de integración
  3. Falta de calidad del código
  4. Problemas al realizar el despliegue

- **Falta de documentación**

  De nuevo, siguiendo el manifiesto ágil podemos observar que en Scrum se prioriza entregar un software funcional a crear una documentación que lo respalde.

  De nuevo, esto trae una serie de dificultades:

  1. Problemas a la hora de construir la arquitectura del proyecto
  2. Dificulta el mantenimiento del Backlog
  3. Dificulta la comunicación entre varios equipos
  4. Falta de trazabilidad del proyecto

- **Falta de planificación**

  Las metodologías ágiles optan por tener un trabajo flexible que no se ajusta a un plan fijo, pero esto puede acarrear consecuencias:

  1. No se respalda al equipo ante nuevas exigencias del cliente
  2. No hay un plan de riesgos

### 9.1.3 Otros problemas propios de Scrum

Además de los problemas derivados de las metodologías ágiles y de la falta de maestría de los miembros de Scrum en sus respectivos equipos, podemos encontrar aún algunos problemas más:

- **Demasiado idealista:**

  Siguiendo el manifiesto ágil, Scrum decide darle capacidad de auto gestión y autonomía a sus miembros, haciéndoles libres de elegir cómo trabajar por encima de imponerles procesos o herramientas.

  Sin embargo, no deberíamos asumir ni que todos los miembros del equipo tendrán esta capaciadad de aplicar bien su autonomía ni que, incluso si sí tienen la capacidad de hacerlo, siempre podrán aplicarla igual de bien. Sí, podríamos decir que si un equipo de Scrum es suficientemente maduro, sus miembros deberían ser capaces de aplicar bien esta autonomía, pero es idealista asumir que siempre lo harán igual de bien.

- **Ceremonias de Scrum:**

  Las ceremonias de Scrum son un elemento central en su funcionamiento, pero no podemos asumir que todos los miembros estarán siempre contentos con su implementación.

  Muy vinculado a la interferencia del trabajo que hemos mencionado antes, las ceremonias de Scrum pueden llegar a resultar negativas para algunos miembros por interferir diariamente en su ciclo de trabajo.

  Es altamente probable que algunos miembros sientan que asistir a todas las reuniones programadas es inútil o no les funciona, así que no rendirán suficientemente bien en esas reuniones, efectivamente haciendo que la reunión no cumpla su propósito y además interfiera en el trabajo de este miembro.

## 9.2 Nuestra respuesta a estas dificultades

Existen múltiples propuestas para corregir estos problemas que tiene Scrum, las mismas fuentes que mencionan las dificultades suelen mencionar también formas de enfrentarlas[16].

En nuestro caso quisimos enfocarnos a los problemas relativos a la madurez de los equipos, especialmente a los equipos que están empezando.

Basándonos en lo que mencionamos acerca de los pasos de aprendizaje de Scrum, entendemos que la mayor parte de equipos inconscientemente incompetentes [1] no tendrán acceso a un Scrum Master suficientemente competente que ayude a superar sus dificultades y a formar correctamente al equipo.

Para superar esta dificultad existen manuales, vídeos, cursos formativos, etc que forman a equipos que no tienen experiencia para que partan de una base mínima. Nuestra propuesta es ofrecer una formación a través de un videojuego, que se apoye en la interacción del usuario para una mejor formación, que sea divertida, atractiva y dé una experiencia, aunque virtual, de lo que supone tomar buenas y malas decisiones y la importancia de determinadas prácticas como las ceremonias.

En nuestro proyecto imitamos la secuencia de ceremonias de Scrum. Cada una de las ceremonias imita el propósito que sirve en una situación real de Scrum, con la intención de enseñar al jugador la importancia de cada reunión.

El juego se divide en dos escenas clave:

1. Planificación/cierre de Sprint
2. Desarrollo de Sprint

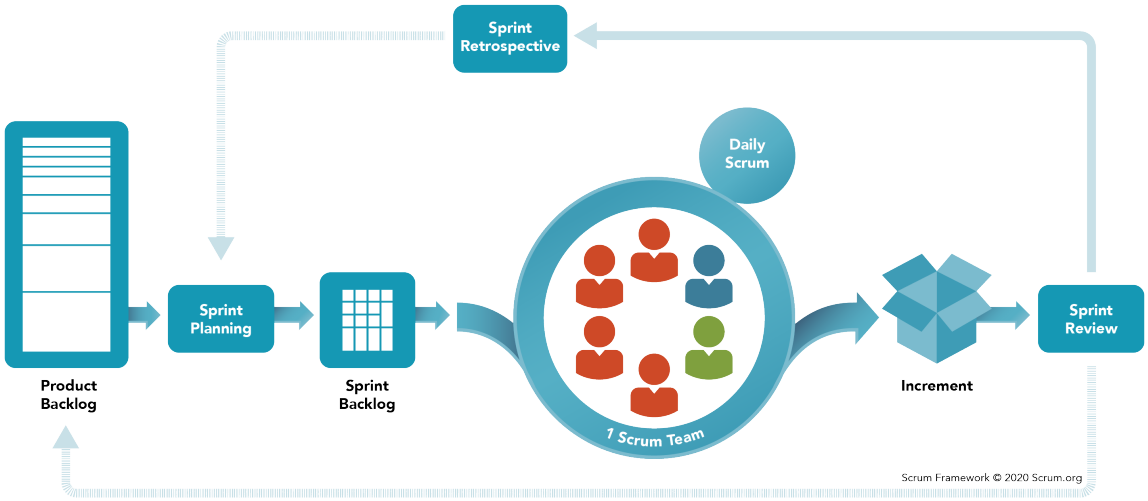



Figura 9.1: Estructura del Sprint

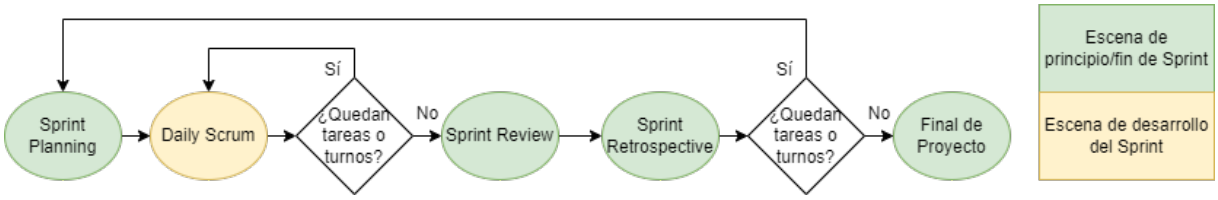



Figura 9.2: Carta ASM de nuestro producto

El Sprint Planning, Sprint Review y Sprint Retrospective se hacen en la primera escena, y las Daily Scrums se realizan durante la segunda escena. La primera escena es tranquila, donde se planifica el Sprint y observa el avance del proyecto. La segunda escena es ágil, el usuario debe decidir cómo avanzar el Sprint en el momento, siendo flexible respecto a lo que había planificado.

Durante el Sprint Planning el usuario interactuará a través de una interfaz que le permitirá seleccionar qué tareas se completarán en el Sprint, qué rol tendrá cada personaje (para acotar las tareas de preferencia de cada personaje), quien será el Scrum Master durante el Sprint y elegir lo que durará el Sprint.

Tras hacer el Sprint Planning, el usuario entra en la segunda fase. Nuestro producto se basa en un sistema llamado RPG por turnos, donde al principio de cada turno seleccionas las acciones de cada personaje, y se ejecutan en un determinado orden. Este menú de selección es lo que hemos encontrado equivalente a la Daily Scrum, donde

al principio del día se observa los avances de cada uno y se indica qué hará cada uno ese mismo día. El paralelismo es que al principio del turno se observa el estado actual de cada personaje y se decide qué acciones realizará cada uno en este mismo turno, similar a cómo se hace en la realidad, con la principal diferencia de que aquí el jugador elige por los personajes, cuando en la realidad cada miembro del equipo elige por sí mismo.

Una vez se acaban las tareas del Sprint o el tiempo del Sprint, se muestra la pantalla de Sprint Review, donde se observa cómo ha avanzado el proyecto, cuántas tareas quedan y cuánto tiempo queda de proyecto. Si el balance es positivo, es decir, quedan menos tareas que tiempo, la motivación de los personajes aumenta.

Por último, se muestra la Sprint Retrospective, donde se espera que el jugador evalúe su Sprint y decida qué hacer en base a su análisis. Nuestra forma de plasmarlo ha sido a través de seleccionar objetivos secundarios. Si en siguiente Sprint se cumplen los objetivos propuestos, la motivación aumentará, sino, disminuirá.

El proyecto finaliza cuando se acabe el tiempo (fracaso) o cuando se completan todas las tareas (éxito).

Otro elemento importante que se muestra en nuestro producto es el Scrum Master, que toma un rol de apoyo que aumenta la motivación y productividad del resto del equipo. Cuanta más motivación tengan los personajes, más eficaces serán trabajando. Esto hace que el rol de Scrum Master es clave para incrementar la capacidad de trabajo del equipo y poder vaciar el Product Backlog cuanto antes.

De esta forma, mostramos la importancia de cada una de las ceremonias de Scrum y el usuario adquiere experiencia ficticia de primera mano en estas ceremonias. Además, el usuario entiende el rol del PO y una de las funciones que tiene el Scrum Master para el equipo. Uniendo estos elementos en un juego da al usuario una experiencia divertida que sirve de introducción a elementos y términos clave de Scrum.

# Estudio tecnológico

*Tras el análisis del videojuego a conseguir, realizaremos, a continuación, un estudio tecnológico para escoger los medios más adecuados para su implementación de acuerdo con las necesidades detectadas en este. Debido a la naturaleza de nuestro proyecto y al tipo de producto a desarrollar, toda la tecnología escogida se empleará para todo el proyecto, por lo que no haremos distinción entre backend y frontend a la hora de realizar la búsqueda de esta.*

## 10.1 Game engine

En primer lugar, debido a que el proyecto se trata de un videojuego, lo primero que debíamos escoger es un game engine o motor de juegos adecuado con el que desarrollarlo, teniendo en cuenta el tipo de juego: un RPG en 2D cuya jugabilidad estará basada en interfaces de usuario. Dependiendo del motor y sus recursos disponibles, se determinan los demás aspectos que debemos tener en cuenta en cuanto a tecnología. Debe ser un game engine ya que, si queremos realizarlo de manera óptima, no nos es suficiente con un entorno de desarrollo que nos permita escribir código. Como somos aficionados a la creación de juegos de hacía tiempo, conocíamos varios motores de antemano. Para realizar un juego acorde a los requisitos, encontramos 3 que podrían ser compatibles:

### 10.1.1 RPGMaker

RPGMaker [10] es un game engine específico para videojuegos del género Role Playing Game, que contiene un editor de mapas, editor de base de datos y un editor de scripts compatible con varios lenguajes de programación, donde gran parte de la creación y configuración del juego se puede hacer mediante interfaz de usuario. Tendríamos en cuenta su versión más actualizada, llamada RPGMaker MZ, que fue publicada en agosto de 2020 y es compatible con Javascript. Ambos tenemos experiencia previa con este motor.

- Pros: RPGMaker presenta una curva de aprendizaje muy liviana, siendo muy intuitivo y permitiendo hacer gran cantidad de funcionalidades mediante su interfaz gráfica. Además, proporciona una base preparada a los juegos con base de datos, funciones básicas del combate, mapas, y una UI básica. Ambos alumnos lo hemos usado con anterioridad. Permite modificar esta base y agregar algunas funcionalidades mediante scripts empleando Javascript y exportar los juegos a móvil.

- Contras: En general, presenta poca flexibilidad. El principal inconveniente es modificar la base del proyecto creado por defecto porque tiene una estructura demasiado rígida. Algunos casos de esta rigidez son que tiene espacios de memoria preasignados en memoria para variables, eventos, etc. Otro problema que presenta es que su licencia es de pago, siendo su demo demasiado limitada para el alcance del proyecto. No es ideal para trabajar en grupo. Su documentación

está en japonés y, además, aunque su comunidad sea amplia, está mayormente enfocada a principiantes y es complicado encontrar material avanzado.

### 10.1.2 Unity3D

Unity3D [14], o simplemente Unity, es un game engine versátil, enfocado tanto a juegos de cualquier género con gráficos tanto en 2D como en 3D. Es uno de los líderes del mercado en la industria de los videojuegos y es empleado tanto por pequeños desarrollos indie como por grandes empresas. Matthew lleva varios años siendo usuario de Unity.

- Pros: Al ser uno de los más usados, la comunidad y el contenido educativo es de gran tamaño. Trabaja con scripts en C#, muy similar a Java, uno de los lenguajes de programación con los que más se trabaja en el grado. Tiene entidades específicas de interfaces de usuario, permitiendo trabajar con varias resoluciones y simplificar el trabajo con los gráficos. La licencia es royalty free: es gratuita si los ingresos del juego son menores a 100k euros y tiene herramientas para trabajo en grupo. Además, permite exportar a diferentes sistemas operativos, entre ellos Windows y Android. La documentación es accesible desde su página oficial y está en varios idiomas.
- Contras: La curva de aprendizaje es más elevada respecto a RPGMaker, siendo menos intuitivo que algunas otras opciones. Al ser tan versátil y no estar enfocado a un género de juegos específicos, no contamos con ningún tipo de base creada y todo lo debemos programar y configurar de cero. Además, Isabel no lo ha usado anteriormente.

### 10.1.3 Godot

Godot [5] es la alternativa Open Source de los motores de videojuegos más utilizada en el mercado. Aunque soporta gráficos 3D cada vez de más calidad, está más enfocado en videojuegos en 2D, ofreciendo una interfaz de usuario amigable.

- Pros: Permite debugging y principalmente utiliza un lenguaje propio, llamado GDScript, que es muy similar a Python, lenguaje de programación actual con a que se ha trabajado varias veces durante el grado, aunque es compatible con

lenguajes basados en C como C++ y C# [18]. Tiene un editor con la propia documentación incluida, lo que facilita el trabajo offline. Los proyectos son muy cómodos de exportar y se pueden trabajar mediante repositorios.

- Contras: La comunidad de Godot es ligeramente menor que en otras opciones. Aunque hay material educativo al respecto, es menos diverso. Además, a pesar de que permite exportar a diversas plataformas, el trabajo con los principales sistemas operativos móviles todavía es experimental. Ambos tenemos muy poca experiencia previa con el motor, así que la incertidumbre es mayor.

Finalmente, el motor de videojuegos elegido, valorando sobre todo el sistema final pensado, la versatilidad para las interfaces de usuario y scripts y la experiencia previa de ambos, decidimos que la herramienta más adecuada para el desarrollo del proyecto sería Unity. Escogeremos, en este caso, una versión LTS de Unity: 2019.4.20f1 LTS

**Comparativa de características de los motores de videojuegos**

| | **RPGMaker** | **Unity** | **Godot** |
|---|---|---|---|
| Género RPG | X | X | X |
| Exporta a Smartphone | X | X | - |
| Conocimiento previo | X | X | - |
| Base preparada | X | - | - |
| Flexibilidad | - | X | X |
| Licencia gratuita | - | X | X |
| Amplia comunidad | X | X | - |
| Documentación accesible | - | X | X |
| Fácil aprendizaje | X | - | - |

Tabla 10.1: Resumen de la comparativa de características de los motores de videojuegos

## 10.2 Framework

Debido al principal defecto de Unity sobre nuestro proyecto a desarrollar, el hecho de tener que hacer cualquier sistema de 0, decidimos buscar un framework enfocado a juegos RPG para contrarrestarlo, de tal forma que tuviéramos una base de la que partir para poder hacer un juego lo mayor completo posible en menos tiempo y poder centrarnos en la gamificación de Scrum.

Unity cuenta con un mercado, llamado Unity Asset Store [2], donde los usuarios de la comunidad pueden vender material propio para el uso en nuestros juegos: gráficos, herramientas personalizadas de Unity, frameworks... entre otros. Buscando en este mercado de la comunidad, nos aseguramos de que el framework que estamos estudiando sea para Unity.

Para encontrar las diferentes opciones disponibles para un framework rpg, usamos los términos: "rpg battle system", "rpg framework", "rpg engine", y "rpg builder". Debido a que las herramientas las han desarrollado individuales, generalmente será necesario realizar pagos, por lo que podría ser un criterio adicional para valorar las diferentes propuestas. Las reviews de otras usuarios también podrían ser un factor a tener en cuenta.

A continuación, analizaremos los diferentes candidatos que encontramos en Unity Asset Store.

**Opción 1. Blacksmith: Action 2D RPG Engine / Starter Kit [3].** Blacksmith fue el primer resultado que encontramos en la búsqueda. Su última actualización fue en septiembre de 2019 y, su precio, de 4 euros

- Como principal aspecto positivo, destacamos que es el más barato del mercado, además de que en el vídeo de su página se muestran multitud de opciones.
- Sin embargo, no es posible encontrar un proyecto demostración para comprobar sus capacidades y la única review de usuario disponible es de un usuario al que le fue regalado el producto, por lo que presenta demasiada incertidumbre.

**Opción 2. RPG Builder [9].** Su última actualización fue el mismo mes en el que realizamos esta búsqueda y, su precio, 174,65 euros.

- Afortunadamente, la página de la tienda de este framework tiene tanto vídeo de demostración como juego demo para poder explorar su potencial. Aparentemente, tiene la capacidad de automatizar bastantes procesos y cuenta con buenas reviews.
- Desgraciadamente, no es compatible con gráficos 2D y no está orientado a personas que sepan programar, lo que hace que sus scripts permitan poca personalización y fallen al extenderlos, además de que es excesivamente caro para lo que ofrece.

**Opción 3. Turn-based RPG Battle Engine 2D [13].** Es una de las opciones más baratas del mercado, tan solo costaba 8,92 euros. (Actualmente, su precio ha aumentado a 14,96)

- Como parte positiva, cuenta con vídeo de demostración, aparentemente es sencillo de entender y simple de modificar, tiene buena documentación y, lo que más nos llamó la atención, cuenta con atención al cliente.
- Como parte negativa, no hay ningún proyecto o juego de demostración para comprobar su funcionamiento, es algo restrictivo para los combates y, según algunos reviewers, la implementación de las funciones del framework es muy poco eficiente.

**Opción 4. 2D RPG Kit [1].** Por 60 euros, nos encontramos con 2D RPG Kit, que contiene actualizaciones periódicas.

- Por una parte, parece muy completo, incluye todo tipo de escenas y transiciones, posee demo para ver su potencial, está documentado y recibe soporte. Además, dice ser muy bueno para los principiantes.
- Sin embargo, es una opción de precio elevado para lo que ofrece.

**Opción 5. ORKFramework [6].** Por 100 euros, encontramos ORKFramework.

- Positivamente, este framework aparenta ser muy flexible y completo, compatible con programadores y no programadores y con documentación en su página oficial [7]. Además, cuenta con soporte garantizado, el creador responde a los correos relativamente rápido y hay un foro bastante activo en el que preguntar dudas, además de una comunidad que crea plugins gratuitos para el sistema.
- Permite varios tipos de combates y puede usarse tanto en juegos 2D como en juegos 3D. Para poder comprobar su capacidad y su usabilidad, incluye una demo y tutoriales del framework en su página oficial. Es lo más aparentemente similar a RPGMaker para Unity que llegamos a encontrar.
- Su única pega es su elevado precio.

**Opción 6. Seguir tutoriales específicos.** Otra opción que tuvimos en cuenta fue hacerlo nosotros mismos. Para ello, encontramos diversos tutoriales para construir nuestra propia base de un juego RPG. Entre los resultados, destacamos el tutorial creado por Pavcreations [8].

- Este tutorial nos permitía hacer nuestra propia base de rpg de manera gratuita. Hacer este tutorial de 0 nos obligaría a entender el framework propio desde sus cimientos, ahorrándonos tiempo de comprender un software ajeno, además de proporcionar material descargable para adelantar algunos procesos y la inclusión de menú, mapa y combate.
- Sin embargo, no incluía menú de pausa y, al tener que implementar todo de 0, el tiempo que tardaríamos en construir una base para el juego se podría prolongar de manera indeterminada y, además, no tendríamos ningún soporte, ya que nuestros tutores no manejan la tecnología de Unity y, al ser nuestro propio código, no tendríamos ningún respaldo de otro programador.

Finalmente, a pesar de su precio, nos decantamos por ORKFramework. Esta decisión vino respaldada por la gran cantidad de información que se ofrecía en su página inicial, como sus tutoriales, los proyectos de demostración que ofrecía y la que creímos que era una buena documentación. Sin embargo, el hecho que más nos inclinó a este framework fue la facilidad para contactar por su creador, bajo el pseudónimo de "Gaming Is Love", por correo electrónico y por su foro dedicado. Además, encontramos multitud de reviews positivas, lo que frente a la incertidumbre nos dio confianza.

Por suerte, a los pocos días de habernos decantado por esta herramienta, ORKFramework publicó una oferta en la que podríamos adquirir una licencia a mitad de precio, por 50 euros, en su versión ORK2.

Sin embargo, este framework, a pesar de ser el que utilizamos a lo largo del desarrollo del proyecto, resultó ser de bastante menos calidad de la que esperábamos, retrasando notablemente nuestro avance por problemas internos del mismo.

## 10.3 Lenguaje programación

Una vez escogido el motor gráfico, en este caso, Unity, faltaba escoger el lenguaje de programación más adecuado para el desarrollo. En este caso, la decisión vendría impuesta por el propio motor, ya que Unity solo es compatible con scripts escritos

en C# y ORKFramework no ofrecía ninguna posibilidad de añadir otro lenguaje de programación.

C#[18] es un lenguaje de programación orientado a objetos y componentes, propiedad de la empresa Microsoft, que permite crear aplicaciones que se ejecutan en .NET. Tiene sus raíces en la familia de lenguajes C, por lo que recuerda a C++. Además, al ser orientado a objetos, es muy similar a Java, lenguaje que se nos ha enseñado a lo largo del grado.

Como entorno de desarrollo para este lenguaje de programación, emplearemos el recomendado por el propio Unity para C# y que viene integrado por defecto, Microsoft Visual Studio [31], que permite una edición de los códigos más cómoda, ya que incluye las librerías de Unity y esto permite que realice sugerencias a la hora de escribir, de forma que se minimizan los errores y se aumenta la velocidad de desarrollo. Además, al estar integrado con el motor gráfico, podemos aprovechar su consola para mostrar resultados al mismo tiempo que ejecutamos el videojuego.

# Parte IV

# Diseño

# Diseño del Software

*En esta sección nos centraremos en el diseño de nuestro proyecto.*

## 11.1 Introducción

Nuestro proyecto se diferencia en gran medida a cualquier proyecto que se nos haya exigido realizar a lo largo del grado. Entre otras cosas, es de mucha mayor magnitud, por lo que decidimos que era mejor no realizar el proyecto desde nada, sino partir de un framework y haciendo uso de herramientas que abstraigan gran parte de la carga de trabajo.

Dividiremos las explicaciones en diseño de bajo nivel y diseño de alto nivel.

En cuanto a diseño de bajo nivel, explicaremos de forma concreta qué esperamos del framework elegido y cómo hemos utilizado lo que nos ofrece, aunque abstraeremos cierta información detallada y nos limitaremos a explicar sólo aquello de lo que hemos hecho uso de todo lo que ofrece el Framework.

Respecto al diseño de alto nivel, explicaremos cómo hemos utilizado la base que tenemos del framework y qué más componentes tendremos que implementar nosotros para plantear la implementación de las funcionalidades propias de nuestro proyecto.

## 11.2 Diseño a bajo nivel

Como ya hemos dicho, no entraremos en detalles de cómo se diseña nuestra aplicación a bajo nivel, pero sí que explicaremos qué requerimos que nuestras herramientas nos ofrezcan.

Al revisar nuestras distintas opciones, queríamos algo que nos permitiera tener una base de datos programada y, a ser posible, configurada, y que la interacción con ella esté abstraida, para así permitir que nos podamos enfocar en el diseño e implementación de más alto nivel.

El framework que elegimos es ORK Framework, que ofrece su propia base de datos que incluye muchísima información por cada entidad. Nosotros no necesitamos utilizar ni todas estas entidades ni todos los atributos de cada una de estas entidades.

Las entidades que sí hemos utilizado han sido las siguientes:

1. **Combatiente:** Los combatientes son la entidad más elemental, que mueven la elaboración de tareas.

   Tiene relaciones con casi todas las demás entidades que hemos usado, y tienen

una serie de atributos propios que hemos utilizado como son nombre, flags (si está en combate o no por ejemplo), métodos, etc.

Los combatientes que hemos usado son los miembros del equipo del jugador y las tareas a las que el jugador se enfrenta. Cada tipo de tarea es un combatiente distinto, cada tarea es una instancia de la tarea obtenida de la base de datos.

2. **Estadística:** Cada estadística de los combatientes debe ser reflejada en la base de datos.

   De cada estadística se guardan su nombre y si es consumible o no (por ejemplo, la motivación, la energía y la completitud son consumibles, pero el ataque y la experiencia no lo son).

3. **Crecimiento de estadísticas:** Cada personaje puede tener crecimiento de estadísticas. En este caso, lo hemos utilizado para las tareas, donde los puntos de historia (que en el backend se traduce en experiencia y niveles) miden sus estadísticas.

   Nosotros sólo hemos creado una entidad en esta tabla, y la hemos relacionado con todas las tareas.

4. **Rol:** Cada personaje tiene un rol, que puede usarse para determinar sus estadísticas, crecimiento de estadísticas, habilidades, etc. Cada rol tiene su propio nivel para cada personaje.

   En nuestro caso, hemos creado 6 roles, uno por cada rol del jugador y uno más para las tareas. Los roles de jugador determinan las habilidades que tiene cada personaje, y el rol de tarea no hace nada.

   Scrum Master no es un rol, ya que un personaje no puede poseer dos roles a la vez.

5. **Habilidad:** Las habilidades son todas las acciones que pueden emplearse durante una Daily Scrum (combate en términos del framework).

   De cada habilidad almacenamos su nombre, eficiencia, tipo de objetivo (usuario, aliado, grupo aliado, enemigo, grupo enemigo o todos), fórmula usada si cambia algún parámetro, estados causados, nivel de habilidad, etc.

   En nuestro caso el tipo de objetivo siempre es un enemigo a excepción de las habilidades de Scrum Master, utilizamos dos fórmulas, una de daño y otra de curación y el nivel de habilidad determina la eficacia de la misma.

6. **Tipo de Habilidad:** Entidad que ayuda a englobar varias habilidades bajo una misma catregoría.

Nosotros hemos creado una para cada rol, una para enemigos, otra para Scrum Master y una última para cambio de rol. El uso que le damos en nuestro caso es para poder englobar todas las habilidades de un mismo tipo bajo un tipo de daño y para que tengan distinta clasificación durante la Daily Scrum, haciendo que sea más intuitivo para el jugador donde encontrar cada habilidad.

7. **Tipo de daño:** Los personajes y/o roles tienen resistencias a determinados tipos de ataques y las habilidades pueden ser etiquetadas con estos tipos.

   Hemos creado un tipo de daño por cada rol, de tal forma que cada tipo de tarea tiene 100% de resistencia a todos los tipos menos a uno, y cada habilidad de jugador que hace daño es de un tipo de daño.

8. **Estado:** Los estados son etiquetas que se le dan a los combatientes en distintas situaciones, y luego el juego procesa estos estados de distintas formas en función de sus atributos (por ejemplo, el estado *Veneno* hará que un personaje pierda vida cada turno).

   Nosotros hemos creado un estado que no hace nada llamado *Scrum Master* que sirve para etiquetar a un personaje como Scrum Master, para luego poderle asignar las habilidades correspondientes.

   Además de eso, hemos creado un estado para cada Stage, que aumenta o disminuye el ataque del combatiente, y otros dos estados, uno llamado *Recuperación de PE*, que aumenta la Energía de un personaje cada turno, y otro llamado *Counter*, que aumenta al máximo la probabilidad de contraataque al recibir daño.

9. **Fórmula:** ORK Framework crea un objeto propio llamado fórmula. Es difícil deducir qué atributos contiene esta entidad, pero sirve para crear una secuencia de operaciones matemáticas que transforman un número haciendo uso de distintos parámetros.

10. **Variable:** Las variables son simples valores que pueden ser de tipo String, Bool, Float y Vector 3. Estos valores son almacenados en la base de datos y pueden ser empleados en numerosas partes del código.

Podemos ver cómo interactúan entre sí estos componentes mediante un diagrama simplificado (es decir, omitiendo componentes y atributos que no utilizamos) que hemos diseñado haciendo ingeniería inversa de ORK Framework.

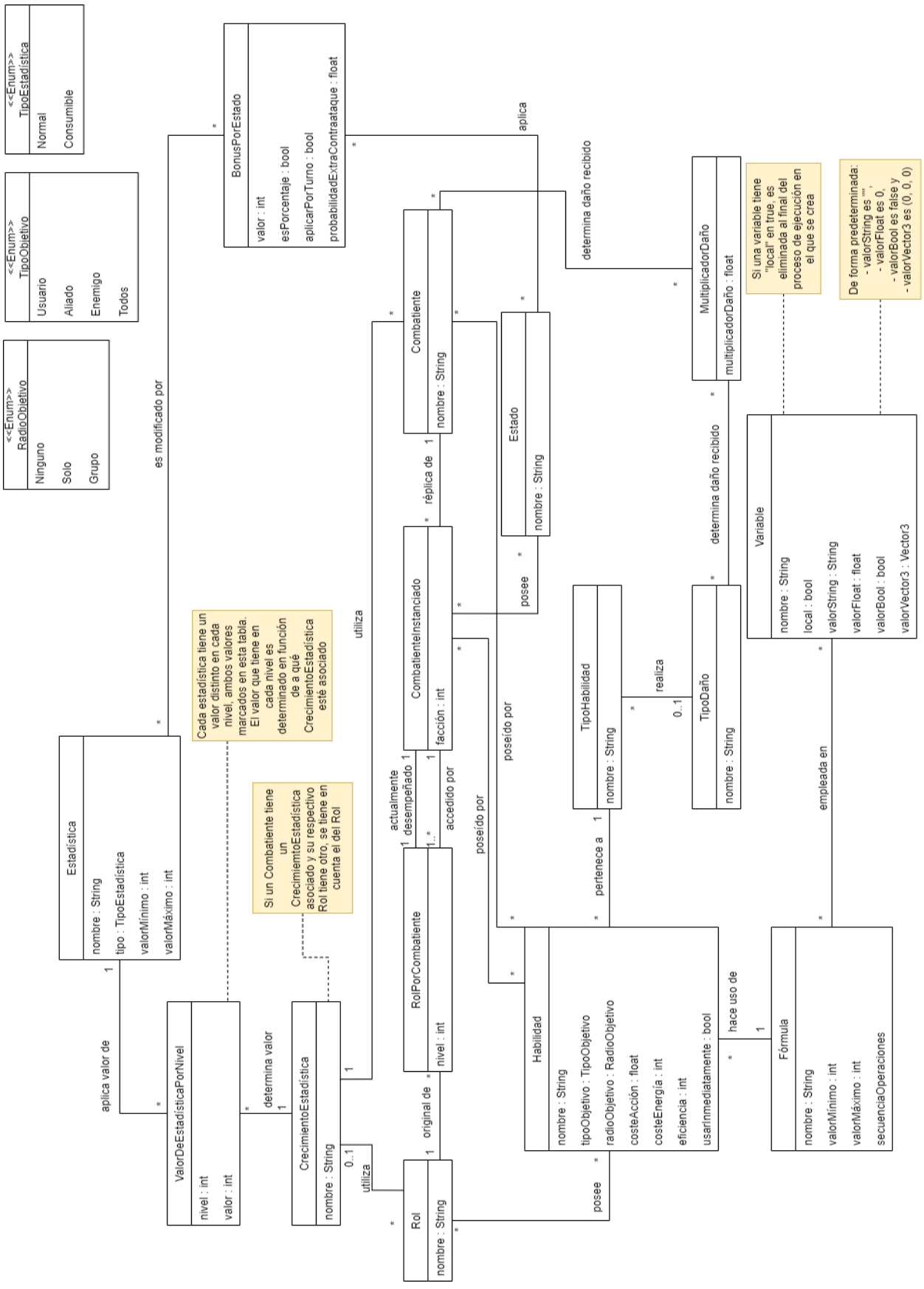



Figura 11.1: Diagrama simplificado de interacción de componentes en ORK Framework

A la entidad **rol** le añadimos un atributo booleano superrol. Determina cuál es el rol que fue originalmente elegido para el personaje al principio del Sprint (es decir, el rol que se elige al principio del sprint tendrá ese atributo puesto a true, y todos los demás lo tendrán en false). Un personaje cuyo rol activo es su superrol contará con una bonificación.

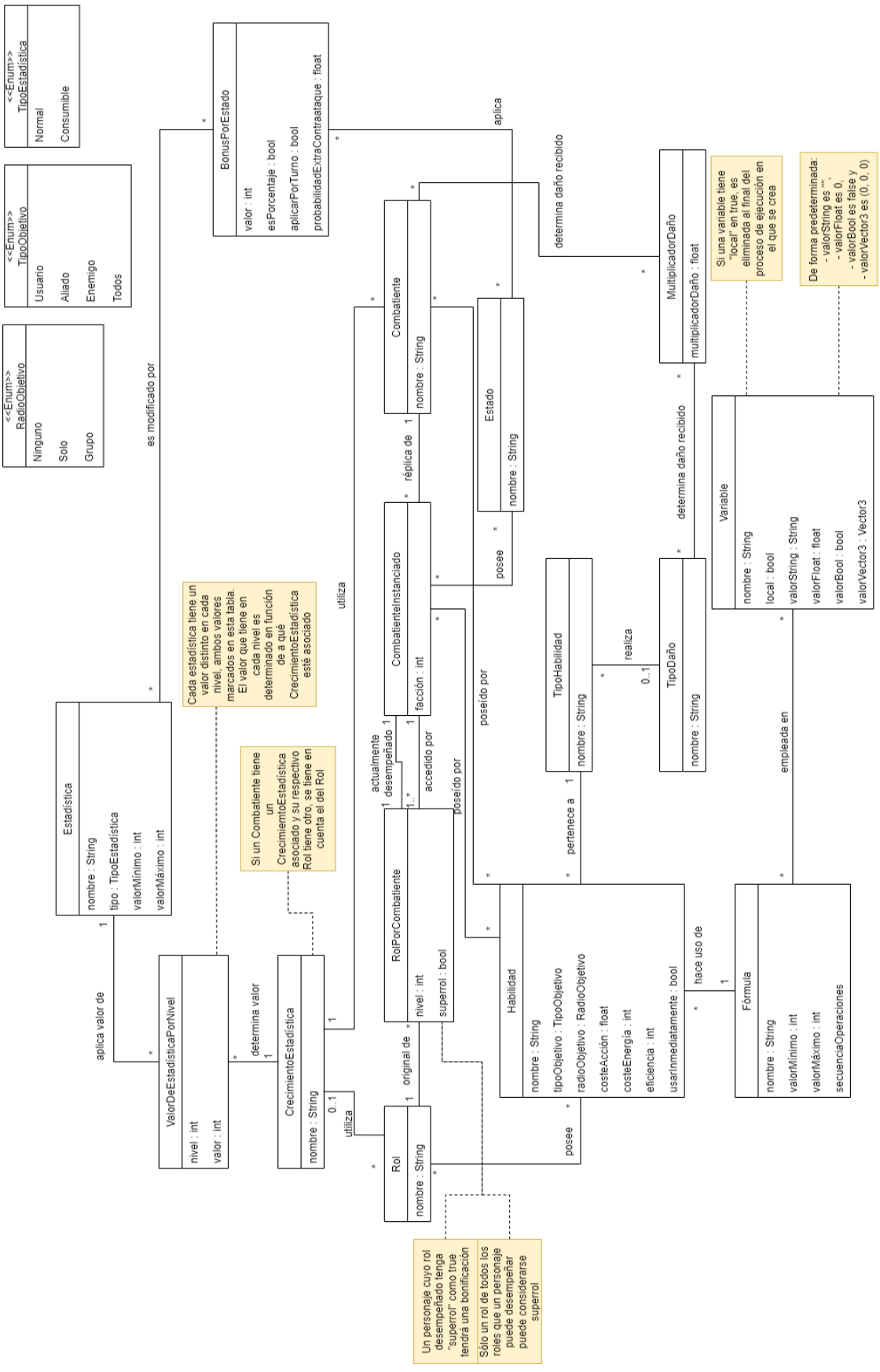



Figura 11.2: Diagrama simplificado de interacción de componentes en ORK Framework con componentes ampliados

Con esto diseñado, tenemos la base principal de nuestro proyecto.

Ahora vamos con el diseño de funcionalidades.

## 11.3 Diseño de alto nivel

Para organizar nuestro trabajo, decidimos dividir el proyecto en 4 partes conexas, una por cada ceremonia de Scrum. Cada ceremonia tiene propósitos distintos, y la forma en la que el usuario gestiona cada una es muy distinta.

Por este motivo, dividimos el diseño en 4 partes y luego las conectamos a través de algunos componentes comunes.

Para el diseño de esta sección abstraemos en gran medida las entidades que hemos mencionado en la sección anterior para simplificar los diagramas que vendrán a continuación.

### 11.3.1 Sprint Planning

Durante el Sprint Planning se realizan 4 actividades:

1. Construcción del Sprint Backlog

2. Selección de roles

3. Elección de Scrum Master

4. Duración del Sprint

Vamos a analizar uno a uno y ver qué requieren.

**Construcción del Sprint Backlog**

Para construir un Sprint Backlog necesitamos las sigientes entidades:

- El propio **Sprint Backlog**. Contendrá las tareas que se realizarán durante el Sprint, y no sirve otro propósito.
- **Las tareas.** ¿Qué necesitamos de ellas en el Sprint Planning?

  *L*os Puntos de Historia, ya que determinan la dificultad de la tarea. ¿Cómo traducimos eso a un juego RPG? En este caso existe una estadística comúnmente usada llamada *nivel*; cuanto mayor es el nivel, más poderoso es el enemigo. Existe otra estadística llamada *experiencia* que cuenta con una serie de umbrales, y cada uno marca un nivel nuevo. Es decir, cuanto mayor sea la experiencia, mayor es el nivel. La relación entre experiencia y nivel se construye en la entidad **CrecimientoEstadística**, pero abstraeremos esto en el diagrama que construiremos. Los Puntos de Historia que el jugador ve son una traducción de la experiencia de esa tarea. No es tampoco una relación 1:1, sino de 1:100, es decir, 1 Punto de Historia son 100 puntos de Experiencia. En definitiva, para poder implementar esta funcionalidad necesitamos que las tareas tengan una estadística de experiencia y otra de nivel, y los puntos de historia son extraídos de la experiencia (dividiendo la experiencia actual por 100).

  Además, tenemos que tener en cuenta el *rol al que son vulnerables* para poder realizar el Sprint Planning apropiadamente. Según hemos estudiado en la sección anterior, esto se implementa mediante una relación entre tipos de daño y combatientes, pero para esta sección lo simplificaremos a un atributo que determine la vulnerabilidad de la tarea en cuestión.

  Aún así, tenemos que tener una cosa más en cuenta. Existen 5 tareas en nuestro proyecto, una por cada rol al que es débil, pero instanciaremos múltiples versiones de cada tarea. ¿Qué atributos varían de una instancia a otra? Por lo que hemos visto hasta ahora, los Puntos de Historia deberían variar entre instancias. Aún así, todos usarán el mismo patrón de crecimiento de estadísticas, mismas habilidades, mismos roles, etc. Es decir, algunos de los atributos que hemos mencionado en la sección anterior.

  Por último, sabemos que podemos **dividir una tarea en dos**, reduciendo así sus Puntos de Historia. Esto deberá ser representado en el diagrama en forma de relación de esta entidad consigo misma.

- **El Product Backlog**. Similar al Sprint Backlog, contendrá tareas, pero en este caso serán tareas del proyecto entero. Las tareas serán desplazadas de un Backlog al otro.

Con estos componentes ya deberíamos tener el Sprint Backlog diseñado.

**Selección de roles**

Para implementar la selección de roles necesitaremos los **roles** de los miembros de nuestro equipo. Cada rol tendrá un nivel, un nombre, y tendrá acceso a un conjunto de **habilidades**. Además, debemos hacer que el rol elegido sea el **superrol** del personaje. También debemos mostrar el nivel de cada rol de cada personaje para que el usuario pueda elegir bien qué rol escoger para cada personaje.

También tendremos que crear la entidad de cada **miembro del equipo** que optará a los distintos roles. Para esta sección solo necesitaremos los nombres de los personajes, ya que no se hará uso de ningún atributo más de ellos.

Estos componentes son los que necesitamos para esta parte.

**Elección del Scrum Master**

Para la elección de Scrum Master bastará con añadir un atributo a la entidad *miembro del equipo* que hemos creado antes, un booleano que determine si es Scrum Master o no. Cada vez que se cambie el Scrum Master, se hace que dicho atributo sea true para el Scrum Master y false para el resto.

Tras esto, se revisa qué personaje es el Scrum Master, y se le asignan las habilidades oportunas, y al resto de personajes se les quita estas habilidades.

Dijimos que Scrum Master sería un estado, pero aquí lo representamos como un booleano para simplificar el diagrama.

**Duración del Sprint**

Para esto crearemos una entidad **Sprint**, con dos componentes, uno que determine en qué turno se encuentra, y otro que indique en qué turno debería considerarse terminado.

Con esto hemos explorado las 4 subsecciones que componen el Sprint Planning, y hemos identificado distintas entidades y atributos. A continuación se presenta el diagrama construido a partir de este análisis.

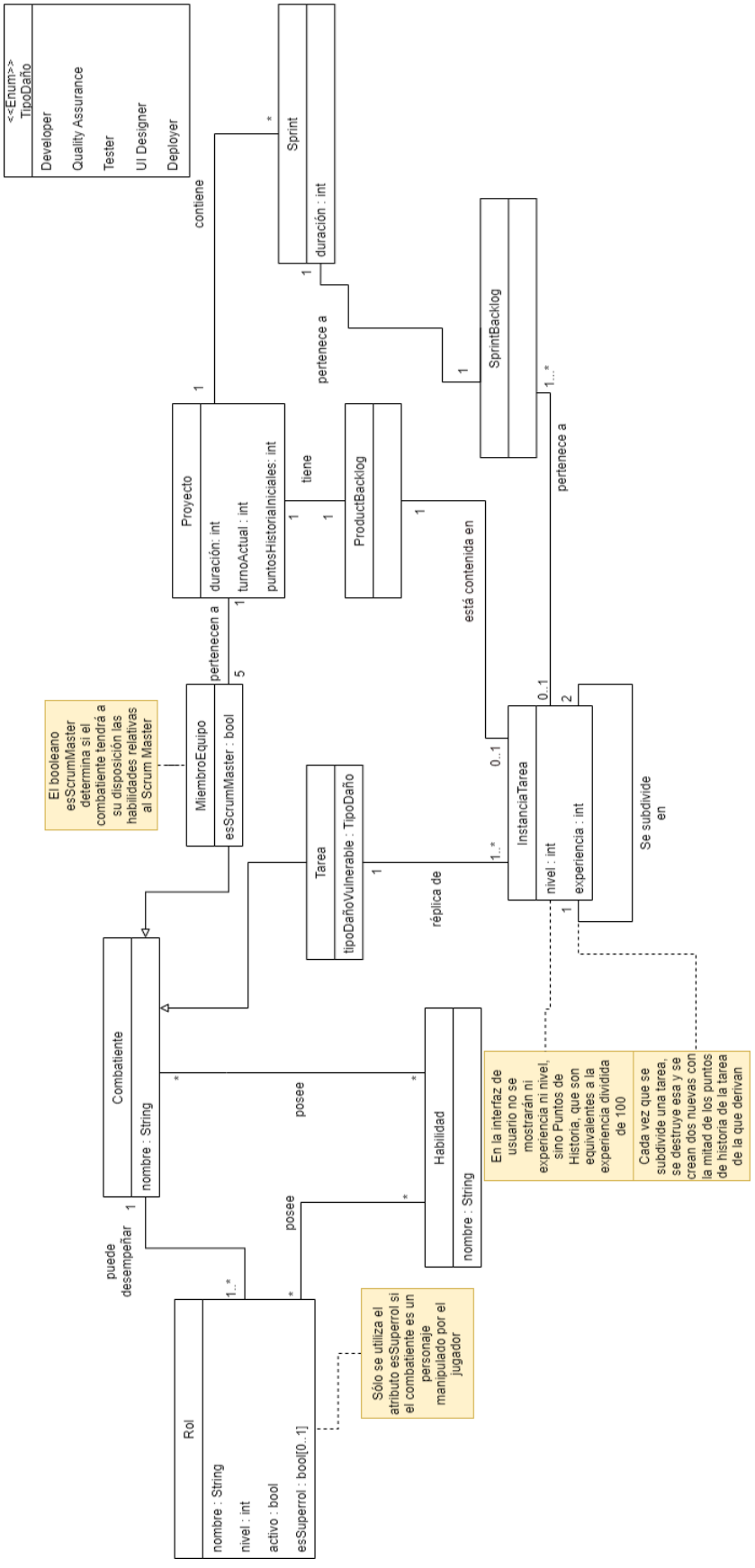



Figura 11.3: Diagrama de dominio de componentes del Sprint Planning

### 11.3.2 Daily Scrum

La Daily Scrum en nuestro proyecto se representa como la escena de *combate* de un juego RPG tradicional.

Se presentarán los personajes del jugador y las tareas seleccionadas para el Sprint. El Sprint (o *combate*) acaba una vez todas las tareas se den por finalizadas o cuando se acabe el tiempo.

La estructura de esta escena se divide en turnos. Cada turno consta de las siguientes fases:

1. **Comienzo del turno:** Se procesan ciertos eventos, por ejemplo, se verifican si los roles y superroles coinciden.

   Al comienzo del turno cada personaje indica su estado de ánimo con un emoticono. A esto se le llama **estado Niko-Niko**, y puede reducir o aumentar la motivación y la productividad.

2. **Selección de acciones a realizar:** Esta fase consta a su vez de las siguientes fases:

   *a*) Elección de personaje

   *b*) Elección de habilidad

   *c*) Elección de objetivo

   Una vez completado el proceso descrito (es decir, una vez se ha seleccionado al objetivo), se vuelve al principio.

Las acciones (relación entre usuario, habilidad y objetivo) se almacenan en un array, en el orden en el que se seleccionasen.

Además, queremos que haya un temporizador, y que si se supera el límite de tiempo, la productividad de los miembros del equipo se reduzca. El contador se reiniciará, pero cada vez que vuelve a superar el límite de tiempo, volverá a bajar la productividad del equipo. El temporizador se reincia al principio de cada turno, y las reducciones de productividad también, y se paraliza durante la ejecución de acciones.

Esta fase finaliza una vez todos los personajes han seleccionado alguna acción que les impida escoger más acciones.

3. **Ejecución de acciones:** Las acciones se procesan en el mismo orden en el que se añadieron al array, y son ejecutadas una tras otra. Si el objetivo de una acción no estuviera disponible en el momento de realizarse, se selecciona un objetivo posible aleatoriamente.

4. **Final del turno:** Al igual que al principio del turno, se procesan algunos eventos, como recuperación de energía, se actualizan los Stages de los personajes, etc.

   Al final de cada turno se añade una entrada al **gráfico Burndown**, que indica si vamos bien o mal en el Sprint.

Ya podemos identificar algunas entidades para nuestro diagrama. Algunas están repetidas de antes, como **miembro del equipo**, **tarea**, **rol (y superrol)**, **Sprint** y **habilidad**, pero hay algunas nuevas como **acción** y **temporizador**. Además, hemos añadido dos términos nuevos, **estado Niko-Niko** y **gráfico Burndown**.

Ahora necesitamos varias de estadísticas para los miembros del equipo y las tareas que determinen cómo influyen sus acciones al Sprint cada turno. Para ello, ampliaremos nuestras entidades **miembro del equipo** y **tarea**. Empezaremos ampliando la entidad **tarea**:

- *Ataque*: Determinará cuánta motivación pueden llegar a reducirle a nuestros personajes.

- *Completitud*: Determina cuánto queda para completar la tarea y darla por terminada. Tal y como se trabaja en ORK (y en juegos RPG en general), a este parámetro se le conoce comúnmente como *vida*, y cuando llega a 0, el combatiente abandona

el combate. Esto quiere decir que necesitamos tanto un parámetro que determine la vida restante como aquel que determina la vida máxima e inicial. En otras palabras, tenemos dos estadísticas: *Completitud* y *Completitud máxima*.

- *Número de veces golpeado*: Cada vez que una tarea es atacada, este número aumenta. Al recibir daño, esta tarea recibirá daño siguiendo la siguiente fórmula:

$$daoFinal = dao/(Nmerodevecesgolpeado + 1)$$

Esto se hace para que si distintos miembros del equipo atacan a una misma tarea, la efectividad de sus acciones sobre ella se reduzca.

A continuación, ampliaremos la entidad **miembro del equipo**.

1. *Stage*: Atributo que va de 1 a 5, donde cada número indica un estado del personaje respecto a su equipo. Cuanto menor sea el Stage, peor se encontrará en su grupo. Concretamente[30]:
   - Stage 1 - Life sucks: El miembro se siente inútil en el equipo y todo lo que lo rodea, y no ve capacidad de mejorar la situación.
   - Stage 2 - My life sucks: El miembro se siente inútil en el equipo, pero no ve que se aplique a todo lo demás.
   - Stage 3 - I'm great: El miembro se siente competente en el equipo, pero no necesariamente ve al resto del equipo igual de competente.
   - Stage 4 - We're great: El equipo siente que son competentes como equipo.
   - Stage 5 - Life is great: El equipo siente que el equipo y todo lo que lo rodea funciona bien, llevando a un estado de motivación síncrono.
2. *Ataque*: Capacidad base de *dañar* a las tareas, es decir, atributo que determina cuanta completitud realiza cada acción.
3. *Motivación*: Equivalente a la *vida* del personaje, es decir, se reduce cada vez que se recibe daño. Sin embargo, al contrario que en el caso de las tareas, el personaje no *muere* al tener su motivación en 0.
4. *Energía*: Recurso utilizado al realizar habilidades. Si no se tiene suficiente energía, el personaje no podrá realizar la habilidad indicada.

5. *Ánimo*: Atributo que va variando en función de la motivación del personaje al final del turno. Cuanto mayor sea el ánimo, mayor será el Stage, con un límite superior de Stage 3. Para alcanzar stages superiores, la moral de todo el equipo debe ser alta.

6. *Estado Niko-Niko*: Número que varía de -1 a 2, donde:

   - -1 = Triste
   - 0 = Serio
   - 1 = Feliz
   - 2 = Muy feliz

   Estos valores afectan a la productividad del personaje. Valores menores que 0 afectan negativamente, iguales a 0 no afectan, y superiores a 0 afectan positivamente.

7. *Estabilidad*: Atributo propio de cada personaje que determina cuanto fluctúa su productividad en función de su Stage, motivación y estado Niko-Niko.

8. *Productividad*: Atributo determinado en base a los Stages, a la motivación y al estado Niko-Niko, y su varianza es determinada por la estabilidad del personaje. Cuanto mayor sea la productividad, más *daño* hará el personaje al realizar acciones. En otras palabras, el *daño* producido depende de la productividad y del ataque, y cuanto más altos sean estos dos atributos, mayor será el *daño* causado.

En la sección anterior explicamos que cada estadística es realmente una entrada distinta de la tabla de estadísticas, implicando que habría dos entidades, una de personaje, y otra de estadísitca, pero para esta sección simplificaremos esto haciendo que cada estadística sea un atributo de la entidad personaje.

Si nos fijamos en cómo definimos la entidad **habilidad** en el diagrama de bajo nivel y vemos cómo se definió en el último diagrama, podemos observar que algunos de sus componentes no fueron incluidos. Esto es porque no se hace uso de ellos durante la Sprint Planning. Los incluiremos en este diagrama, y vamos a explicar qué uso le damos a cada uno:

- *TipoObjetivo:* Determina el tipo de objetivo de la habilidad. Existen 4 tipos:

  1. Usuario (ejemplo: cambio de rol)

2. Aliado (ejemplo: aumentar motivación)
3. Enemigo (ejemplo: realizar/dañar una tarea)
4. Todos (no hacemos uso de este tipo en nuestro proyecto)

- *RadioObjetivo:* Determina si la habilidad selecciona como objetivo a un combatiente dentro de su tipo de objetivo o a todos los que cumplan ese tipo de objetivo (por ejemplo, aumentar la motivación de un aliado o aumentar la motivación de todos los aliados).

- *CosteAcción:* Cada personaje tiene un punto de acción cada turno. Si una habilidad gasta ese punto de acción, ese personaje no podrá hacer más acciones ese turno, pero si una habilidad no gasta puntos de acción, el personaje podrá usar todos los que quiera ese mismo turno.

  En nuestro proyecto, las habilidades de cambio de rol no gastan puntos de acción, porque queremos que el jugador pueda cambiar el rol de su personaje y poder realizar otra acción en el mismo turno. Las habilidades de Scrum Master hacen uso de este parámetro, pero de forma personalizada. Las acciones de Scrum Master no gastan puntos de acción (igual que los cambios de rol), pero sólo podrá realizarse una por turno.

  Otra habilidad que no consume puntos de acción es el contraataque de los enemigos, ya que así pueden realizar infinitos contraataques por turno.

- *CosteEnergía:* Indica la cantidad de energía que gasta la habilidad. Si un personaje no tiene energía suficiente como para utilizar una habilidad, la habilidad no podrá ser seleccionada.

- *Fórmula:* Simplificación de la relación entre habilidad y fórmula que creamos en el primer diagrama. Cada habilidad hace uso de una fórmula para determinar el daño que causan. En nuestro caso, las acciones de ataque y las acciones de Scrum Master hacen uso de dos fórmulas distintas.

- *TipoDaño:* Simplificación de la relación entre la habilidad y su tipo de daño. Cada habilidad puede hacer un tipo de daño, que se utiliza para evaluar la cantidad de daño que hace a distintos enemigos basándose en su resistencia a ese tipo de daño. En nuestro caso, todas las tareas reciben 0% de daño de todos los tipos de daño a excepción del tipo de daño al que son vulnerables, y cada rol permite usar solo habilidades del tipo de daño con el mismo nombre que el rol en cuestión.

- *Eficiencia:* Un número por el que es multiplicado el resultado de la fórmula. En nuestro caso lo usamos para distinguir entre la habilidad básica de cada rol y su variante intensa. La habilidad básica tiene eficiencia = 1 y la intensa tiene eficiencia = 2, es decir, que la variante intensa es el doble de eficaz que la básica.
- *UsarInmediatamente:* Booleano que determina si una habilidad aplica sus efectos en el momento de seleccionarse. Esto se usa en las habilidades de cambio de rol y en los contraataques de las tareas.

También tendremos que mantener un contador de turnos para verificar cuando acabar el **Sprint**.

La entidad **acción** debe ir acompañada de otra llamada **buffer de acciones** que almacena todas las acciones del turno. Ninguna de estas entidades tiene atributos. Acción se relaciona con combatiente y habilidad, y buffer de acciones se relaciona con acción.

**Temporizador** tiene un contador de tiempo, un contador de veces que el tiempo ha sido agotado y un booleano que indique si debe pararse o no. El **calendario Niko-Niko** almacena los estados Niko-Niko de cada combatiente para cada turno. Por último, las **entradas del gráfico Burndown** son generadas cada turno, y almacenan la cantidad de puntos de historia restantes en ese turno específico, y el **gráfico Burndown** almacena todas estas entradas. El gráfico Burndown está vinculado al Sprint, de tal forma que se hace uno nuevo cada vez que se hace un Sprint nuevo, pero el calendario Niko-Niko se mantiene.

Habiendo analizado todos los componentes de una Daily Scrum, podemos extraer las entidades, sus atributos y sus relaciones. El diagrama resultante es el siguiente:

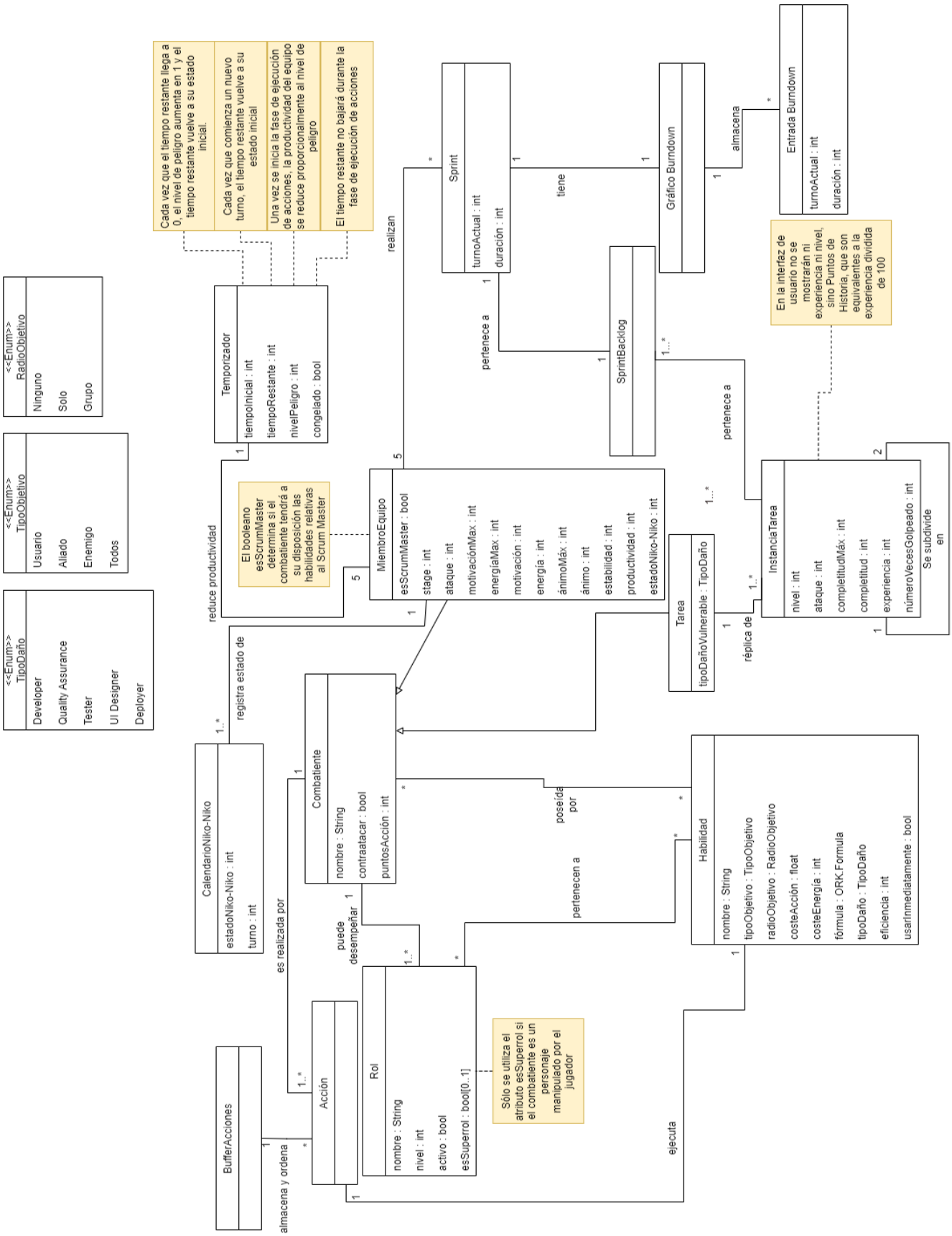



Figura 11.4: Diagrama de dominio de componentes del Daily Scrum

### 11.3.3 Sprint Review

En el Sprint Review, se evalúa lo bien que se ha realizado el Sprint, se revisa si se han realizado correctamente las metas propuestas en el Sprint Retrospective del último Sprint, y consecuentemente se penaliza o premia al jugador.

La **revisión** se realiza comparando los datos del Sprint resultante contra el proyecto. Los datos que se comparan son la cantidad de turnos restantes para el fin del proyecto contra la cantidad de puntos de historia restantes. Se hace de la siguiente forma:

$$ratioTiempo = tiempoRestante / tiempoTotal$$

$$ratioPuntosHistoria = puntosHistoriaRestantes / puntosHistoriaTotales$$

Si: $ratioTiempo > ratioPuntosHistoria \longrightarrow$ Evaluación positiva

Si: $ratioTiempo \leq ratioPuntosHistoria \longrightarrow$ Evaluación negativa

Es decir, necesitamos saber el *tiempo restante*, *tiempo total*, *puntos de historia restantes* y *puntos de historia totales*. Toda esta información podemos extraerla de los atributos del **proyecto** y del **product backlog** que definimos en la sección de Sprint Planning.

Para la revisión de misiones, primero debemos definir las misiones. Existen dos tipos de misiones:

- *Misiones de tiempo:* Exigen que se complete el Sprint en un número de turnos menor que X.
- *Misiones de puntos de historia:* Exigen que a lo largo del Sprint se completen más de X puntos de historia.

Para comprobarse las misiones, podemos extraer las propiedades *turnos empleados* y *puntos de historia completados* de **Sprint** y **Sprint Backlog**, ya que tenemos el atributo *turno actual*, que nos devolvería el número del último turno que ocurrió en el Sprint y podemos sumar los puntos de historia de todas las tareas que fueron completadas del Sprint Backlog, resultando en el número de puntos de historia completados.

Si los turnos empleados son menores a los turnos exigidos por la misión o los puntos de historia completados son mayores a los exigidos por la misión, la misión se consideraría completada.

Completar misiones y tener una evaluación positiva del Sprint aumentarán la motivación del equipo, y lo contrario reducirá la motivación del equipo.

Por último, durante el Sprint Review se hace una cosa más, que es revisar si el proyecto ha llegado a su fin o no. En caso de que no queden turnos o puntos de historia, el proyecto se acaba, con estado de fracaso o éxito respectivamente.

Mostremos el diagrama resultante:

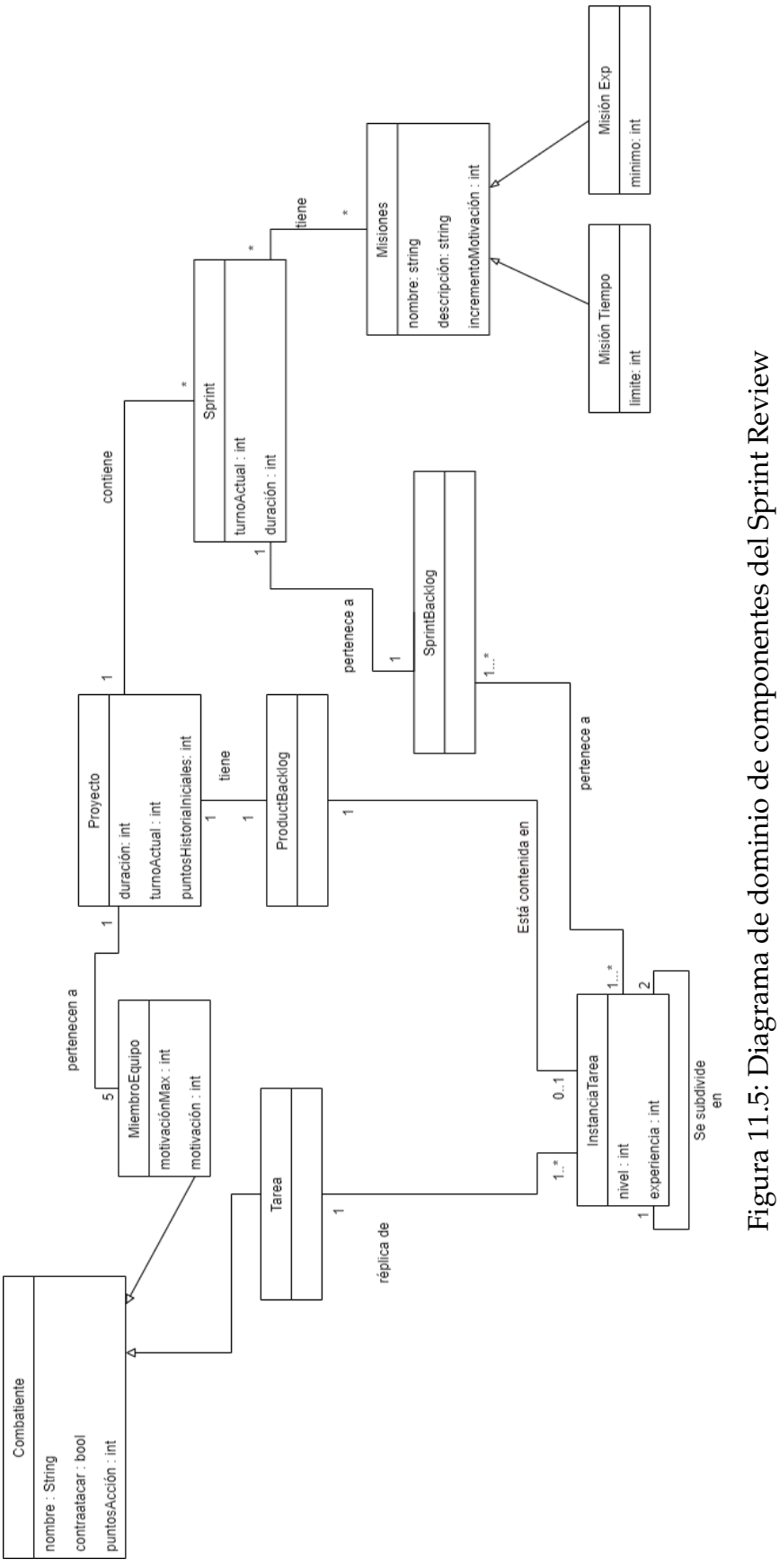



Figura 11.5: Diagrama de dominio de componentes del Sprint Review

### 11.3.4 Sprint Retrospective

Durante el Sprint Retrospective, el jugador puede seleccionar misiones y asignarlas al próximo Sprint.

Las misiones ya las definimos antes, y sus relaciones también. Sólo puede elegirse una misión de cada tipo por Sprint.

No hay ninguna entidad, atributo o relación que añadir, el Sprint Retrospective está englobado en el Sprint Review a efectos de diseño.

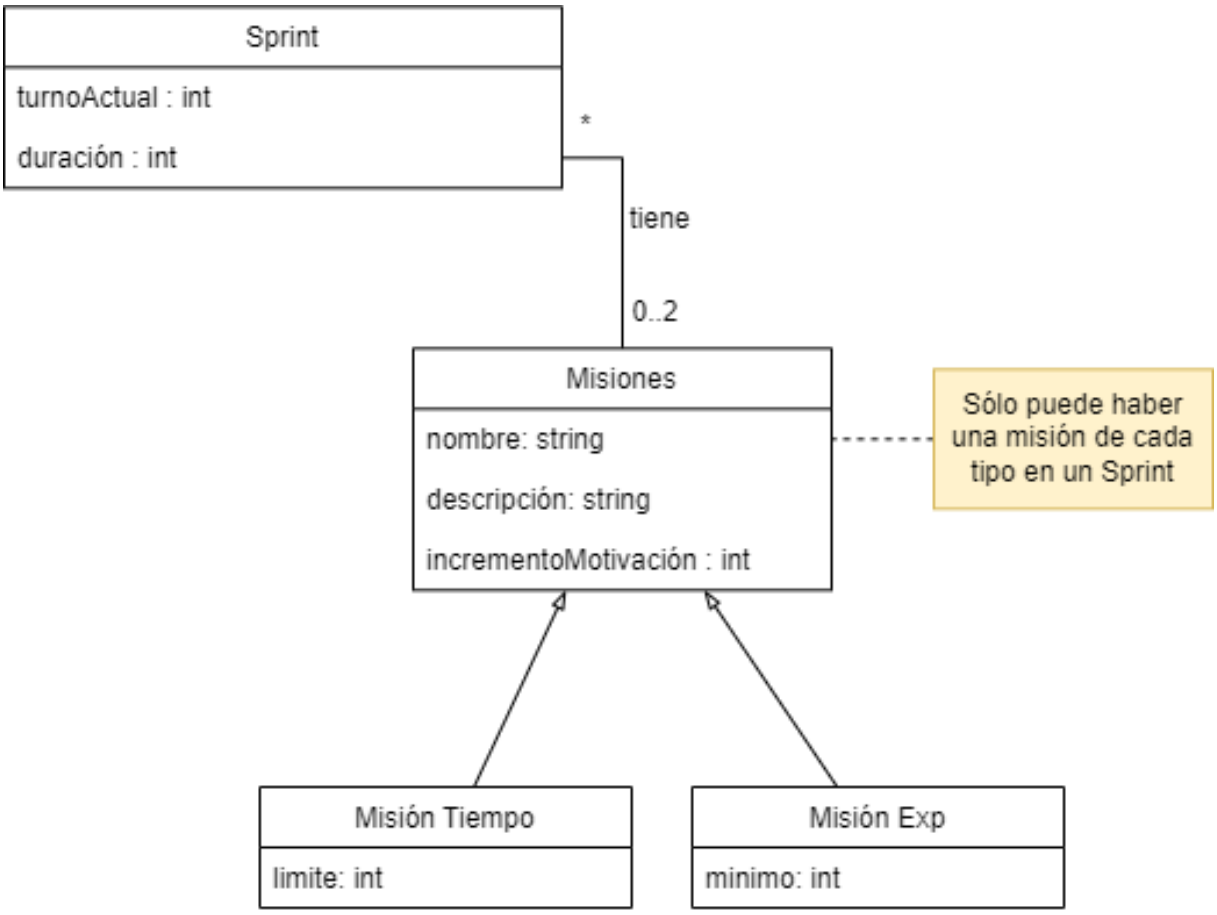



Figura 11.6: Diagrama de dominio de componentes del Sprint Retrospective

### 11.3.5 Diagrama de alto nivel resultante

Combinando los distintos diagramas que hemos realizado, finalmente obtenemos nuestro diagrama final:

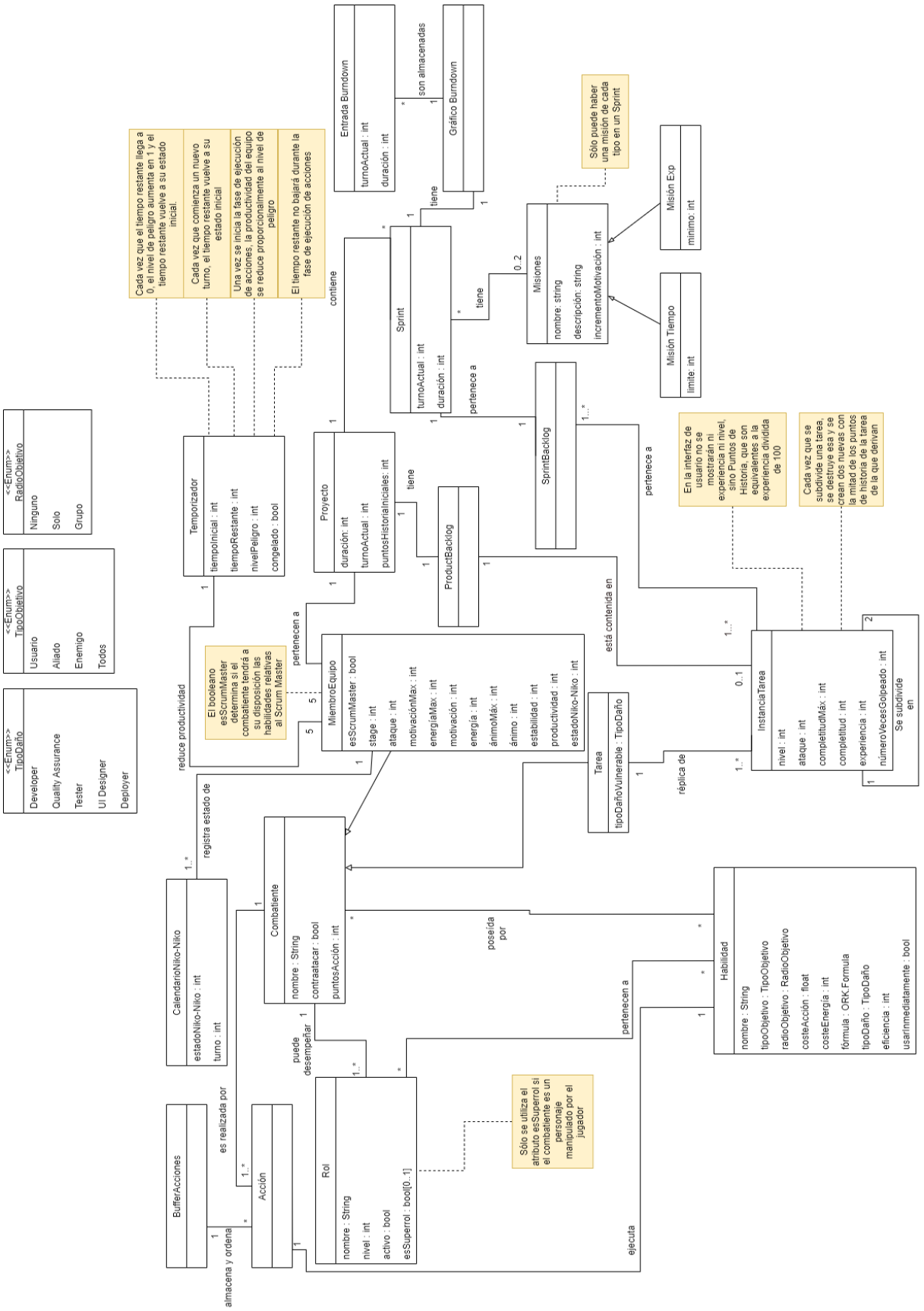



Figura 11.7: Diagrama de dominio completo

# Parte V

# Implementación

# Implementación

*En este capítulo, desarrollaremos cómo se ha llevado a cabo la implementación del juego, teniendo en cuenta el diseño analizado del capítulo anterior.*

## 12.1 Introducción

La línea divisoria entre backend y frontend es difícil de trazar en nuestro proyecto por la estructura de Unity, ya que no se pueden separar ambas partes en distintos proyectos Unity o repositorios. Un ejemplo claro se puede encontrar en varios de los scripts que, para su correcto funcionamiento, contienen funcionalidades atribuibles tanto a *backend* como a *frontend*. Por este motivo, hemos decidido no hacer distinción explícita entre ellas y unificar toda la implementación en un solo capítulo, describiendo así más fielmente cómo transcurrió el proceso.

Para la implementación, en general, se ha empleado la estructura proporcionada por ORK Framework (ORK2) para la base de datos y scripts en lenguaje C# para hacer una ampliación externa de la base de datos, además de las diferentes funcionalidades necesarias para poder representar adecuadamente las ceremonias de Scrum. También se han tomado de ORK las bases para poder implementar la Daily Scrum, es decir, los combates.

Respecto a las interfaces de usuario, Unity posee entidades específicas para su implementación a los que se pueden asociar códigos y diferentes objetos de manera cómoda para poder centrarnos en la representación fiel por pantalla de las funcionalidades. Además, permite implementar funciones simples desde su propio entorno para estas entidades, sin necesidad de programar líneas de código para los comportamientos más básicos.

Desarrollaremos, por orden de dependencia e importancia:

- Ajustes de ORK
- Funciones y atributos adicionales a la base de datos
- Sprint Planning
- Daily Scrum
- Sprint Review
- Sprint Retrospective
- Unificación de la jugabilidad

Durante el desarrollo del backend, surgieron muchos problemas con el Framework escogido. A lo largo del capítulo, explicaremos cómo fuimos paliando las carencias de

ORK y qué funciones básicas tuvimos que reinterpretar para el correcto funcionamiento del sistema.

## 12.2 AJUSTES BÁSICOS DE ORK

Para el correcto funcionamiento de los diferentes elementos y las librerías del framework con el motor de videojuegos que hemos seleccionado, es necesario realizar una serie de ajustes iniciales. Estos ajustes incluyen desde la población de la base de datos de manera básica con los elementos que nos interesan hasta la creación de eventos obligatorios para ejecutar correctamente todo el código.

En primer lugar, descargamos desde Unity Asset Store el contenido de nuestro framework. Ya en el editor de Unity, en el menú superior, hacemos click en Assets -> Import Package y pulsamos la opción *Custom Package* para buscar el archivo. De esta manera, cargamos el contenido del framework en nuestro nuevo proyecto y ya tendremos acceso a las funciones y elementos de ORK.

Siguiendo el tutorial que podemos encontrar en la página oficial del framework, para la correcta ejecución del proyecto, es necesario crear 2 escenas de Unity diferentes, una que funcionará como la pantalla de inicio y que permite iniciar o cerrar la aplicación y una segunda escena donde transcurrirá todo el contenido del juego.

Para la primera escena, tan solo tenemos que añadir un elemento del Framework llamado ORKGameStarter, que carga la base de datos del proyecto y las librerías para poder ejecutar correctamente todos los eventos y scripts de la escena principal.

Poblamos manualmente la base de datos para insertar los tipos de los personajes, los ataques y otras configuraciones básicas que se indican en los tutoriales para que los combates funcionen correctamente y crearemos el evento *StartEvent*.

Un evento de ORK es una entidad que permite programar un Script mediante una interfaz gráfica con bloques del propio framework ORK. Por otra parte, el StartEvent es un evento que se llamará al iniciar el juego desde la primera escena que creamos.

El evento StartEvent se encargará de hacer la configuración inicial necesaria sobre las funcionalidades que añadiremos más adelante, por lo que es necesario actualizarlo iterativamente a lo largo del desarrollo e instanciará a los personajes en la escena para poder trabajar sobre ellos.

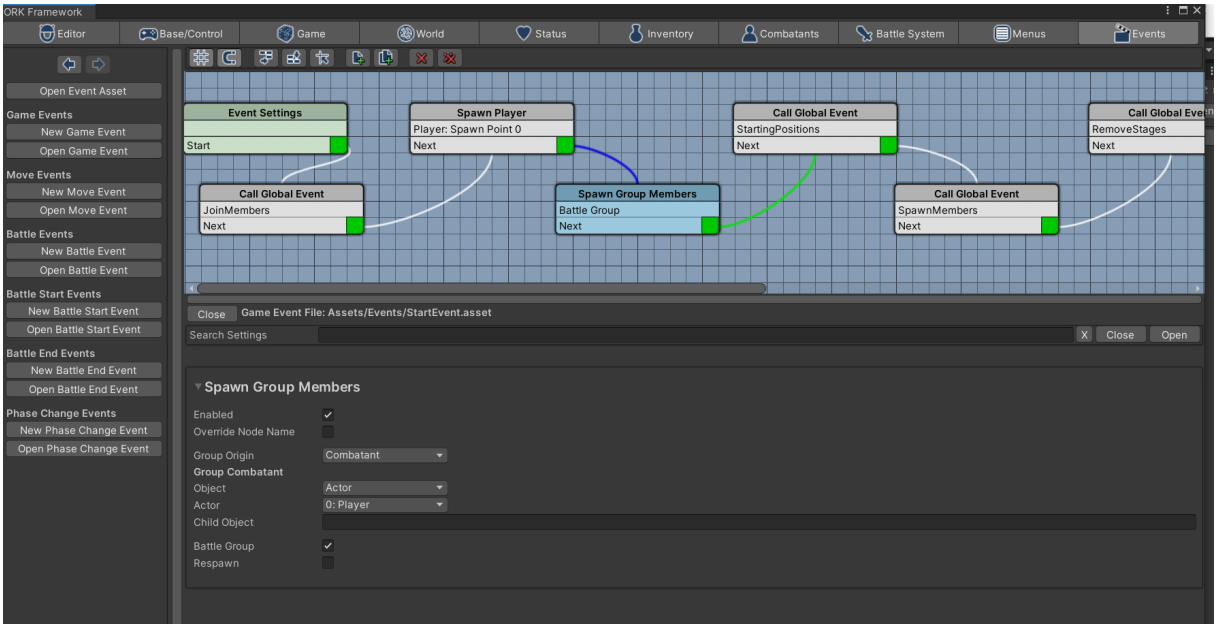



Figura 12.1: Editor de eventos de ORK mediante interfaz de usuario

## 12.3 Funciones y atributos adicionales a la base de datos

### 12.3.1 Atributo Superrol

Durante el Sprint Planning, elegimos el *Superrol* de los miembros de nuestro equipo. Un *Superrol* es una etiqueta con la que se marca a un personaje, y determina el rol inicial del personaje para el Sprint. Aparte del *Superrol*, existe otro término denominado *rol*, que determina el rol que está actualmente utilizándose por el personaje y que puede variar durante el Sprint. Si el *rol* coincide con el *Superrol*, el personaje tiene un aumento de estadísticas. A efectos prácticos, necesitamos tener 2 clases (roles) para los personajes, una que cambie solo durante el Sprint Planning y otra que pueda hacerlo durante el Sprint.

En la idea inicial descrita en el capítulo de diseño, considerábamos implementar esta idea mediante un nuevo atributo de tipo booleano para el rol, clase que ya está vinculada a cada personaje. Este nuevo atributo indicaría si ese rol concreto es el superrol que se ha seleccionado para el personaje durante el Sprint Planning. Sin embargo, esta idea resultaría inviable en la práctica, ya que ORK no da soporte para añadir nuevos atributos a sus datos.

Decidimos, en este caso, que podríamos añadir una segunda clase de rol a los personajes para representar el superrol y simplemente añadir, cuando fuera necesario com-

probar si fueran iguales, comparaciones entre ambas clases, haciendo de manera algo más compleja que la original el booleano. Sin embargo, ORK Framework tampoco permite añadir más de una clase rol a un personaje, por lo que debíamos seguir trabajando cómo implementar el superrol.

Como ORK Framework presentaba estas restricciones de base, decidimos añadir un atributo a su base de datos mediante un script externo e independiente del framework, que conectaríamos a este por medio de sus eventos. Para ello, creamos el script *Superrol*, cuyo único atributo es un string llamado superrol, que representa el nombre del rol escogido como superrol, con un método Get y un método Set. De esta manera, pudimos almacenarlo, consultarlo y modificarlo. Asociamos este script a la entidad de los personajes para que cada instancia pueda contar con el nuevo atributo e interactuar correctamente con el Framework.

Como inicialmente no se tenía preparada ninguna interfaz de usuario, para poder probar y ajustar la conexión de este nuevo atributo adicional con la base de datos, necesitaríamos un evento ORK que nos permitiera asignar un superrol a cada miembro del equipo mediante diálogos, sistema facilitado por el propio framework.

Para este evento, necesitábamos realizar el equivalente a un foreach de un script de los miembros del equipo, para asignar roles por defecto a todos en una misma interacción. Hicimos varias pruebas para poder averiguar cómo trasladar esta idea a la interfaz gráfica proporcionada por el framework, ya que aún no sabíamos cómo conectar los eventos del framework con scripts de códigos externos y no encontrábamos la respuesta en la documentación.

Tras varios intentos fallidos, decidimos contactar con el creador del framework mediante su foro. Nos explicó una solución para poder iterar personajes mediante un tipo de nodo llamado "Selected Data", que permite mantener en una memoria, solo para ese evento, listas de distintas entidades de la base de datos y operar con ellos mediante otros nodos. De esta manera, mezclando el tratamiento de los Selected Data con otras operaciones, pudimos crear un bucle que utilizaríamos como base para otros eventos del proyecto que presentaron el mismo problema.

Al final, llegamos a la conclusión de que, para poder realizar un bucle, debíamos realizar los siguientes pasos con los bloques:

- 1. *Select combatant*, en el caso de iterar combatientes. Para poder escoger la lista de combatientes o miembros del equipo

- 2. *Select Selected Data*, para seleccionar al primer combatiente de la lista, almacenar la información de este combatant en una variable y eliminarlo de la lista.
- 3. Diferentes eventos y funciones para trabajar con la información del combatiente de la variable
- 4. Volver al paso 1. Si da error porque no quedan combatientes, termina el bucle.

Estos mismos pasos los aplicaremos en cualquier otro evento de ORK que implique el uso de bucles.

Como se puede observar, realizar un bucle mediante el editor de eventos es mucho más confuso que programarlo en un Script. Por desgracia, para el funcionamiento del framework es necesario el uso de sus propios eventos en varios casos.

Para poder acceder al código de un script asociado a un personaje dentro de un evento de ORK, como puede ser acceder al atributo superrol desde el evento de selección, es necesario colocar un bloque de funcionalidad llamado *Call function*, que permite acceder a una función de un script asignado a un personaje u objeto sobre el que se realiza el evento.

Para la configuración inicial, desde el evento StartEvent, llamaremos a otro evento que contendrá este bucle para escoger superroles por defecto.

### 12.3.2 Posiciones de los personajes en el mapa

Para poder trabajar correctamente con los personajes y habilidades de nuestro equipo, debemos crear una instancia de ellos. Por ello, esta acción debe realizarse desde el *StartEvent*, ya que es la primera llamada que se realiza al comenzar el juego.

Aunque la mayor parte del tiempo se interaccione con el juego mediante interfaces gráficas, en la escena hay presente tras estas un mapa que hemos tratado para que sea invisible al jugador salvo durante los combates. Este mapa es necesario para que ORK funcione correctamente. Por tanto, al instanciar a un personaje, este aparece en un mapa en el que se sitúan parte de los eventos ORK.

Sin embargo, las instancias de los miembros del equipo se realizan, por defecto, en posiciones aleatorias. El problema que se nos presenta es que, si quisiéramos que en algún punto del juego, más adelante, apareciera el mapa, necesitaríamos tener a los personajes bien ubicados. Otro problema que presenta desconocer sus posiciones

exactas puede ser reubicarlos para que aparezcan en las posiciones correctas durante el combate o hacer cualquier otro cálculo sobre las posiciones.

Por estos motivos, es necesario que las posiciones de los personajes sean controladas desde el momento de su instanciación. Para ello, actuamos de manera similar al atributo *Superrol*, es decir, añadiremos un Script a los personajes donde indicaremos, en forma de vector, en qué punto del mapa deben aparecer, llamado *spawnpos* Utilizaremos la recolocación de personajes en el mapa para más de una funcionalidad, por lo que debemos hacer los eventos reutilizables.

Creamos, entonces, un evento que itera el grupo de personajes instanciados en nuestro equipo y, para cada uno, modifica el atributo *spawnpos* con la posición que debe ocupar en el mapa, calculando un pequeño offset entre los vectores para que no se superpongan.

A continuación, llamamos a otro evento que iterará a cada personaje, accederá a su atributo *spawnpos*, los leerá y los moverá a la posición deseada.

Durante la realización de esta implementación, se empezó trabajando con los personajes antes de instanciarlos, lo que hacía saltar un error y no ejecutar el evento. En este punto fue que asimilamos que sólo se puede trabajar con las instancias de los personajes, nunca directamente con las plantillas configuradas de estos en la base de datos. Por este motivo, antes de cambiar la posición de un personaje siempre es es necesario instanciarlo antes.

## 12.4 Sprint Planning

La primera funcionalidad que desarrollamos en el juego fue la primera ceremonia de Scrum: Sprint Planning. Puesto que en el Sprint Planning se toman decisiones que afectan al transcurso de la Daily Scrum, ambas partes se desarrollaron en paralelo, como indicamos en la sección de organización.

A continuación, analizaremos cómo se implementó cada sección detallada en el diseño.

### 12.4.1 Construcción del Sprint Backlog

Para el funcionamiento básico de la elección de tareas, necesitamos definir tanto un Product Backlog en el que se almacenen todas las tareas del proyecto como un Sprint

Backlog en el que se almacenen las tareas seleccionadas para realizar en el Sprint.

Como para este punto conocíamos mejor cómo conectar los eventos de ORK y los scripts externos a este, decidimos prescindir de los eventos siempre que fuera posible, sobre todo por las limitaciones del editor de eventos y la base de ORK y la consecuente incomodidad para conseguir funcionalidades complejas con este. De esta manera, las implementaciones que describimos a continuación se basan en Scripts en lenguaje C#.

En primera instancia intentamos crear, en un Script nuevo, 2 listas vacías de la clase CombatantGroup que proporciona la librería de ORK Framework: una para el Product Backlog y otra para el Sprint Backlog. Determinamos que la mejor forma de poder identificar a las tareas para cambiarlas de una lista a otra sería mediante la ID del tipo de enemigo o tarea en la base de datos del propio ORK Framework. Siendo de la clase CombatantGroup, tendrían asociadas una lista de combatientes, que sería la que iríamos modificando a lo largo del Sprint Planning.

Para poder modificar esta lista, creamos dos métodos que, dada la ID de una tarea y su experiencia, identifica la primera tarea que coincidiera con estos dos atributos de una lista, la elimina de su lista actual y la añade a la otra lista. Un método se emplearía en el caso de pasar de Product Backlog a Sprint Backlog y el segundo método sería el caso contrario. Aparte, se crearon métodos toString() para obtener los datos de ambas listas por consola.

Sin embargo, esta idea inicial no funcionó como esperábamos, ya que no se podía acceder a la base de datos del Framework ni a las instancias de las tareas de manera directa. ORK nos permitía crear tareas y modificarlas, pero no nos dejaba cambiarlas de grupo ni eliminarlas. Tampoco pudimos encontramos funciones en la documentación del framework que nos permitieran trabajar con la entidad Combatant de manera deseada.

Como trabajar directamente con tareas y con listas de tareas del propio framework era inviable pero crearlas a partir de su ID sí funcionaba correctamente, decidimos probar otra solución que no dependiera de las entidades de ORK.

En esta nueva implementación, representamos cada tareas como una lista de números enteros. Siendo el primer valor la ID del tipo de tarea en la base de datos de ORK y, el segundo valor la experiencia. De esta manera, mientras haya que modificar las tareas, siempre trabajamos con listas de enteros más simples. Solo cuando ya se tiene la selección de tareas deseada, estas sirven para crear, según su ID, combatientes de ORK Framework de la tarea deseada, a los que les asignaremos la experiencia o puntos de

historia justo tras crearlas.

De esta manera, Sprint Backlog y Product Backlog quedan representados como listas de listas de enteros. A partir de aquí, desarrollamos los métodos para trabajar con las listas y los métodos de creación y edición de tareas.

Se crearon 4 métodos para trabajar con las listas de Product Backlog y Sprint Backlog:

- 1. *AddTaskByID*: recibe la ID y la experiencia de la tarea, busca y elimina la primera tarea de Product Backlog que coincida con ID y experiencia y se añade la tarea a SprinBacklog.
- 2. *Return Task By ID*: recibe la ID y la experiencia de la tarea, busca y elimina la primera tarea de Sprint Backlog que coincida con ID y experiencia y la devuelve al ProductBacklog.
- 3. *SplitTask*: recibe la id de la tarea, su experiencia, el nombre de la lista en la que se encuentra y el número de divisiones, elimina la tarea original y devuelve x tareas con la experiencia dividida a partes iguales en la lista origen.
- 4. *JoinAllTasks*: unifica todas las tareas de un mismo tipo, sumando sus experiencias.

Posteriormente, se creó la entidad Proyecto. Asignando al script que trabaja con las listas del Product Backlog y del Sprint Backlog una entidad Proyecto, podemos cargar el Product Backlog inicial al principio del juego. Además, esto permite crear varias instancias de Proyecto con diferentes selección de tareas para el Product Backlog. Así, podremos crear nuevos niveles en nuestro juego fácilmente de cara al futuro.

Debido a que el Product Backlog se modifica durante el Sprint Planning y que las tareas irán desapareciendo de las listas a medida que las vayamos completando, decidimos incluir los atributos Product Backlog y el número total de puntos de historia inicial también en la entidad Proyecto. Esta entidad proyecto la añadimos a un GameObject vacío en la escena del juego, a la que llamamos.

### 12.4.2 Interfaz gráfica de selección de tareas

Una vez comprobado que funcionan correctamente todos los métodos, pasamos a la creación de la interfaz de usuario, por la que los jugadores podrán interactuar con

estos. Para la representación gráfica de este submenú, pensamos que sería mejor aproximarnos a la interfaz gráfica de los proyectos de Github que inventar una disposición nueva, ya que así sería más intuitiva y los usuarios de esta plataforma estarían habituados a interfaces similares. Por ello, buscamos que las dos listas de tareas funcionen como columnas de un proyecto Github y que todas las acciones se puedan aplicar mediante clicks sobre las diferentes tareas.

En primer lugar, creamos en la escena un *Canvas* que contendrá el submenú Task Selection. El Canvas es el área donde todos los elementos UI deben estar. El Canvas es un Game Object con un componente Canvas en él, y todos los elementos UI deben ser hijos de dicho Canvas. [22] Por ello, todos los elementos visibles se basarán en los elementos UI de Unity.

Insertamos en el nuevo Canvas dos elementos de tipo *Scroll*, que representaran el Product Backlog y el Sprint Backlog respectivamente. El elemento Scroll viene implementado por Unity, dejando preparada el área interior con deslizadores sobre la que se va a poder hacer scroll. En el interior de cada scroll, le añadimos un botón que representa la tarjeta de Github de una tarea. Al ser botones, Unity nos permitirá cómodamente asignarle métodos a ejecutar al ser pulsado.

Estos botones que acabamos de añadir servirán de plantillas para las tarjetas de las tareas que queramos representar en cada scroll, o lo que es lo mismo, crearemos a partir de ellas un Prefab de Unity. Los botones que creemos a partir de estas plantillas adoptarán los mismos elementos hijos que sus padres (UI y Scripts) y tendrá como referencia la posición del botón original, lo que nos facilitará situar el botón correctamente en el Canvas. Como en ambas listas se va a representar la tarea de la misma manera, modificaremos tan solo los elementos hijos de una lista y copiaremos las propiedades al botón plantilla de la otra lista.

En una primera versión, añadimos entidades de tipo *Text* para la ID de las tareas y para la experiencia de las tareas. Para poder modificarlas según la tarea a representar, creamos un script que nos permita cambiar las propiedades del botón desde cualquier otro. Añadimos un atributo de tipo Text al script para la ID y otro para la experiencia, ambos públicos, además de métodos para obtener y editar ambos valores.

Para que el script se pudiera aplicar corretamente, desde el editor de Unity, añadimos al botón plantilla un componente de tipo Script y elegimos el que acabábamos de crear. Aparecieron entonces en el editor los atributos asignables del script (ID y experiencia). Para asignarlos a los textos del botón, tan solo debemos arrastrar las entidades

de tipo Text del botón hasta los campos asignables del editor.

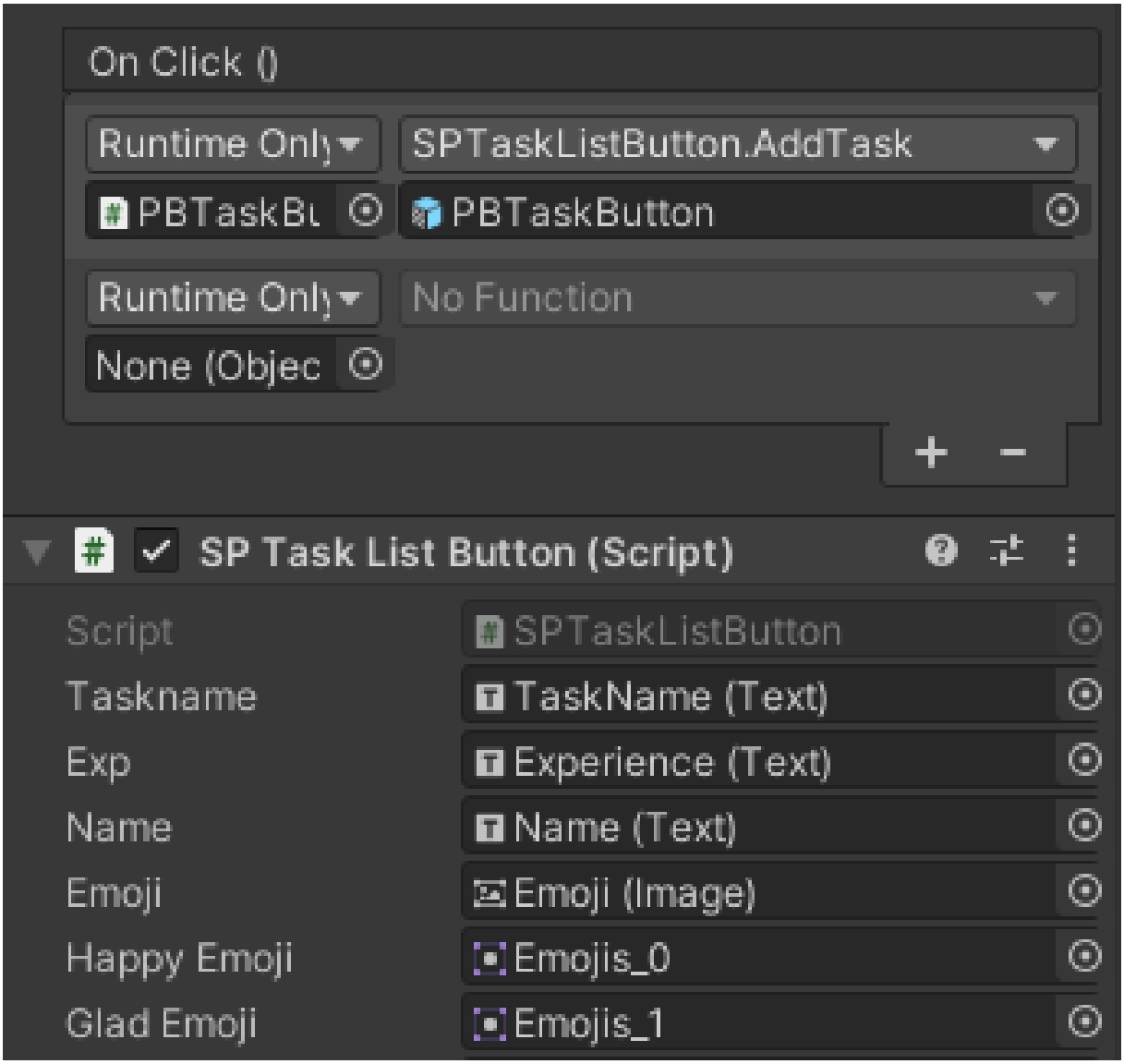



Figura 12.2: Asiganción de elementos a un script desde la interfaz de Unity

Una vez teníamos la plantilla configurada, decidimos crear un Script que hiciera de controlador de los Scrolls. Este Script contendría métodos para leer las listas de tareas y crear un botón en el Scroll por cada tarea, refrescar la lista de botones y para poder interaccionar con las lista de tareas que implementamos con anterioridad. Por ello, tendríamos que asignarle los botones plantilla de ambas listas y el script que controla las interacciones con el Product Backlog y el Sprint Backlog. Este Script lo nombramos como *SPButtonListControl*.

El método *PopulateButtonList()* obtiene ambas listas de tareas y elimina los botones de tareas presentes, si los hay, para evitar duplicados, llamando a un método auxiliar. Recorre las listas y, para cada tarea de cada lista, instancia un botón de la plantilla adecuada, obtiene los datos de la tarea y los transforma a string, modifica las propiedades del botón acorde a estos datos y lo sitúa en su posición correcta.

Para añadir los métodos de añadir, quitar, dividir y juntar tareas, creamos un método en el script asignado a cada botón que llamará a los métodos correspondientes del script controlador, transmitiéndole los datos de la tarea almacenados en ese botón. En el script controlador, para cada uno de estos scripts, se recogen y procesan los datos del botón, se transmiten al script que controlan las listas y, al final de cada uno de estos métodos, se vuelven a repoblar las listas para tener las actualizadas.

Añadimos el trigger de estos métodos en el editor de Unity, sobre las plantillas de los botones, ya que cómodamente podemos seleccionar que se ejecuten determinadas funciones de script asignados a botones mediante su componente .On click". Para ejecutar las funciones dividir y juntar, creamos dos hijos del botón que fueran a su vez botones y que, al hacer click en ellos, llamaran a los métodos correspondientes.

Durante el desarrollo de este submenú, al ser el primero, encontramos diferentes problemas. El principal y el que más prolongó la implementación del sistema de los Scrolls fueron las conexiones entre diferentes Scripts y la signación incorrecta de estos a los elementos de la UI mediante el editor de Unity.

En el primer problema, no éramos capaces de acceder al contenido de las listas Sprint Backlog y Product Backlog, aunque habíamos comprobado que se podía acceder al script mediante un método de prueba. La solución consistió en hacer static los atributos de las listas ya que, de otra manera, no podían ser accedidas por otros scripts diferentes.

El segundo problema, surgió porque asignamos el script controlador a cada scroll, por lo que Unity entendía que se trataba de Scripts diferentes de creación de listas y los botones de tareas se creaban por duplicado y no coincidían con las modificaciones. Se solucionó asignando este controlador a un nuevo GameObject vacío que usaríamos para contener todos los Scripts empleados en los distintos submenús de Sprint Planning. A este GameObject lo llamamos *Sprint Planning Scripts*.

Una vez corregidos los problemas descritos, encontramos que el Scroll no era capaz de mostrar los botones de todas las tareas cuando estas superaban un límite. Esto ocurría porque, en Unity, el área sobre el que se puede deslizar el contenido en un Scroll

tiene tamaño fijo y no cambia aunque se exceda la cantidad de contenido. La mejor forma de solucionarlo fue inicializar este área a 0 y que, cada vez que se instanciara un botón, se aumentara el área sobre la que se podía hacer scroll según el tamaño del botón.

Más adelante, cuando conseguimos una primera versión del menú completo de Sprint Planning, realizamos diferentes modificaciones para que pudiera entenderse mejor, ya que esta primera versión no era lo suficiente intuitiva y distaba de parecerse a los proyectos de Github. Añadimos nuevos atributos, tanto en la plantilla del botón como en los scripts que manipulaban los atributos del botón.

Ocultamos el ID de la tarea y los sustituimos con los nombres reales de las tareas y añadimos etiquetas para aclarar los roles de los enemigos, siguiendo el estilo de las etiquetas de github. Además, se realizó una traducción visual de la experiencia de las tareas a puntos de historia (dividiendo la experiencia real, para ajustarnos más a los valores que podrían aparecer en una estimación real e indicamos gráficamente (con emojis) la dificultad de las tareas.

Una imagen del resultado final del submenú de selección de tareas de Sprint planning se puede encontrar en .

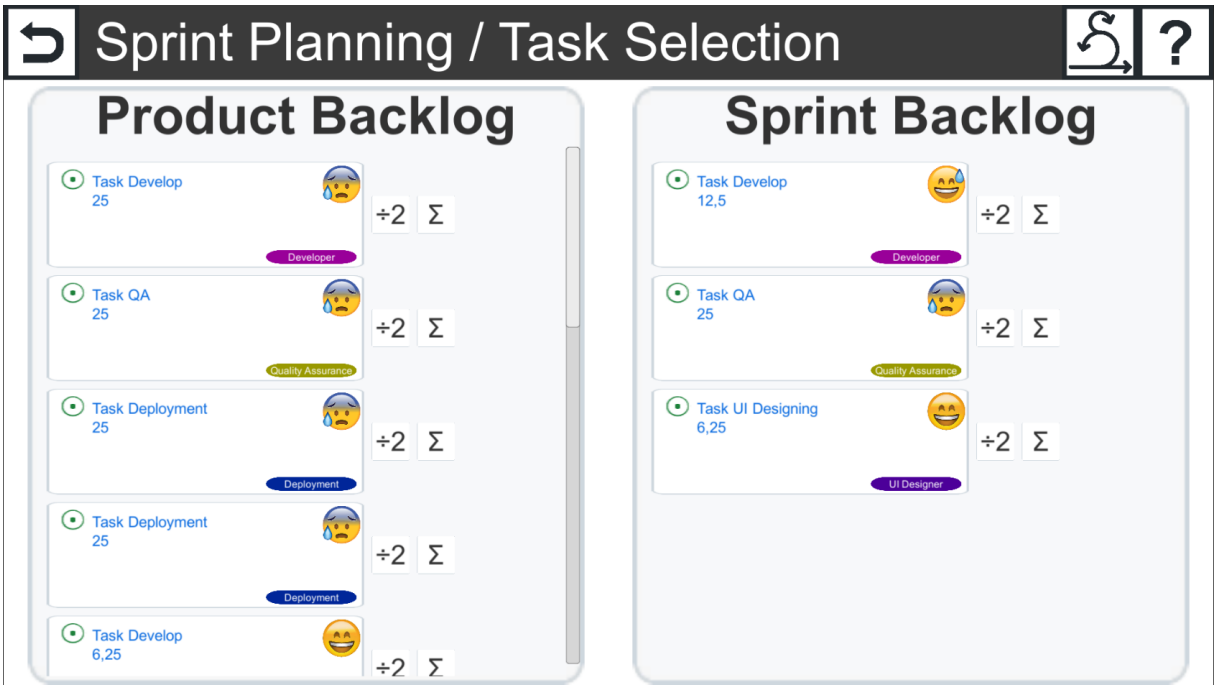



Figura 12.3: Submenú de selección de tareas del Sprint Planning

### 12.4.3 Selección de roles

Para esta funcionalidad, empleamos el atributo Superrol que añadimos mediante Scripts a la base de datos y la clase rol proporcionada por ORK Framework.

En primer lugar, hay que tener en cuenta que, cuando cambiamos de rol de un personaje durante el Sprint Planning, en realidad, estamos escogiendo un superrol, ya que será constante durante todo el Sprint. De esta manera, necesitamos implementar un método que, en función del rol escogido, establezca el superrol y la clase de un personaje.

Para satisfacer esta necesidad, implementamos el método *SetSuperrol*. Este recibe como parámetro de entrada el GameObject perteneciente al miembro al que queremos asignar un superrol y el nombre del superrol escogido. Se extrae el componente superrol del miembro del equipo y utilizamos su método setter con el nombre del superrol. A continuación, según el superrol introducido como parámetro de entrada, asignamos la misma clase. De esta manera, aplicando cualquier cambio en los roles durante el Sprint Planning, el rol y el superrol del personaje serán los mismos al inicio del Daily Scrum.

Una vez implementado el método, el siguiente paso era definir la interfaz de usuario por la que podríamos llamarlo. Como según parte de las ideas que se tenían inicialmente, el tamaño del equipo podría ser variable entre 5 y 9, necesitábamos un menú que permitiera interaccionar con X entradas. Aunque esta idea la decidiéramos dejar para trabajo futuro, debíamos implementar un submenú que permitiera fácilmente mostrar más miembros del equipo.

Este problema del número de personajes indeterminado lo habíamos solucionado con una idea similar en el submenú de selección de tareas empleando Scrolls. En este caso, cada elemento del Scroll representaría a un personaje y presentaría un botón por cada rol para poder escogerlos adecuadamente. Además, esta vez sería más fácil de implementar, ya que teníamos un ejemplo funcional de dos Scrolls y, en este caso, tan solo deberíamos hacer uno que, además, no requeriría de conexión a un tercer Script para poder completar sus datos. Sin embargo, por comodidad para dispositivos móviles, este Scroll lo haríamos horizontal. Este último detalle nos permitiría mostrar cada personaje y sus opciones en un mayor tamaño.

Creamos, entonces, un nuevo Canvas con un Scroll vacío. Siguiendo un orden similar a cuando implementamos la selección de tareas, creamos un botón plantilla con los

elementos UI necesarios que querríamos mostrar de los personajes y le asignamos un Script. En este Script, crearemos un atributo por cada elemento UI de la plantilla que queremos modificar vía código. Y, al igual que con los Scrolls del anterior submenú, creamos un Script que funcionará de controlador, al que llamamos *RoleButtonListControl*, donde también colocaremos el método SetSuperrol previamente descrito.

Empleando la misma estructura vista anteriormente, añadimos un atributo para el botón plantilla, que asignamos con el editor de Unity e implementamos un método auxiliar para borrar los botones hijos de la plantilla al refrescar la lista y un método para rellenar el Scroll, teniendo en cuenta los problemas y soluciones que surgieron en el anterior submenú, como la asignaciones del editor y el tamaño del área sobre la que podemos hacer Scroll.

Sin embargo, para el método que permite poblar la lista de personajes, no resultó tan intuitivo la obtención de los miembros del equipo. Inicialmente, tratamos de hacerlo mediante los métodos de ORK Framework, pero no pudimos encontrar en la documentación ni mediante las sugerencias una manera de obtener la instancia concreta del personaje. Sorprendentemente, encontramos otro método diferente de ORK que podía servirnos, *GetCombatant* de la clase *ComponentHelper*. Este método permitía obtener un combatiente de la clase ORK a partir del GameObject de un personaje. Así que nuestro nuevo objetivo era conseguir los GameObjects de los personajes en la escena para poder extraer su información y modificarla.

Decidimos, para este objetivo, emplear las etiquetas de Unity, que se añaden desde su editor o desde código y permiten obtener o eliminar todos los GameObjects existentes en escena con una determinada etiqueta. Para poder utilizarlas como localizador de los personajes, añadimos a los Prefabs de los personajes del equipo la etiqueta *GroupCombatant*. De esta manera, si en el código buscamos todos los GameObjects con esta etiqueta, nos devolverá los de todos los miembros del equipo. Iterando estos GameObjects y con el método previamente mencionado de la librería de ORK, podremos extraer toda la información básica.

Algunas consideraciones que hay que tener en cuenta al usar las etiquetas de Unity es que es importante reservar esta etiqueta tan solo para los miembros del equipo, ya que obtener un GameObject de otro tipo produciría un error y que no debemos usarlas en exceso, ya que cada GameObject solo puede tener una etiqueta. Además, para poder añadir o retirar etiquetas vía código, tendremos que haberlas creado previamente en el editor de Unity.

El siguiente punto a tener en cuenta era la selección de roles, la cual difería de lo visto en los Scrolls de selección de tareas. En la plantilla de cada personaje, pusimos un botón hijo por cada rol existente, es decir, 5, que tendrían el nombre del rol al que cambiarían al personaje si se pulsaba. Para poder llamar al método SetSuperrol, se añadió en el script del botón un método que recogía el nombre del rol del botón, el GameObject asignado a ese botón y llamaba al método SetSuperrol con esta información. Además, al final del método SetSuperrol, añadimos una llamada al método que actualiza los botones en la interfaz de usuario.

Posteriormente, se explicitó para cada personaje qué niveles tenían para cada rol en su botón de rol correspondiente y se implementó que nombre el rol actual del personaje se coloreara con el color asignado a ese rol, usando el mismo esquema de color que las etiquetas de los enemigos.

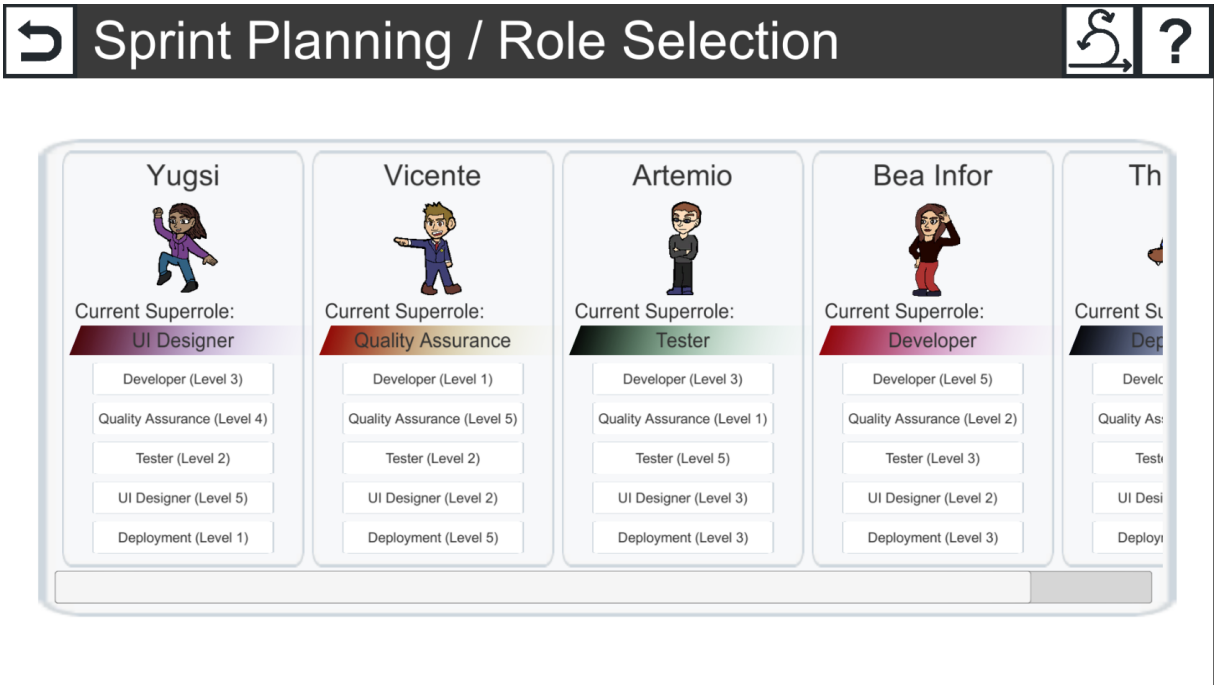



Figura 12.4: Apariencia final del submenú de selección de roles para los miembros del equipo

### 12.4.4 Elección de Scrum Master

El rol Scrum Master, en ORK Framework, es un *StatusEffect*, que ha sido previamente configurado.

Para poder transformar correctamente a un miembro del equipo en Scrum Master,

en primer lugar, creamos un método llamado *SetScrumMaster*. Este método tiene, como parámetro de entrada, el GameObject perteneciente al combatiente.

Este método, en primer lugar, como solo puede haber un Scrum Master, recorre todos los GameObject de los combatientes, obtiene de ellos su entidad Combatiente correspondiente (con la misma función empleada en el submenú de selección de roles) y elimina el estado Scrum Master.

Una vez eliminado el estado de Scrum Master de todo el grupo, se obtiene nuevamente el combatiente del GameObject del parámetro de entrada y se le añade el estado Scrum Master.

Para hacer el submenú de elección de Scrum Master coherente con el resto de submenús que hemos implementado hasta ahora, necesitábamos de nuevo un Scroll horizontal, pero más simple que el de selección de roles, ya que para aplicar Scrum Master decidimos que solo sería necesario pulsar el botón del personaje.

Una vez más, empleamos la misma estructura a nivel de código que en los otros dos submenús: un script para modificar las propiedades de un botón y llamar a otros métodos, asociado al botón plantilla y un script controlador para el comportamiento del Scroll y los botones con todas las precauciones aprendidas en submenús anteriores, al que incorporamos el método SetScrumMaster que implementamos previamente.

Para poder obtener los miembros del equipo en el método que rellena el Scroll con los personajes, hicimos uso de la misma etiqueta de Unity que empleamos para obtenerlos en el submenú de roles y el método *GetCombatant* de la clase *ComponentHelper*. Además, para poder distinguir qué personaje era el actual ScrumMaster visualmente, a la hora de poblar el Scroll con los botones de los personajes, añadimos una comprobación que buscaba el StatusEffect Scrum Master en la entidad combatiente del personaje. Si este tiene el StatusEffect adecuado, el fondo del botón de este personaje se pintará de azul.

Para incorporar la función SetScrumMaster, simplemente indicamos en el editor de Unity que, al hacer click el botón de un personaje, se ejecutara el método que llama a SetScrumMaster en el controlador con el GameObject del personaje asociado al botón.

### 12.4.5 Duración del Sprint y Menú principal

En la entidad Proyecto previamente creada, añadimos tres atributos para llevar la cuenta de los turnos del proyecto y poder evaluar los ratios del Sprint Review más

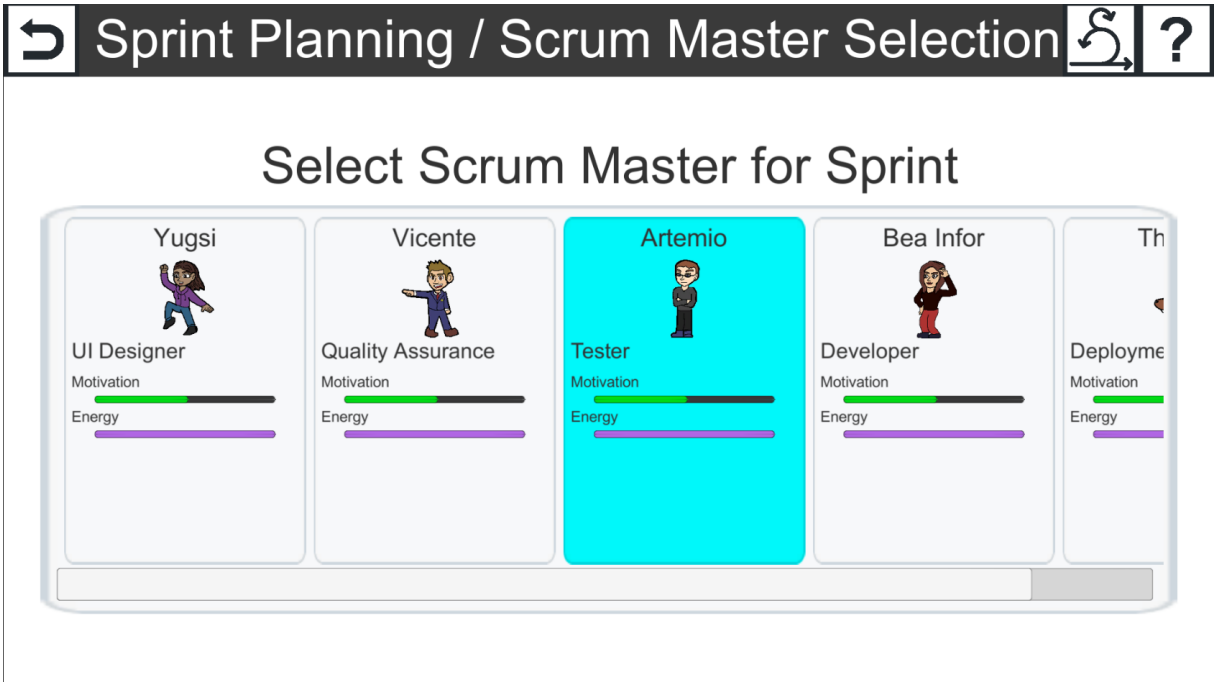



Figura 12.5: Apariencia final del submenú de selección de Scrum Master

adelante.

- El primero, llamado projectTotalTurns, es un número entero fijo que se determina al inicio del proyecto. Indica la duración máxima que puede alcanzar el proyecto, en número de turnos. Por tanto, si se supera el límite indicado por este atributo, se considera que el jugador ha perdido.
- El segundo atributo, projectRemainingTurns, es un número entero que indica cuántos turnos quedan para alcanzar el límite impuesto con el atributo projectTotalTurns. Será el número que se le mostrará al usuario durante el Sprint Planning. Además, sirve de límite superior a la hora de establecer el rango de tiempo que el jugador tiene para establecer la longitud de un nuevo Sprint.
- El tercer atributo, actualTurn, indica el turno actual en el que nos encontramos dentro del Daily Scrum. Al finalizar un Sprint, su valor será el del último turno empleado en combate. Por tanto, nos permite calcular los turnos restantes totales del proyecto.

Inicialmente, quisimos expresar el tiempo como dos posibles medidas: Turnos y semanas. Las semanas se compondrían de 7 turnos. Esto haría que el paralelismo con la

realidad fuera mayor, planteando la duración de los Sprints como semanas. Sin embargo, a la hora de realizar cálculos, solo trabajábamos con turnos. Por facilidad de entendimiento y para simplificar el procesamiento de los turnos, decidimos prescindir de la medida semana y tan solo trabajar con turnos.

Decidimos que la manera más cómoda para escoger el número de turnos sin necesidad de usar escritura sería emplear un Slider. Como realmente no necesitaba un submenú aparte, decidimos incorporarlo junto al menú principal de Sprint Planning. Este Slider podría tomar valores de 1 al número total de turnos restantes del proyecto.

Creamos un Slider y le asociamos un Script para controlar los turnos disponibles y que se parseara la entrada y actualizara el número de turnos seleccionados y lo transmitiera a la duración máxima del evento de combate. A este Script lo llamamos *TurnSelector*. Adicionalmente, creamos un segundo Script llamado *TurnsRemaining* que tan solo se usaría para mostrar correctamente en un texto el número de días restantes

En la clase TurnSelector incluímos la conexión con el evento de combate de ORK y con su entidad Sprint asociada en el método UpdateSlider. La entidad Sprint fue desarrollada en la Daily Scrum, en paralelo con Sprint Planning. Añadimos un método que parseaba de número flotante a entero, ya que el slider trabaja en valores flotantes y nuestros valores de turnos son todos enteros. También incluímos un método que actualiza los textos en pantalla para indicar la selección de turnos actual acorde a la posición del slider.

Una vez preparado el Slider, creamos el botón que llevaría a cada submenú implementado anteriormente y, a cada uno de ellos, le asignamos la función que cargaría sus respectivos Scrolls y que ocultara el menú principal y, en cada submenú, añadimos un botón de vuelta al menú principal.

## 12.5 Daily Scrum

ORK Framework maneja un sistema para los combates RPG. Desde la interfaz de ORK, se puede personalizar prácticamente todo lo que se necesite para que el combate fluya como debe. Sin embargo, no sólo habían aspectos que la interfaz no cubría que nosotros necesitábamos, sino que además el funcionamiento base de ORK no está libre de bugs o funcionalidades incompletas, por lo que gran parte de la implementación la tuvimos que hacer nosotros con código.

Dividiremos Daily Scrum en dos secciones, una que describa cómo configuramos

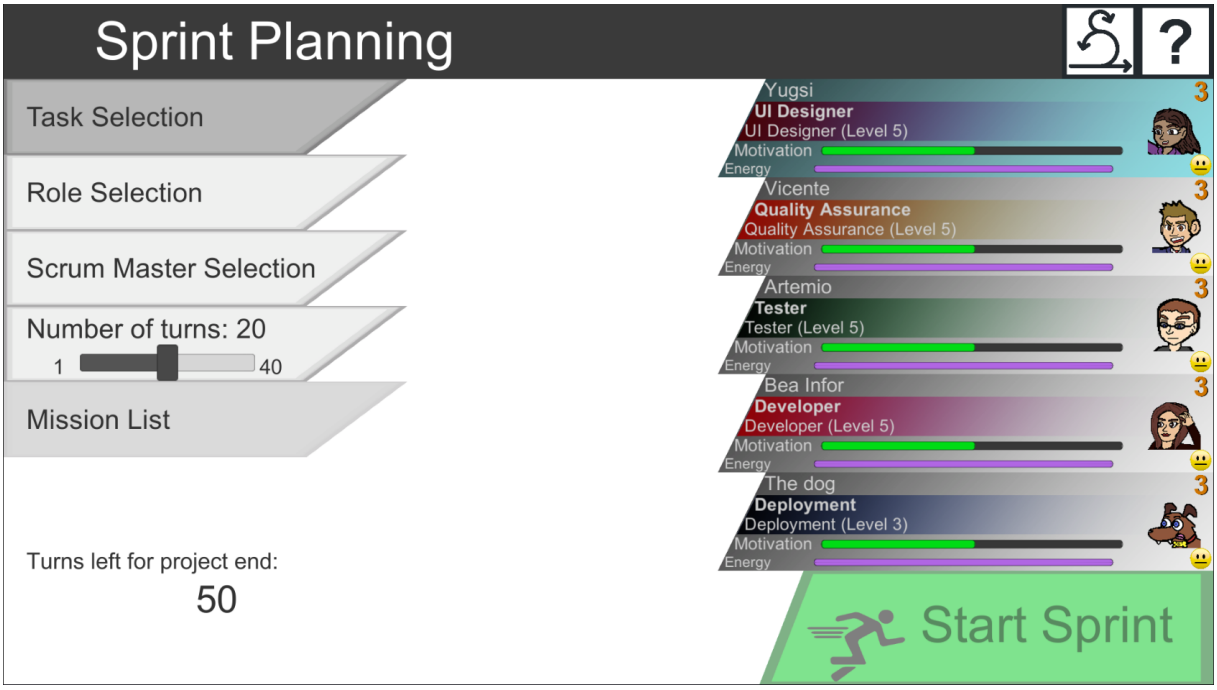



Figura 12.6: Apariencia final del menú principal junto al slider

la interfaz de ORK Framework, y otra que explica las funcionalidades implementadas complementarias a ORK.

### 12.5.1 Interfaz de ORK Framework

A continuación listamos los elementos que configuramos en ORK Framework. En las explicaciones nos centramos en información concreta de interés para distintas funcionalidades:

- **Sistema de combate:** En la interfaz de ORK debes elegir qué tipo de combate se ejecutará en un determinado encuentro. Existen 4 tipos de combate:
  - Combate por turnos: Cada personaje elige sus acciones por turnos y las ejecuta en un orden determinado.
  - Combate de tiempo activo: Similar al anterior, pero el turno de cada personaje es determinado por un medidor que se llena conforme pasa el tiempo; cuando se llena el medidor, le toca a ese personaje. El medidor de los personajes pueden seguirse llenando incluso mientras un personaje está decidiendo sus acciones.

- Combate a tiempo real: No se toman decisiones por turnos, sino a tiempo real. Generalmente cada acción tiene asignado un botón o combinación de botones.
- Combate por fases: Cada facción tiene una fase, y el combate va pasando de una fase a otra. Cada bando toma sus decisiones y las ejecuta en su fase.

Para nuestro proyecto, elegimos el combate por fases para todos los encuentros, donde el jugador puede elegir en qué orden se ejecutan las acciones de sus personajes.

Aquí tuvimos que hacer pequeños ajustes, como limitar el número de puntos de acción de cada personaje a 1 (es decir, que cada personaje pueda hacer hasta 1 acción que gaste puntos de acción), indicar que las acciones se ejecutan en orden de selección tras seleccionar acciones para todos los personajes, etc.

- **Interfaz de usuario:** La interfaz de usuario relativa a la Daily Scrum en principio iba a ser construida completamente usando las funcionalidades de ORK Framework, pero finalmente sólo mostramos el daño causado, el nombre de la habilidad utilizada y el menú de selección de personaje/habilidad.

  En otras palabras, los indicadores, el contador, el calendario Niko-Niko y el gráfico Burndown han sido diseñados por nosotros sin hacer uso de herramientas propias de ORK Framework.

- **Combatientes y roles:** Desde la interfaz de ORK Framework creamos los distintos personajes que participan en la Daily Scrum (esto incluye tanto a los miembros del equipo como a las tareas).

  Elegimos sus estadísticas base, habilidades para cada rol, crecimiento de estadísticas basadas en el nivel, experiencia necesaria para subir de nivel...

- **Comportamiento de estadísticas particulares:** Ya hemos mencionado que elegimos las estadísticas base de los personajes desde la interfaz de ORK Framework, pero desde aquí también manipulamos el comportamiento y efecto de algunas estadísticas específicas. Estas estadísticas son la motivación, productividad y número de veces golpeado.

  En la sección de fórmulas de ORK creamos una compleja fórmula que calcula el daño, y hay un punto en el que el daño recibido y realizado sigue procesos lógicos distintos en función de si el usuario es una tarea o no. Esto se hace para que:

1. Un miembro del equipo hará daño relativo a su motivación.
2. Un miembro del equipo hará daño relativo a su productividad.
3. Una tarea recibirá menos daño cuantas más veces haya sido golpeada. La fórmula no revisa si es una tarea o no el que recibe daño, pero en ningún momento aumentamos el *número de veces golpeado* de un miembro del equipo.

Además, en la fórmula de daño también se contempla la penalización de haber pasado demasiado tiempo planeando durante una Daily Scrum.

A continuación presentamos una captura de la fórmula diseñada en la interfaz de usuario de ORK Framework:

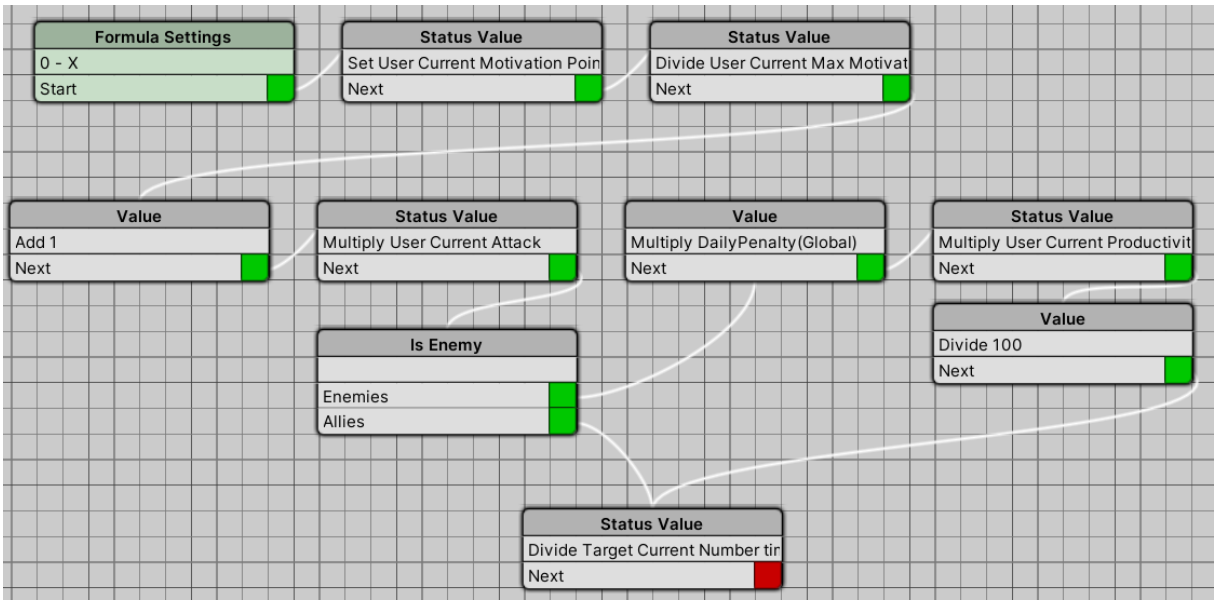



Figura 12.7: Fórmula de daño empleada en nuestro programa

- **Habilidades:** Usando la interfaz de ORK determinamos los atributos de las habilidades.

  Para ciertas funcionalidades, como el aumento de motivación al vencer a un enemigo o el aumentar la estadística *número de veces golpeado*, a las habilidades les asociamos un evento que se ejecuta al realizarse la habilidad.

- **Estados:** Los estados que definimos son los 5 stages, y en función de qué estado tiene el personaje, su estadística de ataque aumentará consecuentemente.

  Aún así, hemos creado 3 estados más. Explicamos qué son a continuación:

  1. *Recuperación de PE:* Este estado hace que los personajes con él recuperen cierta cantidad de energía cada turno. Se lo otorgamos a todos los miembros del equipo.
  2. *Counter:* Este estado se le aplica a todas las tareas. Aumenta la probabilidad de contraataque a 100 % al recibir daño.
  3. *Scrum Master:* En el diseño especificamos que el rol de Scrum Master se determinaría en base a un atributo booleano que sería propio de cada miembro del equipo, y aquel que lo tuviera en *true* sería el Scrum Master. Los estados en juegos RPG también pueden verse como etiquetas que se le ponen a los combatientes, no necesariamente deben tener efecto. Hemos seguido esta lógica y hemos hecho que el miembro designado como Scrum Master sea el portador de este estado, y nadie más pueda tenerlo.

- **Eventos:** Creamos múltiples eventos para el proceso de combate. Los más destacables son los de inicio de combate, los que son ejecutados al realizar ciertas acciones (como mencionamos al explicar las habilidades) y aquellos que se ejecutan periódicamente durante la Daily Scrum. Algunos destacables son:

  - **BattleUpdate:** Probablemente nuestro evento más importante para la Daily Scrum. Este evento se ejecuta al comenzar el combate y al principio de cada turno.

    Sus funcionalidades son:

    1. Reiniciar una variable que determina si el usuario ha seleccionado alguna habilidad de Scrum Master.
    2. Descongelar el contador de la Daily Scrum.
    3. Reiniciar el contador de la Daily Scrum.
    4. Añadir entradas al gráfico Burndown.

5. Actualizar el calendario Niko-Niko con los nuevos estados Niko-Niko de este turno
6. Comprobar quién es Scrum Master y otorga las habilidades correspondientes.
7. Actualizar los Stages y ánimo de los distintos miembros del equipo.

A continuación dejamos una captura de pantalla de su proceso de ejecución:

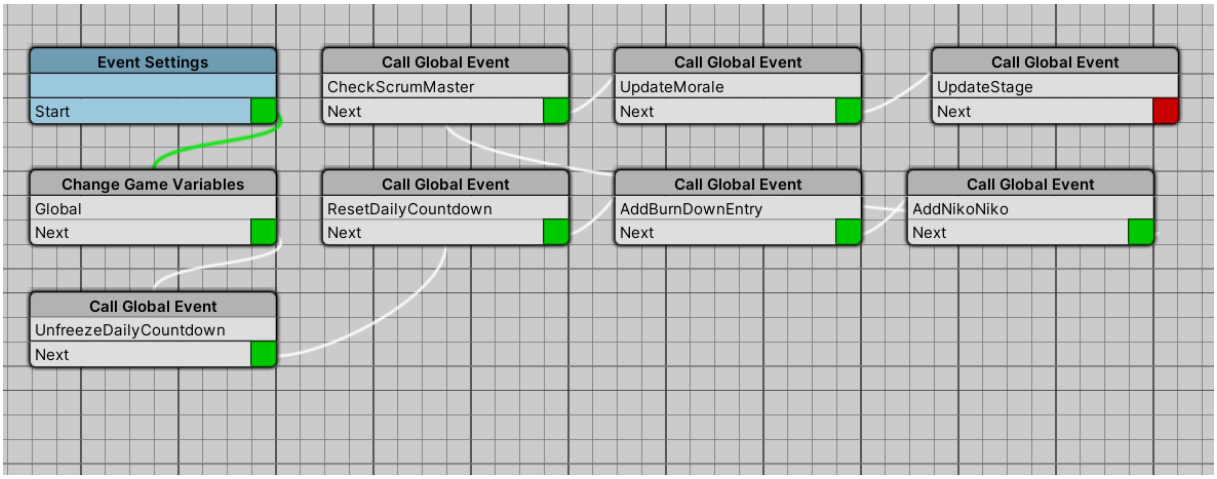



Figura 12.8: Evento Battle Update

- **EnemyBattleUpdate:** Se ejecuta al principio de cada turno. Reinicia el número de golpes que el enemigo ha recibido.
- **TurnEndEvent:** Se ejecuta al final de cada turno. Actualiza el estado Niko-Niko de cada miembro y disminuye la motivación de los miembros del equipo si no hicieron ninguna acción.

Con esto hemos cubierto los principales usos que le dimos a la interfaz de ORK Framework para el combate.

### 12.5.2 Código complementario a ORK Framework

Esta sección la subdiviremos en dos:

1. Funcionalidades que implementamos de forma ajena a las funcionalidades de ORK (a excepción de la interacción con la base de datos).
2. Ajustes que tuvimos que hacer para superar las dificultades que presentó el framework.

**Funcionalidades ajenas a ORK Framework**

1. **Calendario Niko-Niko:** Esta explicación la dividiremos en tres partes: los estados Niko-Niko, el calendario Niko-Niko y cómo se creó la interfaz de usuario.

   Los estados Niko-Niko son determinados al final de cada turno a través del evento TurnEndEvent. Se expresan en forma de una estadística que tienen los personajes, y son determinados usando un factor aleatorio, con probabilidad de mejores estados si la motivación es alta. A continuación se muestra la función que actualiza el estado Niko-Niko de un miembro del equipo:

```
public void AddNikoNiko()
{
    Combatant combatant = ComponentHelper.GetCombatant(gameObject); //Get Combatant

    int motivation = combatant.Status[1].GetValue();        //Get motivation
    int maxMotivation = combatant.Status[0].GetValue();     //Get max motivation (should be 100)

    float motivationPercentage = (float)motivation / maxMotivation; //Get % of remaining motivation

    int r = Mathf.RoundToInt(Random.Range(0, motivationPercentage));    //Create a random number based on the remaining motivation

    int nikoValue = Random.Range(-1, 2) + r;    //Determine niko-niko state based on a random chance + the random number calculated earlier

    combatant.Status[13].Set(nikoValue, nikoValue); //Update niko-niko stat

    int newMotivation = motivation + 10 * nikoValue;        //Increase motivation if the character is happy, but reduce if the character is sad
    combatant.Status[1].Set(newMotivation, newMotivation);

    GetComponent<NikoNikoHistorial>().historialNikoNiko.Add(nikoValue); //Add niko-niko state to history for the calendar
    nikoNikoCalendar.IncreaseNumber();
}
```

Figura 12.9: Código que determina el estado Niko-Niko resultante

Al añadirse un estado Niko-Niko, se añade a un historial, que es único para cada personaje. Luego otra clase que representa el calendario crea una entrada por cada estado que haya en el historial de cada personaje. Expresemos esto como una secuencia:

*a*) El jugador genera el calendario pulsando el botón.

*b*) Por cada miembro del equipo, el calendario crea una fila.

*c*) Se revisa el historial de cada miembro y crea una entrada por cada elemento del historial, ordenados por orden de entrada.

La interfaz de usuario de Niko-Niko fue creada usando elementos propios de Unity, como son los objetos Canvas, Image, etc. y con código creamos una coherencia entre estos objetos, y los vamos generando y posicionando como queremos que se presente al usuario.

2. **Gráfico Burndown:** Al principio de cada turno, en el evento BattleUpdate, se crea una nueva entrada Burndown y se añade a una lista.

   Una entrada Burndown es una clase que creamos nosotros, pero a efectos prácticos es un vector bidimensional cuyo valor X es el turno en el que se genera y el valor Y es la cantidad de Puntos de Historia que quedan por completar.

   El gráfico Burndown es generado al pulsar el botón oportuno. Se recorre la lista de entradas y se generan de forma gráfica.

   Fue especialmente difícil hacer que la gráfica se mostrase bien en distintas resoluciones porque no podíamos hacer que las entradas modificasen su posición correctamente en función del tamaño de la pantalla. Nuestra solución fue crear

objetos invisibles que escalasen correctamente con el tamaño de la pantalla para que actuasen como referencia para las entradas de la tabla, de tal forma que cada vez que el programa detecta que se ha cambiado la resolución de la pantalla, se actualiza la posición de las entradas basándose en la posición de estos objetos invisibles que creamos.

3. **Contador:** El contador debe avanzar mientras el jugador está decidiendo, debe pararse mientras los personajes actúan y debe reiniciarse al principio de cada turno. Cada vez que el contador llega a 0, se reinicia, pero aumenta en 1 la penalización que se le hará al daño que produzcan los miembros del equipo.

   Como ya hemos mencionado, el evento BattleUpdate reinicia el contador y lo descongela. ¿Pero qué lo congela?

   Todas las habilidades tienen la misma animación. La animación se define a través de eventos. La animación de las habilidades visualmente no hace nada, pero el evento que controla esa animación detiene el contador. Es decir, el contador se detiene al ejecutarse una habilidad. Está bien, pero... ¿por qué hacerlo ahí?

   En ORK, los eventos propios del combate pueden ejecutarse en estos 4 momentos:

   - Inicializar un combatiente por primera vez
   - Generar un combatiente (independientemente de que ya se haya generado antes)
   - Comienzo del turno
   - Final del turno

   Para lo que queremos, ignoramos los dos primeros casos porque no se inicializa ni genera ningún combatiente al iniciar la ejecución de acciones. Un turno comienza al principio de la selección de acciones y un turno finaliza al terminar de ejecutar las acciones, es decir, los eventos asociados a estos momentos son llamados antes de elegir las acciones y tras ejecutarse todas las acciones. Lo que queremos es detener el contador al terminar de seleccionar las acciones, y este momento no coincide con ninguno de los 4 que ORK ofrece. Por este motivo, se nos ocurrió que fuese la primera habilidad que se ejecuta en el turno lo que detenga al contador.

   Somos conscientes de que la solución que hemos propuesto no es la más limpia; entre otras cosas, la función de parar el contador ocurre una vez por cada acción, cuando sólo debería ocurrir una vez por turno. Aún así, no encontramos mejor

forma de implementarlo. Intentamos hacer que el contador pudiera determinar por sí mismo si debería estar activo o no, haciendo que comprobase la fase del combate y que, dependiendo de si el jugador estuviera seleccionando acciones o si estas estuvieran ejecutándose, el contador estaría o no en marcha, pero no logramos hacerlo, y el desarrollador del framework no pudo ayudarnos a determinar de forma programática la fase en la que estaba el combate.

En definitiva, las habilidades paran el contador porque no encontramos otra forma de hacer que el contador se detuviera tras elegir las acciones.

El contador es completamente ajeno a ORK Framework. Se actualiza en cada frame, determina si el tiempo ha llegado a 0 y, si ha llegado a 0, aumenta en 1 su atributo timeOut y actualiza una variable global de ORK que es la que luego se llama en la fórmula de daño.

Existe una complejidad extra del contador. Tiene dos métodos para congelarse, uno que es el que se llama al finalizar la selección de acciones, y otro que se llama al abrirse la ventana de ayuda. Estos dos métodos se distinguen en que el segundo sólo parará el contador si es la primera vez que se llama. Esto se hace para que el jugador no pueda pausar el contador para poder planear su Daily.

La interfaz de usuario fue creada, otra vez, utilizando exclusivamente herramientas propias de Unity acompañadas de un código que ayuda a gestionar la interacción de estos componentes.

4. **Indicadores:** Existen 3 tipos de indicadores: Indicador de completitud de enemigos, indicador de avance de Sprint e indicadores de los miembros del equipo.

   Todos son creados sin hacer uso de ORK Framework, sólo para acceder a algunos datos como la motivación y energía de los personajes o el turno actual del Sprint (el turno de final de Sprint no se almacena en la base de datos de ORK, sino en una clase que nosotros creamos llamada Sprint).

**Funcionalidades complementarias a ORK**

Existen algunas mecánicas que tuvimos que implementar uniendo ORK con nuestro código.

Hasta cierto punto, el contador, el calendario Niko-Niko y el gráfico Burndown figuran en esta sección ya que para congelarse (en el caso del contador) y actualizarse (en el caso del calendario Niko-Niko y el gráfico Burndown), se hacen uso de eventos de ORK.

Aún así, existen unas funcionalidades más que mencionaremos aquí.

- **Limitar habilidades de Scrum Master a 1 por turno:** ORK Framework da soporte para que distintas acciones consuman más o menos Puntos de Acción; cuando los Puntos de Acción de un personaje llegan a 0, no podrá realizar más acciones ese turno. Todas las acciones que dañen a enemigos y no hacer nada reduce los Puntos de Acción a 0, mientras que cambiar de rol y las habilidades de Scrum Master no reducen los Puntos de Acción. Necesitamos que el Scrum Master pueda hacer una acción de Scrum Master y más acciones en un mismo turno, pero sólo podrá usar una acción de Scrum Master.

  Pensamos en hacer que las acciones de Scrum Master gastasen algo más de la mitad de los Puntos de Acción, para que luego el personaje no pudiera hacer otra acción de Scrum Master por no tener suficientes Puntos de Acción. Sin embargo, esto tiene dos inconvenientes:

  1. Si no tiene Puntos de Acción suficientes para hacer una acción de Scrum Master, tampoco tendrá suficientes para hacer otra acción (que no sea cambiar de rol).
  2. Se puede hacer que un personaje pueda gastar más Puntos de Acción de los que tiene, y que no pueda seguir seleccoinando acciones si sus Puntos de Acción son menor o igual que 0, de esa forma, podría seguir usando sus acciones aunque no tuviera todos sus Puntos de Acción. El inconveniente es que esto también se aplicaría a las acciones de Scrum Master, por lo que no soluciona nada.

  Nuestra solución fue la siguiente:

  Cada botón de la interfaz de usuario generada por ORK framework (es decir, los botones de selección de personaje/habilidad/objetivo) es un objeto individual, que creamos a partir de prefabs de Unity. Los prefabs son elementos de juego replicables en Unity. Podemos hacer que un botón particular use un prefab distinto a los demás botones. Hicimos que Scrum Master usase un botón propio, y le implementamos una lógica propia a ese botón, haciendo que se desactive si el jugador ya ha seleccionado una acción de Scrum Master este turno.

  Hicimos que se ejecutase un evento determinado al confirmar la ejecución de una habilidad de Scrum Master. En este evento, se almacena información en una variable, que determina si se ha seleccionado o no una habilidad de Scrum Master.

Si esta variable está en *true*, el botón se desactiva, impidiendo que el jugador pueda hacer acciones de Scrum Master. Esta variable se pone automáticamente a *false* durante el evento de BattleUpdate.

- **Coste de las habilidades:** Las habilidades tienen su coste determinado en la interfaz de ORK Framework, y funciona correctamente. Además, en ORK también puedes determinar cuándo se consume el recurso que corresponde al coste de dicha habilidad. Lo lógico es que la energía se consuma tras ejecutar la habilidad.

  Sin embargo, ORK Framework tiene un bug. Si la energía se consume tras la ejecución de la habilidad, la habilidad se ejecutará independientemente de si el personaje tenía suficiente energía a la hora de ejecutar la acción, y el jugador podrá seleccionar acciones siempre y cuando consuman menos energía de la que le queda ahora mismo al combatiente.

  Lo explicamos con unos ejemplos. Recordemos que no consumir los puntos de acción permite que ese mismo personaje haga más acciones en el mismo turno. El primer ejemplo es el caso que funciona, y el segundo es el que explica el problema:

  **Caso 1:**

  1. Nuestro personaje tiene 20 de energía.
  2. No puede seleccionar una habilidad que cuesta 30 de energía.
  3. Selecciona una habilidad que cuesta 15 de energía. Esta habilidad consume todos los puntos de acción.
  4. El personaje ejecuta la acción.

  **Caso 2:**

  1. Nuestro personaje tiene 20 de energía.
  2. No puede seleccionar una habilidad que cuesta 30 de energía.
  3. Selecciona una habilidad que cuesta 15 de energía. Esta habilidad no consume puntos de acción.
  4. Selecciona la misma habilidad de nuevo.
  5. Selecciona la misma habilidad de nuevo.
  6. Selecciona la misma habilidad de nuevo.
  7. Selecciona una habilidad con coste 20 y que consume los puntos de acción.

8. El personaje ejecuta todas las acciones. Su energía varía en este orden:
   *a*) Empieza en 20.
   *b*) Realiza la primera acción: se reduce a 5.
   *c*) Realiza la segunda acción: se reduce a 0.
   *d*) Realiza la tercera acción: se reduce a 0.
   *e*) Realiza la cuarta acción: se reduce a 0.
   *f*) Realiza la quinta acción: se reduce a 0.

Como podemos ver, ORK Framework funciona sólo si se asume que el personaje va a realizar sólo una acción por turno, ya que evalúa la energía que le queda al personaje respecto al coste de la habilidad sólo en el momento de seleccionar dicha habilidad.

Nuestra solución fue la siguiente:

Observamos que si se hacía que la energía se consumiera al seleccionar la habilidad, sí que se revisaba si el personaje tenía suficiente energía para ejecutarla durante la ejecución de acciones. Esto no tiene ningún sentido porque podemos llegar a un caso como este:

1. Nuestro personaje tiene 20 de energía.
2. No puede seleccionar una habilidad que cuesta 30 de energía.
3. Selecciona una habilidad que cuesta 15 de energía. Esta habilidad consume todos los puntos de acción.
4. La energía del personaje baja a 5.
5. El personaje no puede ejecutar la acción porque no tiene 20 de energía, por lo que no hace nada.

Aún así, pudimos utilizar esto para nuestro favor.

Hicimos que la selección y ejecución de acciones siguiera el siguiente proceso:

1. Seleccionamos una habilidad. La energía se reduce en el momento.
2. Se ejecuta un evento que aumenta la energía del usuario en la misma cantidad que se le redujo.
3. Se ejecuta la acción. En este proceso se verifica que el personaje tenga energía suficiente para ejecutar la acción.

4. A la habilidad le añadimos un nuevo efecto, que es que le reduzca la energía al usuario equivalente al coste de la habilidad.

De esta forma, el mismo ejemplo que hemos puesto antes ocurriría así:

1. Nuestro personaje tiene 20 de energía.
2. No puede seleccionar una habilidad que cuesta 30 de energía.
3. Selecciona una habilidad que cuesta 15 de energía. Esta habilidad consume todos los puntos de acción.
4. La energía del personaje baja a 5.
5. Se ejecuta el evento que aumenta la energía del personaje.
6. La energía del personaje sube a 20.
7. El personaje ejecuta la acción (puede hacerlo porque tiene más de 15 de energía).
8. La energía del personaje se reduce a 5.

A efectos prácticos, conseguimos lo que queremos, que es que el personaje pueda ejecutar sus acciones gastando la energía correspondiente.

Exponemos cómo quedaría el ejemplo que expresaba el bug tras aplicar nuestra solución:

1. Nuestro personaje tiene 20 de energía.
2. No puede seleccionar una habilidad que cuesta 30 de energía.
3. Selecciona una habilidad que cuesta 15 de energía. Esta habilidad no consume puntos de acción.
4. Su energía baja a 5 y tras ejecutar el evento, sube a 20.
5. Selecciona la misma habilidad de nuevo.
6. Su energía baja a 5 y tras ejecutar el evento, sube a 20.
7. Selecciona la misma habilidad de nuevo.
8. Su energía baja a 5 y tras ejecutar el evento, sube a 20.
9. Selecciona la misma habilidad de nuevo.
10. Su energía baja a 5 y tras ejecutar el evento, sube a 20.
11. Selecciona una habilidad con coste 20 y que consume los puntos de acción.

12. Su energía baja a 0 y tras ejecutar el evento, sube a 20.
13. El personaje ejecuta las 5 acciones así:
    *a*) Tiene 20 de energía, la acción cuesta 15, por lo que ejecuta la acción.
    *b*) Su energía se reduce a 5.
    *c*) Tiene 5 de energía, la acción cuesta 15, por lo que no puede ejecutar la acción.
    *d*) Tiene 5 de energía, la acción cuesta 15, por lo que no puede ejecutar la acción.
    *e*) Tiene 5 de energía, la acción cuesta 15, por lo que no puede ejecutar la acción.
    *f*) Tiene 5 de energía, la acción cuesta 20, por lo que no puede ejecutar la acción.

Como podemos observar, ahora el sistema de coste de energía funciona como debería.

Con esto, hemos cubierto todas las funcionalidades destacables que hemos implementado acerca de la escena de Daily Scrum.


Figura 12.10: Apariencia final de la vista de Daily Scrum

## 12.6 Sprint Review

### 12.6.1 Conexión con ORK

Para la implementación de Sprint Review, en primer lugar, creamos el script principal, SprintReviewUIInfo, que será llamado mediante un evento de ORK Framework que se activa al terminar el combate. Este script lo añadimos al GameObject "Proyecto", que añadimos en la escena previamente.

El primer paso, antes de definir el script, fue realizar la conexión entre ORK y el script. Por una parte, en el script del Sprint Review añadimos un método temporal que tan solo imprimiría por la consola de Unity un mensaje, para comprobar que la conexión funcionaba. En el editor de ORK Framework, creamos un evento de tipo "Battle End Event", que ORK permite asociar a un evento de combate para ser llamado al finalizarse. A este evento tan solo le pusimos un bloque de funcionalidad del tipo Çall function", que ya hemos visto anteriormente, para llamar a uno de nuestros scripts originales.

Es aquí cuando apareció nuestro primer problema de conexión entre ORK y el evento Sprint Review. Para hacer llamadas a un script externo, este debe estar asociado a los GameObjects de los personajes del equipo. Sin embargo, nosotros queremos una única instancia del script Sprint Review, de manera que había que buscar otra alternativa. Podíamos hacer un script intermediario que tan solo redireccionara del evento al script deseado, pero sería añadir un script con sus respectivas conexiones que se usaría poco.

Otra alternativa pa ra solucionar este problema era crear esta llamada intermediaria, que vimos beneficiosa de la anterior idea, pero en un script ya existente y anexionado a los personajes del equipo. Por ello, escogimos crear este método en el script Superrol, desarrollado previamente. Este método se llama SprintEnding y su única función es localizar el GameObject en escena llamado "Proyecto"para ejecutar su script de Sprint Review.

Uno de los primeros problemas que nos encontramos, al probar que ORK llamara correctamente desde su evento al script nuevo mediante mensajes de consola simples, fue que el evento se llamaba una vez por combatiente existente en nuestro equipo, en este caso, se ejecutaba 5 veces. Para controlar que solo se llamara una vez, añadimos un atributo de tipo booleano, que indica si se ha llamado al evento tras finalizar el combate, inicializándolo con valor falso, se manera que solo se ejecutará el método que

hay en el script de Sprint Review si el booleano es falso y cambiándolo a verdadero tras la primera llamada para evitar repeticiones. Una vez solucionado este problema, el fin del combate y el inicio de la Sprint Review estaban listos.

### 12.6.2 Ratios

Para poder evaluar correctamente si el Sprint ha sido bueno o malo, de acuerdo al diseño, debemos calcular el ratio de porcentaje de puntos de historia derrotados contra el porcentaje de tiempo usado. Para ello, debemos tener en cuenta varios valores, que también mostraremos al jugador en la interfaz de usuario:

- 1. Turnos totales del proyecto
- 2. Cantidad de turnos restantes
- 3. Turnos empleados durante el Sprint
- 4. Cantidad de puntos de historia pendientes
- 5. Puntos de historia totales del proyecto
- 6. Puntos de historia finalizados durante el Sprint

Los atributos de tiempo ya los teníamos implementados en la entidad Proyecto. Para poder tener los nuevos atributos de experiencia de manera ordenada, los añadimos a la misma entidad.

Tal como teníamos implementado el Product Backlog y el Sprint Backlog, no se modificaban con el transcurso del combate, por lo que al final de la Daily Scrum no teníamos las listas actualizadas según las tareas que se habían logrado terminar y era difícil determinar los parámetros necesarios para los ratios. Necesitábamos que los backlogs se fueran modificando a medida que se fueran completando tareas en el Daily Scrum, pero era difícil establecer un trigger para llamar al evento en el momento adecuado.

Al igual que existían eventos de ORK que se ejecutaban al iniciar o terminar un turno o al terminar una batalla, se nos ocurrió que podría existir alguna manera de activar un evento cada vez que se derrotara a un enemigo, en el mejor de los casos via Script externo. Preguntamos entonces en el foro de ORK Framework y, por suerte, un usuario nos enseñó una manera simple de hacer justo lo que estábamos buscando,

registrando en ORK un evento a ejecutar cada vez que se eliminara un enemigo y pasando por parámetro al evento al enemigo caído. En este método, eliminaríamos del Sprint Backlog la tarea derrotada, añadiríamos su experiencia al atributo de experiencia derrotada en el sprint y la restaríamos de experiencia pendiente. Así, al final del combate devolveríamos las tareas que no se hubieran derrotado al Product Backlog y seguirían presentes de cara al siguiente Sprint y a los cálculos de los ratios.

Sin embargo, tal como representábamos las tareas, había ciertos problemas para distinguir tareas idénticas en el Sprint Backlog y poder eliminarlas adecuadamente, ya que representamos una tarea como una id y su experiencia. La experiencia de los enemigos resultó no ser guardada por ORK de manera adecuada y siempre valía 0. Más adelante, descubriríamos que tampoco escalaría el nivel de los enemigos correctamente con la experiencia con las que los creábamos. Esto hacía que solo tuviéramos como criterio de búsqueda la id, que se repetiría en todas las tareas del mismo tipo.

Como solución, decidimos añadir un tercer valor a la lista de enteros que representaba a las tareas: la id de la tarea dentro del grupo de combate, que se añadiría al crear el grupo de combatientes a partir de las tareas seleccionadas. De esta manera, si la tarea eliminada del combate coincidía con una tarea del Sprint Backlog en ID de tipo de tarea e ID dentro del grupo de combatientes, esta tarea del Sprint Backlog sería la que deberíamos eliminar.

Una vez solucionado, ya podíamos calcular los 6 valores necesarios para la Sprint Review. En un método, enviábamos estos 6 valores y los ratios calculados a partir de estos al script de Sprint Review, que evaluaría los ratios y prepararía todos los datos para mostrarlos por la interfaz de usuario por medio de paneles de texto.

### 12.6.3 Misiones

Debido a que, en el diseño, explicamos que en la Sprint Review se deberían mostrar tanto el resultado del Sprint en forma de ratios como el resultado de las misiones (si se han cumplido o no), la implementación de Sprint Review no termina de estar completa.

Para implementar las misiones, deberíamos decidir primero cómo las procesaríamos. Teníamos, en este punto, dos opciones para implementar los dos tipos de misiones:

- Crear una entidad padre Misión y que cada tipo de misión fuera una entidad hija, heredando atributos comunes de la clase padre y con sus atributos específicos

en la clase hija. La ventaja principal era que permitiría ahorrar líneas de código y tener métodos compatibles para ambos tipos de misión. Sin embargo, la desventaja principal que presentaba era que, aunque procesáramos la clase padre, desconocíamos si existía una manera de, conocida la clase padre de un objeto, ser capaz de sacar su clase hija, lo cuál era fundamental para evaluar todas las misiones correctamente.

- Hacer dos entidades independientes para los dos tipos de misiones, aunque compartan varios atributos. La ventaja que presentaba era que sería mucho más fácil distinguir los tipos de la misión, pero a costa de repetir muchas líneas de código para cada tipo de misión.

Aunque idealmente era mejor la primera, la incertidumbre y la dificultad para solucionar su problema principal provocó que en un primer momento tratáramos de emplear la segunda opción. Sin embargo, el código quedaba bastante confuso para su lectura y sería muy difícil ampliar nuevos tipos de misión en el futuro. En cualquier caso, ambas soluciones tendrían ciertas partes de su estructura similares, por lo que tampoco sería muy difícil pasar de una solución a otra en poco tiempo.

Creamos las entidades para la misión de tiempo y la misión de experiencia. La única diferencia entre los dos tipos de misión era el atributo que indicaba la medida de la misión. En un caso, expresaba número de turnos y, en el segundo, cantidad de experiencia.

A continuación, implementamos un Script llamado *MissionsEvaluations* encargado de almacenar las listas de misiones que se estaban cumpliendo en el momento ,sus métodos de evaluación, uno por cada tipo de misión y un método que invoca a la evaluación de todos los tipos de misiones. Aparte, se añadió un pequeño método para que se cargaran al inicio de la partida las diferentes misiones que se le sugerirían al jugador en la Sprint Retrospective.

Al Script encargado de preparar la interfaz de usuario de la Sprint Review, le asociamos el script *MissionsEvaluations* para poder acceder a las misiones y a sus resultados y poder mostrarlos por pantalla en la interfaz de usuario. La Sprint Review llamaría al método que realiza todas las evaluaciones de las tareas, que cambiarían un atributo booleano común en ambos tipos de misiones que indica si se ha cumplido la misión o no y recompensaría o penalizaría la motivación de los miembros del equipo según un atributo que indicaba el porcentaje. Entonces, se consultarían las misiones para extraer sus atributos y mostrarlos por pantalla.

Tras haber preparado la interfaz de usuario para los ratios y sus variables, habíamos reservado espacio para mostrar 6 misiones en forma de "tarjetasçon tamaño razonable. Para poder enseñarlos por pantalla, creamos dos bucles casi idénticos, uno para la misión de tiempo y otro para la misión de experiencia. Para cada misión se creaba un panel en el que se escribían los datos según si se había superado la misión, se coloreaba según su tipo y se colocaba en una de las posiciones de la pantalla reservadas para la misión. Para la posición y la uniformidad de estas tarjetas de misión, volvimos a usar el método de plantillas utilizado en anteriores interfaces de usuarios.

Cuando se implementó toda esta estructura, quedó un código demasiado confuso y difícil de mantener. Por suerte, Matthew encontró en uno de sus antiguos proyectos de Unity un método de Unity que permitía distinguir clase hija de un elemento en una colección de elementos de una clase padre. Por ello, pudimos modificar la estructura a la idea inicial, lo que nos permitió eliminar muchas líneas de código duplicado y agilizar los métodos.

Creamos una clase padre de tipo misión, con todos los atributos comunes de las misiones y una clase hija por cada tipo de misión con sus atributos específicos de evaluación. En cada método que requería distinguir el tipo de misión concreto, como en los métodos de evaluación, usábamos la funcionalidad de Unity *typeof*, añadiéndole como argumento la clase de una entidad hija y la compárabamos con el resultado de aplicarle a la misión la funcionalidad básica de Unity *GetType()*.

Como posteriormente determinaríamos que sólo se podría escoger una misión de cada tipo y solo teníamos dos misiones, preparamos las funciones de interfaz de usuario para que tan solo se mostraran una misión de tiempo y una misión de experiencia.

## 12.7 Sprint Retrospective

### 12.7.1 Selección de misiones

Para la última ceremonia de Scrum, queríamos representar una interfaz en la que se pudiera escoger las misiones que se deberían cumplir en el siguiente Sprint. En esta interfaz gráfica, de nuevo, empleamos un Scroll con un botón por misión sugerida para poder ofrecer al jugador un número indefinido de opciones. Al acceder a la Sprint Retrospective, se debería vaciar la lista de misiones del Sprint anterior y, al hacer click sobre una misión sugerida, esta se debería añadir a la lista de misiones a evaluar en la siguiente Sprint Retrospective.

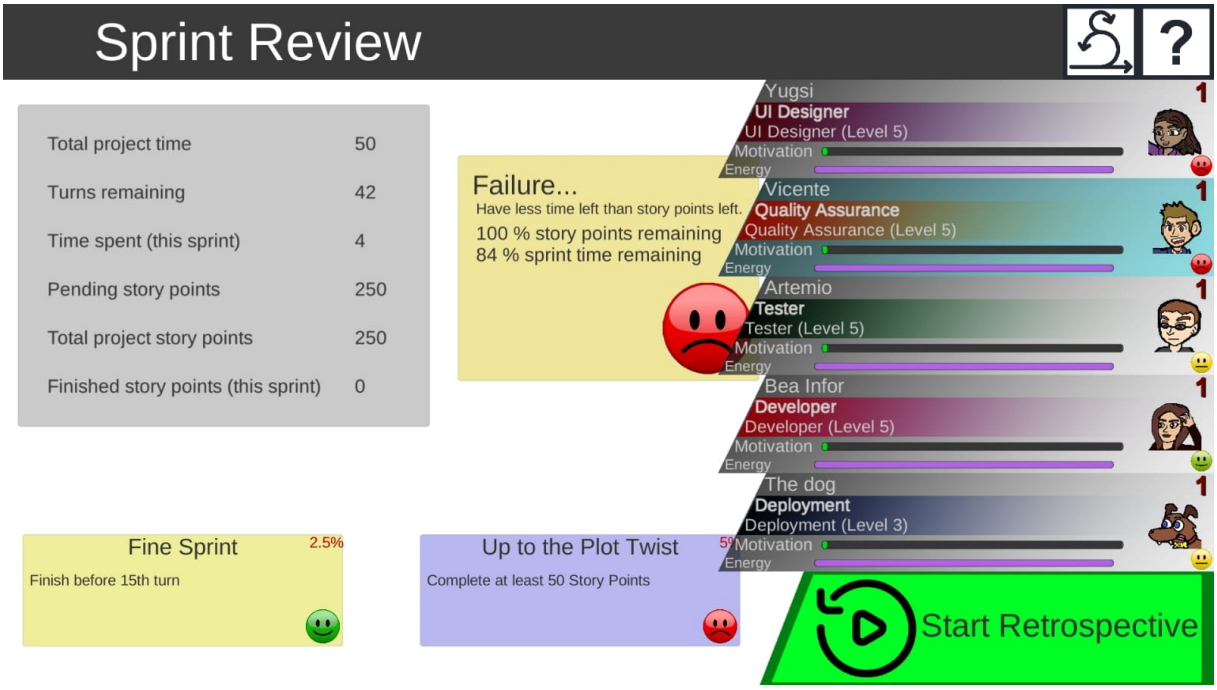



Figura 12.11: Apariencia final de la interfaz Sprint Review

El procedimiento era idéntico al del resto de Scrolls, tan solo variando en las restricciones existentes para que se añadieran las misiones al sprint al hacer click sobre ellas. Inicialmente, decidimos que se pudieran añadir hasta 6 misiones, que era la cantidad de tarjetas de misión que podían encajar bien en el la interfaz de Sprint Retrospective sin necesidad de modificarla. Para que las restricciones se pudieran cumplir, debíamos poner un contador de misiones. De esta manera, al llamar al contador, si el número de misiones que se pueden elegir coincidía con el del contador, la nueva misión no se añadiría.

Al igual que se podían añadir, también se deberían poder quitar una misión seleccionada si se había elegido por error o ya no se estaba conforme con la decisión. Para ello, añadimos un método de deselección de tareas que eliminaba la tarea de la lista de tareas del Sprint.

Para poder distinguir mejor si una tarea estaba o no en un Sprint, al bucle que instanciaba los botones le añadimos el coloreado del botón de la misión en verde si ya se había elegido. Además, añadimos una restricción para que no se pudiera escoger la misma misión más de una vez.

Al probar los métodos haciendo click en los botones y añadiendo y quitando misiones alternas, nos dimos cuenta de que, a pesar de que se añadían las misiones co-

rrectamente y en el orden que las pulsábamos, al volver a hacer click sobre ellas, a veces, se eliminaba una misión distinta de la lista. Pronto nos dimos cuenta de que se eliminaban en el mismo orden en el que se añadieron, independientemente del botón seleccionado.

Probando a hacer modificaciones de código y debuggeando, nos dimos cuenta de que Unity no era capaz de procesar las igualdades de las clases nuevas que habíamos creado. En este caso, no era capaz de encontrar las igualdades entre las misiones, por lo que, en su método correspondiente, se eliminaba la primera que encontrada en la lista del sprint y no la recibida en el parámetro de entrada.

Para solucionarlo, en las clase padre de las misiones, creamos métodos Equals manuales, compatibles con nuestro código, que comprobaban la igualdad de todos los atributos. Cada vez que quisiéramos comprobar la igualdad de dos misiones, tan solo llamaríamos a nuestro nuevo método. Fue en ese momento que las misiones del scroll comenzaron a escogerse y deseleccionarse correctamente para el siguiente sprint.

Decidimos, más adelante, que tan solo se podría escoger una misión de cada tipo. Por tanto, la restricción del contador se cambió a comprobar si en la lista de misiones ya había una con el mismo tipo de la misión introducida por parámetro. También modificamos las posiciones y los tamaños de los paneles de misión del menú Sprint Review para ajustarnos a esta nueva restricción. Además, en el Sprint Planning, añadimos un nuevo submenú que permitiría listar las misiones seleccionadas, para poder planificar más cómodamente el Sprint teniéndolas en cuenta.

## 12.8 Unificación de la jugabilidad

El último paso restante, sería completar el ciclo de la jugabilidad y realizar la llamada al siguiente sprint o al fin del proyecto, en el caso de haber terminado. Para esto, simplemente añadimos una función que, al pulsar el botón de terminar la ceremonia y empezar el siguiente, se ejecutara un método que comprobara la lista de tareas restantes y los turnos restantes del proyecto y, en función de los resultados, ejecutara una de las siguientes acciones:

- Si al proyecto le quedaban turnos y quedaban tareas pendientes, se hace visible la interfaz de Sprint Planning con los datos actuales del proyecto.
- Si no quedan turnos restantes pero sí tareas por realizar, se considera que el jugador ha perdido y se hace visible una pantalla de derrota.

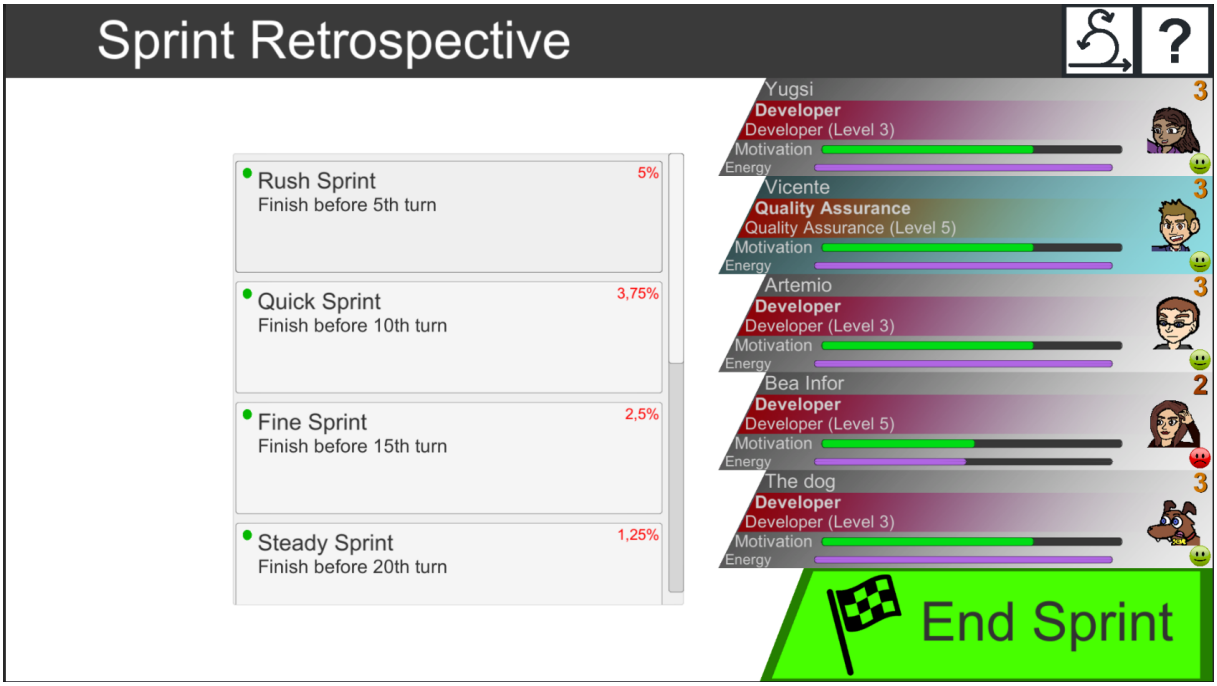



Figura 12.12: Apariencia final de la interfaz Sprint Retrospective

- Si ya no quedan tareas y quedan uno o más turnos, se considera que el jugador ha ganado y se hace visible una pantalla de derrota.

El único problema que tuvimos con esto fue que el segundo combate tenía problemas para iniciar. Descubrimos que era porque alguno de los eventos propios de ORK Framework necesarios para iniciarlo, como la posición ajustada de algunos personajes, no se estaba cumpliendo. De esta manera, decidimos añadir a la base de datos de ORK Framework eventos globales que correspondieran con los eventos necesarios para el funcionamiento. Estos eventos globales, cuentan con una ID en la base de datos que permite llamarlos en un Script con tan solo una línea de código.

De esta manera, se pudo conectar varios sprints y el proyecto ya era jugable en todo su ciclo. A partir de aquí tan solo fuimos puliendo aspectos del juego y solucionando pequeños bugs.


Figura 12.13: Pantalla de derrota

# Pruebas realizadas

*En este capítulo, trataremos las pruebas realizadas al juego durante el desarollo del proyecto.*

## 13.1 Introducción

Durante el transcurso de todo proyecto software, es necesario la realización de pruebas para minimizar el número de errores en el comportamiento de la aplicación final. Sabemos, además, que lo ideal es que estas pruebas se implementen por cada nueva funcionalidad añadida. A pesar de que lo óptimo en cualquier aplicación es realizar pruebas de todo tipo, en nuestro proyecto solo hemos podido implementar pruebas de aceptación. Esto ha sido producido por dos motivos principales:

- Falta de experiencia previa: la naturaleza de nuestro proyecto, como bien hemos expresado a lo largo de los distintos capítulos, es diferente al de una aplicación web, que es el tipo de aplicación al que se nos enfoca en nuestro grado. Además, nunca se ha trabajado con las herramientas Unity y ORK a lo largo del grado. Por el tipo de aplicación que es, no poseíamos conocimiento sobre cómo realizar pruebas unitarias ni pruebas de interfaz de usuario.
- Falta de tiempo: a pesar de que habíamos reservado horas del proyecto para las pruebas, la inexperiencia frente a la forma de realizar las pruebas para este tipo de aplicaciones no nos permitía adaptarnos al tiempo restante. El extenso alcance del proyecto también ha limitado el tiempo que podríamos dedicar a las pruebas, si no queríamos exceder la cota superior de horas.

Para las pruebas de aceptación para cada funcionalidad, se contemplaba que se realizaran justo tras su implementación, siguiendo las buenas prácticas. Además, en el bloque de desarrollo destinado a pulir el juego, reservaríamos tiempo tanto para hacer pruebas de aceptación adicionales como para que los usuarios piloto pudieran jugarlo, realizando sus propias pruebas, en busca de fallos que no hubiéramos podido encontrar.

Debido a nuestra situación, hemos sido aún más conscientes de los problemas que puede acarrear a la larga la falta de otros tipos de pruebas. Sin poder realizar pruebas automáticas, cada vez que realicemos un cambio en la aplicación, debemos de plantear pruebas de aceptación específicas en las que se pueda observar el comportamiento de las funciones afectadas, por lo que es mucho más probable que un error pase desapercibido.

A continuación, explicaremos las pruebas de aceptación más relevantes que hemos realizado a lo largo del proyecto.

## 13.2 Pruebas del Sprint Planning

| Caso de uso | Prueba realizada |
|---|---|
| CU-001<br><br>Mover una tarea entre Sprint Backlog y Product Backlog | P-001<br><br>▪ Iniciamos el juego<br>▪ Pulsamos el botón de selección de tareas en el menú de Sprint Planning<br>▪ Hacemos click en una tarea de la lista Product Backlog<br>▪ La tarea debe pasar a mostrarse en la lista Sprint Backlog, habiendo desaparecido de Product Backlog<br>▪ Para mayor seguridad, se añade una impresión por consola de las dos listas al realizar un movimiento de tarea.<br>▪ Si todo ha ido bien, pulsamos el botón de la tarea que hemos movido a Sprint Backlog.<br>▪ La tarea debe mostrarse de nuevo en el Product Backlog y haber desaparecido de la lita del Sprint Backlog |
| CU-002<br><br>Dividir una tarea | P-002<br><br>▪ Iniciamos el juego<br>▪ Accedemos al menú de selección de tareas del menú Sprint Planning.<br>▪ Hacemos click en una tarea de la lista Product Backlog |

| Caso de uso | Prueba realizada |
|---|---|
| | ▪ Pulsamos el botón de división entre dos que debe aparecer junto a la tarea que hemos pasado al Sprint Backlog.<br>▪ La tarea original debe haber desaparecido de Sprint Backlog y se debe haber sustituido por dos nuevas tareas del mismo tipo con la mitad de puntos de experiencia de la tarea original. |
| CU-003<br><br>Juntar tareas del mismo tipo | P-003<br>▪ Iniciamos el juego<br>▪ Accedemos al menú de selección de tareas del menú Sprint Planning.<br>▪ Hacemos click en una tarea de la lista Product Backlog<br>▪ Dividimos esta tarea, de manera recursiva, varias veces, de manera que aparezcan muchas tareas del mismo tipo con distinta cantidad de puntos de historia.<br>▪ Hacemos click en el botón sumatorio de cualquiera de las tareas de la lista<br>▪ En Sprint Backlog deben de haber desaparecido todas las tareas menores de la lista y haberse generado una tarea nueva, del mismo tipo, con el número de puntos de historia de la tarea original. |

| **Caso de uso** | **Prueba realizada** |
|---|---|
| CU-004<br><br>Un Sprint no puede empezar sin tareas | P-004<br><br>▪ Iniciamos el juego<br>▪ Comprobamos que, en el Sprint Planning, el botón de iniciar sprint está desactivado pulsándolo. No debe ocurrir nada.<br>▪ Hacemos click en una tarea de la lista Product Backlog<br>▪ Añadimos una tarea al Sprint<br>▪ Hacemos click en el botón que nos permite volver al menú principal del Sprint Planning<br>▪ Comprobamos que el botón de comenzar Sprint está activo pulsándolo. Debe iniciar una fase Daily Scrum. |

## 13.3 Pruebas de Daily Scrum

| Caso de uso | Prueba realizada |
|---|---|
| CU-005<br><br>La motivación afecta al daño causado | P-005<br><br>▪ Cambiamos el código de NikoNikoUpdate, forzando que los personajes tengan siempre emoticono neutral<br><br>▪ Cambiamos la fórmula de daño para que no procese nada posterior a la multiplicación del ataque (es decir, que no procese si es o no enemigo, la productividad, el número de golpes que ha recibido ni la penalización por Daily Scrum)<br><br>▪ Cambiamos el evento BattleUpdate, para que no procese ni UpdateMorale ni UpdateStage<br><br>▪ Cambiamos el evento TurnEndEvent para que no se reduzca la motivación de los personajes que no actúen<br><br>▪ Iniciamos el juego<br><br>▪ Iniciamos un Sprint con una tarea con 50 Puntos de Historia, que sea vulnerable a un personaje específico. El Sprint debe durar al menos 20 turnos, el personaje que realizará los ataques debe tener el rol al que el enemigo es vulnerable y no debe ser Scrum Master<br><br>▪ Atacamos con la habilidad básica del personaje que tenga el rol adecuado. Anotamos el daño causado<br><br>▪ Ahora el personaje debería tener motivación mínima. Atacamos de nuevo. Anotamos el daño causado |

| Caso de uso | Prueba realizada |
|---|---|
| | ▪ Con el Scrum Master, aumentamos la motivación del personaje en cuestión al máximo<br>▪ Con la motivación al máximo, atacamos con el personaje. Anotamos el daño causado<br>▪ Paramos el juego, vamos a la fórmula de daño y revisamos que los datos obtenidos coinciden con la fórmula |
| CU-006<br><br>Los stages y el ánimo se cambian de forma correcta y aplican el efecto correspondiente | P-006<br>▪ Cambiamos el código de NikoNikoUpdate, forzando que los personajes tengan siempre emoticono neutral<br>▪ Cambiamos el evento TurnEndEvent para que no se reduzca la motivación de los personajes que no actúen<br>▪ Iniciamos el juego<br>▪ Iniciamos un Sprint con una tarea con 3 Puntos de Historia. El Sprint debe durar al menos 40 turnos<br>▪ Comprobamos el ánimo y stage inicial de un personaje (en la interfaz de Unity, seleccionar el objeto del personaje, y en el Inspector, accedemos al componente “Combatant Component (Script)”, y desplegamos los Status Values). Morale debería ser 5 y Stage 3<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 5, Stage 3 y Attack 40 |

| | |
|---|---|
| | ▪ Con un personaje que no haga daño a la tarea de 3 PH atacamos a dicha tarea. Con la motivación bajada, el ánimo debería reducir en cada turno. Al final del turno, Morale debería ser 4, Stage 2 y Attack 30<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 3, Stage 2 y Attack 30<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 2, Stage 2 y Attack 30<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 1, Stage 1 y Attack 20<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 0, Stage 1 y Attack 20<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 0, Stage 1 y Attack 20<br>▪ Paramos la ejecución del juego<br>▪ Iniciamos el juego<br>▪ Iniciamos un Sprint con una tarea con 3 Puntos de Historia. El Sprint debe durar al menos 40 turnos<br>▪ Usando la habilidad “Ally Motivation”, aumentamos la motivación un miembro del equipo. Morale debería ser 6, Stage 3 y Attack 40<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 7, Stage 3 y Attack 40<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 8, Stage 3 y Attack 40 |

| | |
|---|---|
| | ▪ Finalizamos el turno sin hacer nada. Morale debería ser 9, Stage 3 y Attack 40<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 10, Stage 3 y Attack 40<br>▪ Usando la habilidad "Party Motivation", aumentamos la motivación todo el equipo. Revisamos las estadísticas de un miembro que no sea el mismo que revisamos antes. Morale debería ser 6, Stage 3 y Attack 40<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 7, el Stage de todo el equipo 4 y Attack de todo el equipo 50<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 8, el Stage de todo el equipo 4 y Attack de todo el equipo 50<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 9, el Stage de todo el equipo 5 y Attack de todo el equipo 60<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 10, el Stage de todo el equipo 5 y Attack de todo el equipo 60<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 10, el Stage de todo el equipo 5 y Attack de todo el equipo 60<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 10, el Stage de todo el equipo 4 y Attack de todo el equipo 50<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 10, el Stage de todo el equipo 4 y Attack de todo el equipo 50 |

| | |
|---|---|
| | ▪ Finalizamos el turno sin hacer nada. Morale debería ser 10, el Stage de todo el equipo 4 y Attack de todo el equipo 50<br>▪ Finalizamos el turno sin hacer nada. Morale debería ser 10, el Stage de todo el equipo 5 y Attack de todo el equipo 60<br>▪ Paramos la ejecución del juego |
| CU-007<br><br>Los estados niko-niko alteran la productividad de forma relativa a la estabilidad del personaje | P-007<br>▪ Cambiamos la fórmula de daño para que no tenga en cuenta la motivación, la penalización de Daily ni el número de veces que es golpeado un enemigo<br>▪ Cambiamos el evento BattleUpdate para que no actualice el ánimo ni la moral<br>▪ Cambiamos el código de NikoNikoUpdate para forzar siempre caras tristes<br>▪ Nos aseguramos de tener un personaje con estabilidad 100 y otro con estabilidad 0, y hacemos que ambos tengan el mismo nivel en el rol Developer<br>▪ Iniciamos el juego<br>▪ Iniciamos un Sprint con una tarea con 50 PH que sea vulnerable al rol Developer<br>▪ Atacamos a la tarea con ambos personajes. Anotamos el daño que causa cada uno. El personaje con mayor estabilidad debería hacer más daño que el personaje con menor estabilidad<br>▪ Paramos la ejecución del juego |

| | |
|---|---|
| | ▪ Cambiamos el código de NikoNikoUpdate para forzar siempre caras felices<br>▪ Iniciamos el juego<br>▪ Iniciamos un Sprint con una tarea con 50 PH que sea vulnerable al rol Developer<br>▪ Atacamos a la tarea con ambos personajes. Anotamos el daño que causa cada uno. El personaje con mayor estabilidad debería hacer menos daño que el personaje con menor estabilidad<br>▪ Paramos la ejecución del juego<br>▪ Revisamos si el daño causado es coherente con la fórmula de alteración de la productividad vinculada a los estados Niko-Niko y la estabilidad |
| CU-008<br><br>Atacar a una tarea varias veces en un turno divide el daño recibido por el número de ataques recibido + 1 | P-008<br>▪ Cambiamos el código de NikoNikoUpdate para forzar siempre caras neutrales<br>▪ Hacemos que todos los personajes tengan el mismo nivel del rol Developer<br>▪ Iniciamos el juego<br>▪ Iniciamos un Sprint con una tarea con 50 PH que sea vulnerable al rol Developer<br>▪ Hacemos que todos los personajes hagan daño a la tarea. Anotamos el daño causado de cada acción consecutiva<br>▪ Paramos la ejecución del juego |

| | |
|---|---|
| | ▪ Revisamos si el daño causado por cada acción es correcto (daño original / (número ataques recibidos + 1)) |
| CU-009<br><br>Las entradas del gráfico Burndown se añaden de forma correcta | P-009<br>▪ Iniciamos el juego<br>▪ Iniciamos un Sprint con 10 tareas de 0,5 PH cada una, todas vulnerables al rol Developer. Hacemos que todos los miembros del equipo sean Developers. El Sprint debe durar 10 turnos.<br>▪ Cada turno hacemos que un personaje complete una tarea (debería poder vencer a cualquier tarea independientemente de su motivación, productividad y Stage)<br>▪ Comprobamos el gráfico Burndown cada turno. Siempre y cuando el gráfico Burndown siga la línea ideal, debería estar funcionando bien<br>▪ Para mayor precisión, podemos imprimir por consola los componentes de cada entrada Burndown que se va generando y compararlo con el valor esperado |
| CU-010<br><br>No puede realizarse una acción que cueste más energía de la que le queda al personaje | P-010<br>▪ Iniciamos el juego<br>▪ Iniciamos un Sprint. El Sprint debe durar al menos 10 turnos<br>▪ El personaje que es Scrum Master debe elegir hacer “Party Motivation” y “Intense ____” |

| | |
|---|---|
| | ▪ El personaje debería quedarse sin energía y debe efectuar ambas acciones. De esta forma nos aseguramos de que el personaje, en condiciones normales, puede hacer dos acciones distintas en un turno<br>▪ Ese mismo personaje no podrá actuar el próximo turno (por no tener suficiente energía), por lo que terminamos el turno sin hacer nada<br>▪ Ahora el personaje debería tener 20 de energía, suficiente para poder hacer "Ally Motivation". Elegimos hacer esa habilidad, una habilidad de cambio de rol y una habilidad que cueste algo de energía<br>▪ Se deberían ejecutar las dos primeras acciones ("Ally Motivation" y cambio de rol), pero la tercera no debería poderse ejecutar |

## 13.4 Pruebas de Sprint Review

| Caso de uso | Prueba realizada |
|---|---|
| CU-011<br><br>No se muestran misiones opcionales en el primer Sprint | P-011<br><br>▪ Iniciamos el juego<br>▪ En el menú Sprint Planning, el botón de la lista de misiones debe estar deshabilitado. Hacemos click sobre l botón, no debe ocurrir nada.<br>▪ Añadimos una tarea de 6 puntos de historia en el Sprint Planning, del rol developer.<br>▪ Elegimos superrol developer para el personaje que tenga nivel 5 en este rol.<br>▪ Comenzamos el Sprint.<br>▪ Derrotamos a la tarea con el personaje que tiene el rol developer en nivel 5 con intense develop.<br>▪ Cuando la tarea sea derrotada, nos debe mandar al menú de Sprint Review.<br>▪ Observamos que, en el menú, solo aparece una única misión indicando los ratios de experiencia y número de turnos. |
| CU-012<br><br>Las misiones se evalúan correctamente en la Sprint Review | P-012<br><br>▪ Iniciamos el juego<br>▪ Añadimos una tarea de 6 puntos de historia en el Sprint Planning, del rol developer y configuramos los roles acorde a la tarea.<br>▪ Comenzamos el Sprint.<br>▪ Derrotamos a la tarea lo más rápido posible. |

| Caso de uso | Prueba realizada |
|---|---|
| | ▪ Cuando la tarea sea derrotada, nos debe mandar al menú de Sprint Review, cuya misión solo será la de los ratios.<br>▪ Le damos al botón que nos lleva a la Sprint Retrospective<br>▪ Escogemos la primera tarea de la lista haciendo click en su botón<br>▪ Realizaremos otro Sprint con la misma configuración, 6 puntos de historia de developer y el dejaremos los mismos roles.<br>▪ Iniciaremos el Sprint, curando en el primer turno con el Scrum Master al personaje developer y realizando intense develop con el personaje developer, para asegurarnos de acabar rápido la tarea.<br>▪ Al finalizar el Sprint, se abrirá el menú de Sprint Review.<br>▪ Debería aparecer, además de la misión de los ratios básica, la misión que escogimos en la reunión Sprint Retrospective del primer Sprint, con una cara feliz, ya que habremos acabado la misión antes de los 5 turnos. |

## 13.5 Pruebas de Sprint Retrospective

| Caso de uso | Prueba realizada |
|---|---|
| CU-013<br><br>La selección de misiones funciona correctamente en la interfaz de Sprint Retrospective | P-013<br><br>▪ Iniciamos el juego<br>▪ Añadimos una tarea de 6 puntos de historia,para hacer un sprint rápido.<br>▪ Configuramos los roles de los personajes acorde a la tarea escogida.<br>▪ Comenzamos el Sprint y realizamos la tarea lo antes posible.<br>▪ Cuando la tarea sea derrotada, nos debe mandar al menú de Sprint Review. Pulsamos el botón que nos lleva al menú Sprint Retrospective.<br>▪ Debe aparecer una lista con las misiones disponibles<br>▪ Hacemos click en el botón correspondiente a la primera misión de la lista.<br>▪ Si está en la lista la lista, el botón de la tarea aparecerá de color verde.<br>▪ Hacemos click en la segunda misión, que es del mismo tipo.<br>▪ Solo podemos añadir una misión de cada tipo. La primera misión que seleccionamos se deseleccionará y se seleccionará la segunda misión.<br>▪ Hacemos click en la misión seleccionada para deseleccionarla. El botón volverá a ser blanco. |

| **Caso de uso** | **Prueba realizada** |
|---|---|
| CU-014<br><br>La selección de tareas elegidas en la Sprint Retrospective se mantiene en el Sprint Planning | P-014<br><br>▪ Iniciamos el juego<br><br>▪ Añadimos una tarea de 6 puntos de historia,para hacer un sprint rápido y configuramos los roles de los personajes en consecuencia.<br><br>▪ Comenzamos el Sprint y realizamos la tarea lo antes posible.<br><br>▪ Cuando la tarea sea derrotada, nos debe mandar al menú de Sprint Review. Pulsamos el botón que nos lleva al menú Sprint Retrospective.<br><br>▪ Hacemos click en el botón correspondiente a la primera misión de la lista.<br><br>▪ Pulsamos en el botón para terminar Sprint, nos llevará al Sprint Planning para configurar el segundo Sprint.<br><br>▪ Pulsamos el botón de la lista de misiones, que debe haberse activado.<br><br>▪ En la lista debe mostrarse solo la misión que escogimos en el menú Sprint Retrospective. |

## 13.6 PRUEBAS DE UNIFICACIÓN DE JUGABILIDAD

| Caso de uso | Prueba realizada |
|---|---|
| CU-015<br><br>Si se acaban los turnos, se produce una derrota. (Sin Acceso al código) | P-015<br><br>▪ Iniciamos el juego<br><br>▪ Añadimos cualquier tarea y comenzamos un Sprint del máximo número de turnos que nos permita, 40.<br><br>▪ En todos los turnos, pulsamos el botón *End Daily Scrum* para terminarlo sin hacer nada.<br><br>▪ Al acabar el combate, en el menú de Sprint Review, pulsamos para ir al Sprint Retrospective. Sin escoger misiones, iniciaremos un nuevo Sprint.<br><br>▪ Añadimos la misma tarea que en el primer Sprint y configuramos la duración máxima del proyecto, 10.<br><br>▪ Comenzamos el Sprint.<br><br>▪ En todos los turnos, pulsamos el botón *End Daily Scrum* para terminarlo sin hacer nada.<br><br>▪ Pasamos por los menús Sprint Review y Sprint Retrospective sin hacer nada. Pulsamos en nuevo Sprint.<br><br>▪ Nos debe aparecer la pantalla de derrota.<br><br>▪ Otra manera de probar que se muestra la derrota correctamente es modificar en la clase Project el número de turnos máximo por 1.<br><br>▪ iniciar el juego e iniciamos un sprint de un turno |

| Caso de uso | Prueba realizada |
|---|---|
| | ▪ No haciendo nada en el turno y avanzando por los menús Sprint Review y Sprint Retrospective, nos mandará a la pantalla de derrota. |
| CU-016<br><br>Si se acaban todas las tareas antes que acaben los turnos, se produce una victoria. (Accediendo al código) | P-016<br>▪ Modificamos del product backlog inicial del proyecto para que contenga solo una tarea de tipo develop con 500 puntos de historia.<br>▪ Iniciamos el juego<br>▪ Añadimos la tarea al Sprint y hacemos que todos los personajes tengan rol developer, que aparecerá con 5 puntos de historia.<br>▪ Comenzamos un Sprint con 5 turnos.<br>▪ En el primer turno, derrotamos a la tarea con el personaje que tenga nivel 5 de rol developer.<br>▪ Al acabar el combate, en el menú de Sprint Review, pulsamos para ir al Sprint Retrospective. Sin escoger misiones, iniciaremos un nuevo Sprint.<br>▪ Nos debe aparecer la pantalla de victoria.<br>▪ Si no queremos modificar el código, simplemente jugaremos todo el proyecto acabando las tareas antes de 50 turnos y comprobaremos si nos aparece la pantalla de victoria. |

| **Caso de uso** | **Prueba realizada** |
|---|---|
| CU-017<br><br>Las tareas que no se derrotaron en un Sprint, se conservan para el siguiente. | P-017<br><br>▪ Comenzamos un Sprint con 5 turnos.<br>▪ Aunque no vamos a poder derrotarla, intentamos atacar a la tarea durante los 5 turnos de la Daily Scrum.<br>▪ En el menú de Sprint Review, la misión de los ratios nos dirá que hemos derrotado el 0% de las tareas.<br>▪ No escogemos tarea en el menú Sprint Retrospective.<br>▪ Pulsamos en el botón para comenzar un nuevo Sprint.<br>▪ En el menú Sprint Planning, pulsamos en el botón que nos lleva a la selección de tareas.<br>▪ La tarea de 50 puntos de developer debe estar al final de la lista de tareas del Product Backlog. Volvemos a elegirla.<br>▪ Comenzamos el Sprint con 5 turnos.<br>▪ Debe aparecer la tarea de 50 puntos de developer con el 100% de realización pendiente.<br>▪ Iniciamos el juego<br>▪ Añadimos una tarea de 50 puntos de historia, de clase developer, al Sprint y hacemos que todos los personajes tengan rol developer. |

## 13.7 Matriz de trazabilidad

| CU\P | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | X | | | | | | | | | | | | | | | | |
| 2 | | X | | | | | | | | | | | | | | | |
| 3 | | | X | | | | | | | | | | | | | | |
| 4 | | | | X | | | | | | | | | | | | | |
| 5 | | | | | X | | | | | | | | | | | | |
| 6 | | | | | | X | | | | | | | | | | | |
| 7 | | | | | | | X | | | | | | | | | | |
| 8 | | | | | | | | X | | | | | | | | | |
| 9 | | | | | | | | | X | | | | | | | | |
| 10 | | | | | | | | | | X | | | | | | | |
| 11 | | | | | | | | | | | X | | | | | | |
| 12 | | | | | | | | | | | | X | | | | | |
| 13 | | | | | | | | | | | | | X | | | | |
| 14 | | | | | | | | | | | | | | X | | | |
| 15 | | | | | | | | | | | | | | | X | | |
| 16 | | | | | | | | | | | | | | | | X | |
| 17 | | | | | | | | | | | | | | | | | X |

# Parte VI

# Cierre

# Conclusiones

*En este último capítulo expondremos las conclusiones finales del proyecto, incluyendo la parte positiva, los contratiempos sufridos y los pasos a seguir en el futuro para seguir desarrollando este trabajo.*

## 14.1 Informe postmortem

En esta sección detallaremos aspectos de nuestro proyecto que servirán para proyectos de calibre similar. Lo presentaremos en forma de logros conseguidos, errores cometidos y errores evitados.

### 14.1.1 Logros

Identificamos dos grandes logros, uno de ellos era el objetivo de nuestro proyecto y el otro es una consecuencia de lo distinto que es nuestro proyecto de otras aplicaciones de carácter similar:

- **Nueva forma de enseñar Scrum:** Nuestro objetivo era crear una forma nueva de enseñar Scrum que no se hubiera implementado antes.

  Esta no es la única versión gamificada de Scrum, pero lo más frecuente son juegos que imitan el formato de Trivial o usan un formato de cuestionarios para hacer preguntas teóricas de Scrum y otorguen puntos; en otras palabras, examinan tu conocimiento de Scrum de forma divertida, pero no enseñan Scrum ni te exponen una situación real de uso de Scrum.

  En ese sentido, nuestro proyecto es algo único que no sólo permite al usuario conocer términos teóricos de Scrum, sino que además expone una serie de casos de uso reales, donde el jugador está obligado a participar y debe ingeniar técnicas y estrategias para superar los distintos obstáculos que se le presentan.

- **Reinvención de la fórmula RPG:** Aunque no fuese el objetivo de nuestro proyecto, es cierto que hemos reinventado gran parte de la fórmula tradicional de juegos RPG por turnos. Vayamos por partes:

  - *Vida de los personajes:* La fórmula tradicional dicta que todos los personajes empiezan con el 100 % de su vida, y si se reduce se deben utilizar otros recursos para aumentarla, y su única función es determinar si el personaje es capaz de luchar o no.

    Nosotros hemos cambiado esto completamente, haciendo que los personajes empiecen con su vida al 50 % y que no sólo consumiendo recursos externos (energía en forma de habilidades de Scrum Master en nuestro caso) se pueda aumentar; por ejemplo, si un personaje vence a una tarea, aumenta su motivación (o vida). Además, nuestra *vida* no dicta si el personaje es capaz de

actuar o no, sino cómo de útil es ese personaje, haciendo que la motivación sea un factor a tener en cuenta constantemente, y que tener 50 % no sea lo mismo que tenr 60 %.

Por otro lado, sí que hemos mantenido el formato tradicional para las tareas, rompiendo una vez más otro formato universal; tradicionalmente, los personajes del jugador y sus oponentes comparten el mismo conjunto de estadísticas, pero aquí hemos querido distinguirlos.

- *Sistema de turnos:* En los juegos RPG tradicionales, el jugador elige las acciones de sus personajes en un orden determinado, pero luego las acciones se ejecutan en un orden distinto que depende de factores internos del juego.

  Nosotros hemos decidido darle al jugador la libertad de elegir el orden de las acciones de sus personajes, rompiendo ese sistema. Además, hemos añadido otra característica más: la existencia de acciones que no consuman el turno entero, pero que puedan limitar las acciones restantes del turno.

  Ambos conceptos no son nuevos, pero normalmente se limitan a juegos RPG tácticos[27], por lo que incluirlos en el sistema RPG más tradicional rompe varias ideas propias de dicho sistema.

- *Dinámica del juego:* La dinámica tradicional de este género sigue los siguientes pasos:

  1. Viajar en un mapa
  2. Gestionar equipo (generalmente en un pueblo)
  3. Enfrentar enemigos aleatorios
  4. Actualización de recursos en base al enfrentamiento previo
  5. Volver al paso 1

  Nuestro juego se aleja de esta dinámica. No existe la capacidad de viajar, las distintas gestiones no consumen recursos, pero son excluyentes, y los enemigos no son aleatorios, sino que son elegidos.

  La dinámica de nuestro juego sería la siguiente:

  1. Gestionar equipo (Sprint Planning y Retrospective)
  2. Elegir enemigos (Sprint Planning)
  3. Enfrentarse a enemigos seleccionados (Daily Scrum)
  4. Actualización del estado del equipo en base al éxito o fracaso del enfrentamiento (Sprint Review)
  5. Volver al paso 1

A continuación listamos los errores que hemos cometido a lo largo del desarrollo del proyecto y los que hemos evitado.

### 14.1.2 Errores cometidos

- **Decidir no utilizar framework y rectificar tarde:** Nosotros empezamos el TFG en Junio de 2021 aproximadamente. El plan era que Isabel aprendiera a usar Unity en ese tiempo y Matthew construyera un framework que se adaptase a las necesidades de nuestro proyecto en ese tiempo. Hasta Octubre no decidimos cancelar esta idea y buscar nosotros mismos un framework, cuando es una decisión que debía haberse tomado antes.

- **Aprendizaje del framework ineficiente:** Al elegir el framework, tuvimos que aprender a utilizarlo. Debido a que otras asignaturas nos consumían tiempo, no podíamos dedicarle tiempo suficiente a planificar el cómo aprenderíamos a utilizar la herramienta, y decidimos que cada uno se gestionase como viera conveniente. El resultado fue que invertimos mucho más tiempo del que deberíamos haber invertido en este aprendizaje. Si sumamos este retraso al mencionado en el error previo, vemos que perdimos mucho tiempo total en los primeros meses.

- **No haber empezado desde la plantilla ofrecida:** El framework escogido ofrecía un tutorial que, si seguías de principio a fin, daba lugar a un proyecto simple con varias mecánicas implementadas. Este proyecto está disponible en la página web del framework, se puede descargar y usar como plantilla inicial.

  Decidimos hacer nuestro proyecto desde el principio por varios motivos, uno de ellos es porque la plantilla era de un proyecto en 3D, mientras nuestro juego era 2D. Sin embargo, haber empezado desde la plantilla hubiera ahorrado tiempo de crear algunos elementos básicos en la base de datos.

- **Elección del framework:** Rectificamos y elegimos un framework ya implementado que se adaptara a nuestras necesidades, pero no sabíamos que el framework se alejase tanto de lo que realmente queríamos.

  En nuestra defensa diremos que consultamos múltiples frameworks, y este era el que mejor parecía, nos guiamos por opiniones de usuarios, por características, manuales de usuario, etc. Aún a día de hoy sigo pensando que no vimos un framework mejor, pero aún así, elegir este framework fue un error.

  Este framework ha traído excesivos problemas:

- Es muy rígido, y al parecer está pensado para usuarios que tengan poco conocimiento de programación, ya que da poco soporte para complementarse con código que haya hecho el usuario.
- No estaba bien diseñado. Haciendo uso exclusivamente de funcionalidades ofrecidas por el framework era fácil encontrar errores y comportamientos no intencionados.
- Ofrece muchísimas opciones, pero todas son muy superficiales y no ofrecen forma de hacerlas más complejas.

En definitiva, deberíamos o bien haber seguido con el plan original de construir nosotros el framework, o utilizar RPG Maker o alguna herramienta similar que sepamos que funciona bien para el tipo de proyecto que queríamos diseñar.

### 14.1.3 Errores evitados

- **Limitaciones del framework:** La elección del framework, como ya hemos dicho, trajo consigo una serie de limitaciones. Aún así, pudimos sortearlas y hacer el proyecto que queríamos haciendo uso de nuestros conocimientos de Unity, y haciendo llamadas a la base de datos del framework a través de nuestro propio código para crear las funcionalidades que el framework debería haber ofrecido desde el principio.

  Algunos ejemplos son:

  1. Interfaz de usuario original ajena al framework
  2. Sistema de varias acciones por personaje y orden de turnos correctamente funcional
  3. Creación de un grupo de enemigos basado en las elecciones del usuario

- **Flexibilidad con la fecha límite:** Debido al retraso de los primeros meses y las limitaciones del framework, no podíamos tener un proyecto decente para Junio, por lo que hicimos la planificación teniendo en cuenta que no entregaríamos el proyecto en Junio. En Septiembre teníamos algo razonable, pero aún podíamos acercarnos más a nuestro límite de 300 horas y no vimos necesidad de darlo por terminado antes de tiempo, pudiendo hacer una mejor versión del mismo. Finalmente, el proyecto será entregado en la convocatoria de Noviembre.

  Creemos que esta toma de decisiones ha sido clave para contraarrestar prácticamente todos los errores cometidos. Estamos contentos de haber retrasado la

entrega, y confiamos en que el resultado de estos atrasos es un proyecto del que estamos realmente orgullosos.

- **Usar Unity Collab:** Para el tipo de archivos que estábamos usando, Git no funcionaba, por lo que tuvimos que empezar a usar Git LFS, pero era de pago a partir de hacer commits de cierto tamaño, así que exploramos otras posibilidades ya que su uso era inviable. Usamos Unity Collab (que luego migró a Plastic SCM), herramienta que nos ayudó mucho a gestionar el código fuente de nuestra aplicación. Si en cambio hubiéramos insistido en usar Git o compartir los archivos via Google Drive o algo similar, hubieramos tenido problemas serios.

## 14.2 Lecciones aprendidas

Dado que este TFG ha sido realizado por dos alumnos, esta sección será dividida en dos secciones, una por cada uno de nosotros.

### 14.2.1 Lecciones aprendidas por Isabel Arrans Vega

Una gran parte de las lecciones que he aprendido en este proyecto se pueden condensar en una pequeña frase: "He aprendido a usar Unity". Comencé la implementación de este trabajo asustada, ya que lo habitual en proyectos importantes es emplear tecnologías conocidas para disminuir los riesgos y la incertidumbre y yo apenas sabía manipular la interfaz principal del editor de Unity. Decidí aventurarme, sin embargo, gracias a que mi compañero tenía experiencia y a la cantidad de contenido educativo que se puede encontrar empleando cualquier navegador.

A medida que fue avanzando el proyecto, fui haciendo progresos, poco a poco, integrando en mi conocimiento conceptos nuevos y usos específicos de Unity. Decidí atreverme con tareas complejas y eso me sirvió para aprender más. Con cada concepto nuevo que aprendía, trataba de adaptar los códigos para que funcionaran mejor. Además, el tener a mi compañero para aconsejarme cuando me encontraba con una tarea más difícil para mí, hacía que pudiera lentamente seguir avanzando. Al final, resultó que las interfaces se asimilaban fácilmente a medida que se iban usando y que C# era casi un calco de Java. Sin duda, a raíz de estas situaciones, también aprendí una importante lección. Salimos del grado con capacidad suficiente para aprender cualquier tecnología, por lo que no debemos atemorizarnos de ello.

El haber aprendido a trabajar en un proyecto real con Unity me produce satisfac-

ción, ya que uno de mis objetivos personales, a nivel laboral, es dedicarme al mundo de los videojuegos, donde se emplea este motor gráfico en un gran número de casos. En resumen, siento que este proyecto me abre puertas al desarrollo de futuros videojuegos y que me acerca más al sector.

Otra de las cosas que más claras me han quedado durante el transcurso del proyecto ha sido la importancia de fijar días y horas para realizar las daily meetings, pero que estas deben tener también cierta flexibilidad. Al principio del proyecto, no teníamos días asignados para realizar dailies y, a veces, se espaciaban demasiado en el tiempo. No solo esto nos desorientaba respecto al avance del otro sino que además, al no tener la "presión"de tener que haber avanzado en las tareas poder comunicarlo, hacía que avanzáramos más lento. Sin embargo, tener los días y horas estrictamente fijos tampoco ayudó, puesto que muchas veces, por las situaciones personales de cada uno, no podíamos acudir uno de los dos a la daily y no teníamos claro dónde reubicarla. En las semanas más difíciles de coordinar, durante las vacaciones de verano, también probamos una fórmula de realizar las dailies por escrito.

Durante el transcurso del proyecto, fuimos planteando diferentes estrategias para la daily meeting. Aprendí la importancia de sintetizar esta reunión para que interrumpiera lo menos posible el avance en el proyecto. Al inicio del proyecto, las dailies podían llegar a durar una media hora, por lo que eran más difíciles de encajar y, hacia el final, pudimos condensarlas en 5 minutos o menos, haciéndola sencilla de encuadrar en cualquier día. Si volviera a comenzar un proyecto aplicando Scrum, sería capaz de ahorrar más tiempo al inicio de este.

Además de las lecciones anteriores, también relacionado con el empleo de Scrum, he comprendido la importancia de la elección de la duración de un Sprint para un mejor rendimiento. De nuevo, se trata de límites temporales. Generalmente, hemos trabajado con Sprints de entre dos y cuatro semanas de duración, los límites establecidos por la metodología. Sin embargo, en algunas ocasiones, la duración de los Sprints ha sido planificada más larga de lo recomendado y esto nos ha reducido el rendimiento. El mayor ejemplo de esto ocurrió entre abril y junio, correspondiendo con el fin del segundo cuatrimestre. Decidimos hacer un Sprint de mes y medio de duración. Se completaron todas las tareas, pero debido al amplio periodo de tiempo y a otras asignaturas, no trabajamos tan bien como si hubiéramos dividido ese sprint en dos o tres sprints de menor duración y con menor número de tareas cada uno.

Al final, este proyecto es en el de mayor duración y alcance en el que he participado, por lo que cualquier lección sobre procedimientos a largo plazo que haya podido

recibir durante el grado, se ha visto reflejado más que nunca. Mis conocimientos sobre Scrum se han visto reforzados y he aprendido a trabajar con una tecnología que, en el ámbito personal, me va a ayudar de cara al futuro. Hoy día, me siento mucho más capaz de emprender un proyecto software, ya sea tanto de videojuegos como de otro ámbito, que antes de realizar este trabajo de fin de grado.

### 14.2.2 Lecciones aprendidas por Matthew Bwye Lera

"No reinventar la rueda."

Creo que con esta frase se sintetiza una de las primeras lecciones que aprendí. Aunque no estuviese realmente reinventando algo, estaba construyendo algo que ya existía y era accesible; empecé el TFG intentando crear un framework. Tenía motivos personales (ajenos al TFG) para hacer esto, pero crear dicho framework ya sería tarea suficiente para un TFG de por sí, y no fui capaz de verlo.

Puede que sea porque somos seres guiados por impulsos, pero observar bien de qué recursos disponemos y planificar cómo gestionarlos es una lección que aprendemos recurrentemente en la vida. Durante este proyecto puedo pensar en al menos 3 ocasiones en las que fallé aplicando esta lógica, y yo considero que no se me da mal ni planificar anticipadamente, ni hacer pronósticos realistas, ni pensar antes de actuar, pero aún así esto pasa en ocasiones. Voy a listar estas 3 ocasiones y explicar por qué hice lo que hice, y cómo lo haría ahora:

1. **Crear un framework:** Como ya hemos dicho, la carga de trabajo era excesiva. El motivo por el que pensé que era viable es porque subestimé la carga de trabajo, tanto la de la realización del framework como la del resto del proyecto. Mientras estaba haciendo el framework me di cuenta de la dimensión de lo que estaba haciendo y de cuantos problemas podrían identificarse durante su uso. Al ver esto, planeé cómo enfrentar estos problemas en lugar de seguir con la implementación, y me di cuenta de lo inviable que era.

   Si empezase este trabajo ahora, primero planearía bien qué componentes hay que implementar para el framework, cuantos hay que implementar para el proyecto, ver cuales se solapan, hacer una estimación temporal de cada uno, e intentar encajarlo con el tiempo del que disponemos. Si el tiempo total invertido superase el tiempo del que disponemos, haría una clasificación siguiendo dos métricas: importancia e importabilidad. Es decir, cuanto menos importante sea algo y más fácil sea de importar (de un framework externo o consumo de una API por ejem-

plo), más probablemente lo deje fuera del marco de trabajo. Aquello que quede fuera no se implementará o será importado de proyectos ajenos.

Esta lógica es la que seguimos a la hora de elegir un framework y, a pesar de las decepciones que nos ha causado, hemos conseguido cumplir lo que nos propusimos.

2. **Aprender a utilizar ORK Framework:** Esto también se explica en la sección de errores cometidos[§14.1.2]. Isabel y yo tuvimos problemas para coordinarnos por incompatibilidad horaria, así que decidimos que cada uno aprendiera a utilizar el framework por su cuenta. Consecuentemente, no planificamos previamente, de hecho, ni siquiera pusimos fechas límites, que es lo más elemental. Consecuentemente, seguimos un proceso caótico que llevó más tiempo del que debería.

   Ahora mismo, si se me presentase una situación similar, primero aplicaría una cosa que sí hicimos bien: tener reuniones regulares para compartir qué ha aprendido cada uno y hacer un seguimiento de los avances de cada uno, pero además de eso añadiría fechas límite, y las respaldaría con estimaciones temporales del resto del proyecto.

3. **Implementar sin diseñar:** Muy relacionado con la última frase del párrafo previo, nosotros no hicimos una planificación detallada antes de implementar.

   Aunque pueda parecer disparatado, seguimos una lógica apropiada, y ambos estábamos de acuerdo, y nuestros tutores aprobaron nuestra decisión. No conocíamos el framework, no sabíamos exactamente qué había implementado, qué había que implementar, y cuánto costaría hacer lo que tuvieramos que hacer, por lo que pensamos que era mejor saltar a una primera etapa de implementación para entender la herramienta, para luego empezar a diseñar.

   Sinceramente, no me parece que hayamos tomado una mala decisión, por eso mismo no figura entre los errores cometidos. Aún así, ahora yo detallaría un poco más qué necesitamos, incluso si con esquemas intuitivos informales y, más importante aún, **haría una estimación temporal** para saber de cuánto tiempo disponemos para decidir qué framework usar y cómo utilizarlo. En otras palabras, los dos elementos mencionados antes[1][2] son, en mi opinión, consecuencia de haber hecho una planificación previa insuficiente.

Por otro lado, aunque muy relacionado con estos puntos mencionados, me siento ahora mucho más capaz de diseñar un sistema. La diferencia respecto a todo lo que haya hecho en el grado es que esta vez para completar el diseño de forma precisa he

hecho uso de ingeniería inversa. Nunca antes había hecho esto en un proyecto real, y menos de este tamaño, y creo que me ha enseñado tanto a ser capaz de aplicar ingeniería inversa de nuevo en el futuro, como a ser capaz de diseñar sistemas más perfectos.

A pesar de que el framework escogido ha sido muy problemático, no se me ocurre una herramienta que hubiera funcionado mejor en Unity, por lo que poco puedo recomendar respecto a esto. Aún así, sí que hay algo que quisiera mencionar. Dentro de la universidad se nos han presentado mejores y peores frameworks, pero creo que en general el alumnado asume que al llegar al mundo laboral los frameworks que van a usar serán mejores que los que han usado en la universidad. En este proyecto he aprendido que incluso el mejor framework que puedas encontrar no tiene por qué ser bueno, y debes aprender a ser flexible con lo que se te ofrece. En otras palabras, no debemos asumir que nuestra experiencia con frameworks frustrantes acaba en la universidad, y aquello que aprendimos en esas asignaturas nos será de ayuda en el futuro.

Por último, aunque es algo muy personal, siento que ahora me vería mucho más capaz de hacer el framework que me propuse hacer en un principio. Al hacer uso de algo, entiendes qué hace bien y qué hace mal. Siento que tendría muy presente aquellas frustraciones que encontré al usar ORK Framework y trataría de evitarlas.

## 14.3 Actualidad del proyecto

Como hemos mencionado en secciones anteriores, contactamos con usuarios piloto para poder terminar de perfilar el juego. El pilotaje tenía tres objetivos fundamentales: pulir los bugs de la aplicación que no hubiéramos encontrado, comprobar que las mecánicas del juego se entendían y asegurarnos de que el juego servía para aprender Scrum.

Para crear las encuestas, enviarlas y obtener los resultados cómodamente, empleamos la herramienta Google Forms, sencilla de usar tanto para nosotros como para los usuarios piloto, permitiendo exportar las respuestas como gráficas. Decidimos contactar con amigos estudiantes del grado y externos a este, a fin de poder comprender cómo funcionaba el juego para distintos niveles de conocimiento software.

Como quisimos hacer un formulario cómodo de responder para evitar el rechazo de los usuarios piloto a rellenarllo, las preguntas que consideramos funcamentales, tienen múltiples opciones o son dicotómicas. Dividimos las preguntas en 3 bloques diferenciados de 3 preguntas cada uno y un cuarto bloque de preguntas abiertas, opcional. En

primer lugar, tres preguntas identifican al tipo de usuario mediante su rango de edad, su nivel de conocimiento previo de ingeniería del software y su nivel de conocimiento de Scrum. El segundo bloque se centra en la valoración de la diversión y el aprendizaje sobre Scrum del juego. El tercero se enfoca a aspectos técnicos del juego: dificultad, comprensión de mecánicas y valoración de la interfaz de usuario. Por último, en el bloque de preguntas abiertas, añadimos espacio para que pudieran hacer sugerencias sobre los aspectos que se preguntaban brevemente en anteriores bloques.

Con la primera interacción, 8 usuarios rellenaron el cuestionario. Esto nos permitió descubrir algunos bugs y que algunos conceptos de Scrum no quedaban del todo claros, a pesar de que servían de introducción a ellos, sin embargo, 7 de los 8 usuarios pensaban que el juego era útil para iniciarse a Scrum. Por ello, nos centramos en mejorar las explicaciones internas del juego y de Scrum y pulir los bugs encontrados.

De nuevo, preparamos una nueva encuesta. Esta segunda encuesta es idéntica a la anterior, salvo por una pregunta para identificar si los usuarios probaron la versión previa del juego. Decidimos ampliar el rango de usuarios piloto para poder hacer una muestra fina más significativa. Tenemos constancia de que más de 10 usuarios probaron la nueva versión del juego. Sin embargo, tan solo uno respondió al cuestionario propuesto, por lo que no podemos sacar conclusiones claras. Es por ello que no podremos incluir los resultados en esta sección.

Actualmente, el proyecto cuenta con explicaciones sobre las diferentes ceremonias y conceptos de Scrum, ayudas para entender el funcionamiento de los ditintos menús y, además, un videotutorial en el menú principal, por lo que esperamos que esta versión sea aún mejor para introducirse a Scrum.

## 14.4 Trabajo futuro

Este proyecto expresa bien lo que queremos lograr, y funciona como una introducción sólida a Scrum. Sin embargo, hay aspectos mejorables, a nivel de jugabilidad, entretenimiento, formación y paralelismo con la realidad. A continuación se hacen algunas propuestas a realizar de trabajo futuro:

- **Ampliación de las pruebas:** Como explicamos en el capítulo sobre las pruebas realizadas, por falta de tiempo y conocimiento solo pudimos llevar a cabo pruebas de aceptación. Lo primero que deberíamos hacer, de cara al trabajo futuro, sería la ampliación de la cobertura de las pruebas, estudiando cómo implemen-

tar en Unity los diferentes tipos de pruebas y llevándolas a cabo. De esta manera, podremos extender el juego de manera más segura con las siguientes funcionalidades descritas en esta sección.

- **Sprint Retrospective:** La Sprint Retrospective es claramente la ceremonia peor representada de todas. Aunque tiene una funcionalidad sólida en el juego, no llega ni a expresar bien del todo su utilidad en la realidad ni expresa bien los pasos que normalmente se siguen durante una Sprint Retrospective. Algunas ideas que tenemos para mejorar esta sección son:

  - Ampliar el número de misiones que existen y modificar las ya existentes para que sean más coherentes tanto con el propósito de una Sprint Retrospective como con el resto de elementos del juego. En general ahora mismo da la sensación de que la Sprint Retrospective es la reunión menos importante y destacable en el juego, y es porque no encaja bien con el resto de reuniones, podría eliminarse y el juego funcionaría exactamente igual.
  - Mostrar las listas de aspectos que fueron bien, fueron mejorables o fueron mal. En base a esto, tenemos dos alternativas:

    1. Nuestro programa las rellena con consejos para mejorar el próximo sprint, y el jugador se basa en estos consejos para hacer su selección de misiones.
    2. El jugador rellena su contenido con aquello que él vea apropiado. Tenemos 3 propuestas para este caso:
       - El juego ofrece un distinto conjunto de misiones en función de qué haya puesto el jugador.
       - El juego premia al jugador que hace una retrospectiva coherente y penalice al que hace una retrospectiva incoherente.
       - Que esta lista sirva simplemente para que el jugador se acostumbre a anotar su opinión del Sprint y tomar decisiones en base a ello.

- **Ley de Parkinson[23]: el trabajo se expande hasta llenar el tiempo disponible para que se termine:** Este es un fenómeno que existe en los entornos de trabajo. Aunque hay numerosas propuestas para combatirlo, no es esa nuestra meta, sino que queríamos expresar la existencia de este fenómeno para aumentar el peso de la decisión de determinar el tiempo que debe durar un Sprint. Se nos ocurrieron varias formas de mostrar esta ley:

- Reducir la productividad de los personajes si aún queda mucho tiempo de Sprint y quedan pocas tareas. Esencialmente así penalizamos al jugador que prefiere ir sobre seguro con el tiempo.
- Aumentar la productividad del equipo si queda poco tiempo para completar el Sprint y aún quedan bastantes tareas. Similar al anterior, pero en lugar de penalizar, premiamos un Sprint bien medido. Este nos gusta menos, porque más que incentivar a medir bien los Sprints, va a incentivar que los jugadores hagan Sprints cortos.
- Hacer que se generen tareas falsas de distracción si queda mucho tiempo de Sprint y quedan pocas tareas, representando la procrastinación de los trabajadores. Estas tareas obstaculizan la realización del Sprint.

Hay una dificultad común entre estas propuestas, que es que hay que encontrar una buena forma de medir lo que queda de Sprint tanto en número de tareas como en número de turnos. De momento nuestra propuesta para métrica es reutilizar la que ya se usa en el gráfico burndown. El problema de utilizar esa misma métrica es que esencialmente estamos penalizando al jugador que consiga mantenerse por debajo de la línea ideal del gráfico, y de esa forma al final estamos incentivando a que el jugador sea productivo, pero no en exceso. Parte del trabajo futuro de esta propuesta es encontrar la mejor forma de implementarlo, que es donde consideramos que reside la mayor dificultad de esta propuesta.

- **Calendario Niko-Niko:** Tal y como está implementado ahora mismo, los estados del calendario niko-niko funcionan muy bien; son útiles y se deben tener en cuenta, además añade un factor aleatorio que puede romper las fórmulas óptimas que encuentre el usuario. Sin embargo, no hay un motivo para revisar el calendario en sí, ya que los estados de turnos anteriores no son relevantes para ningún elemento en el juego.

  Para aumentar la relevancia de este elemento, hemos pensado en implementar un nuevo tipo de misión en la Sprint Retrospective relacionado con el calendario niko-niko, por ejemplo, una misión de evitar que haya una cara triste en X turnos. Para que dependa menos de la suerte, esto irá acompañado de una nueva habilidad de Scrum Master que disminuya la probabilidad de que hayan caras tristes. De esta forma, el jugador que haya escogido una misión de este tipo sí revisará el calendario niko-niko de vez en cuando para ver cómo progresa en su misión.

- **Gráfico Burndown:** El gráfico burndown es otro elemento que pasa desapercibido durante el juego. Similar al calendario niko-niko, proponemos crear misiones

que incentiven el revisarlo durante la Daily Scrum. Otra forma que hemos pensado para enfrentar este problema es hacer que en función de cómo esté el gráfico burndown, la productividad de los personajes cambie (§14.4), por ejemplo, reduciendo la productividad de los personajes si el gráfico se aleja de la línea ideal, o alterando la fórmula de probabilidad de los estados niko-niko.

- **Puntos de historia de las tareas:** Por lo que hemos visto, en nuestro producto existe una cantidad de puntos de historia por tarea que suele funcionar mejor (5∼7 suele ser lo óptimo). A priori esto no parece mal, ya que parece que enseña bien a que no está bien hacer tareas demasiado fáciles ni tampoco tareas demasiado difíciles en una situación real, pero somos conscientes de que tampoco existe una fórmula ideal, muchas veces estimamos mal y una tarea es más difícil de lo que parece, o más fácil, o simplemente hay tareas que no se pueden dividir más, o hay veces que al dividir tanto una tarea al final hay que hacer esfuerzo extra para integrar todas las partes de la tarea.

  Pensamos que no hace falta hacer una correlación perfecta entre la gestión de tareas real y la que hay en nuestro producto, pero desde luego nos gustaría que fuese más fiel a la realidad.

  Ante este problema se nos ocurren las siguientes propuestas de solución (y no son necesariamente excluyentes, es decir, podrían aplicarse varias simultaneamente):

  - Crear misiones en la Sprint Retrospective que recompensen al jugador que se enfrenta a tareas con más puntos de historia, para incentivar que el usuario se enfrente a tareas más difíciles.
  - Poner un límite inferior de Puntos de Historia para cada enemigo, para representar que no siempre podremos dividir una tarea indefinidamente, y que hay veces que hay que enfrentarse a tareas difíciles sin poderse simplificar más.
  - Crear una tarea nueva de integración si se subdividen demasiado las tareas. Esta es la más fiel a la realidad, pero el problema está en que su implementación podría ser compleja y además podría complicar aún más la experiencia del usuario.

- **Alteración de requisitos:** Este es un problema recurrente en proyectos software, y es uno que justamente las metodologías ágiles tratan de solventar, por lo que creemos que es esencial que esto se exprese de alguna forma en nuestro software porque destacaría una vez más la utilidad de Scrum.

La propuesta de mejora que hacemos aquí es que en función de cómo se avance en el proyecto, el puede añadir nuevas tareas al Product Backlog al finalizar el Sprint Review. En principio querríamos que esta expansión sólo se le ofreciera al jugador que va muy bien. Estas tareas se añadirían al Product Backlog, pero no haría falta completarlas para que se considere que el proyecto está terminado. A nivel de jugabilidad esto puede estar bien también, ya que el jugador adapta la dificultad del juego a su habilidad[17].

Si estas tareas extras se cumplen, tendríamos que reflejarlo de alguna forma. Lo que hemos planteado es que al final del proyecto se asigne una puntuación en función de lo bien que se haya hecho, y que esta puntuación sólo puede alcanzar su valor máximo si se completan todas las tareas opcionales, incentivando una toma de decisión basada en evaluando el riesgo frente a la recompensa.

Otra forma de expresar los cambios de requisitos podría ser haciendo que el acorte el tiempo de desarrollo del proyecto, aumentando la presión del jugador. Esto también pasa en proyectos software reales, y vemos que es importante cubrir este caso.

- **Ampliación de Scrum Master:** Tal y como está implementado, el Scrum Master se limita a hacer acciones extras que incrementan la motivación del equipo, pero en la realidad esa no es su única labor. Estas son las responsabilidades de un Scrum Master [33]:

  - Forma al equipo para autogestionarse y en la funcionalidad cruzada.
  - Ayuda al equipo a generar incrementos adecuados durante el Sprint.
  - Eliminar obstáculos e impedimentos que tenga el equipo.
  - Asegurarse de que se cumplen bien todos los eventos de Scrum.
  - Ayuda a crear un objetivo de producto y Product Backlog adecuados.
  - Ayudar al equipo a entender la importancia de la claridad del Product Backlog.
  - Ayudar a establecer una planificación de producto empírica en el entorno de trabajo.
  - Facilitar la colaboración de interesados.
  - Liderar y formar al equipo para Scrum.
  - Planear e incentivar implementaciones de Scrum.

- Ayudar a entender la importancia de un punto de vista empírico para trabajo complejo.
- Favorecer la comunicación entre el equipo y los interesados.

En resumen, el Scrum Master:

1. Vela por el bienestar del equipo y que estén correctamente formados para aplicar Scrum.
2. Se asegura de que se aplica Scrum correctamente y que todos entiendan su importancia.
3. Actúa como intermediario entre los interesados, el y el equipo de desarrollo, facilitando la comunicación entre ellos.

En nuestro producto reflejamos (y de forma imperfecta) la primera afirmación (1). La segunda (2) es responsabilidad del jugador, y es lo que marca su aprendizaje, y la tercera (3) es computada automáticamente por el juego.

De base, nuestro producto es incapaz de representar bien las responsabilidades del Scrum Master, ya que parte de ellas son imposibles de representar. Nuestro producto no busca formar a alguien en ser Scrum Master, sino en entender Scrum y recibir una introducción a él, así que dimos más importancia a que fuese el propio jugador el responsable de que se aplique Scrum correctamente, al mismo tiempo que el rol de Scrum Master no recae completamente sobre él.

Con esto aclarado, nuestra propuesta de trabajo futuro es ampliar sólo la primera (1) de las tres afirmaciones mencionadas antes. Queremos incluir acciones de Scrum Master que representen otras de sus responsabilidades, como son el formar a los demás miembros, habilitar mejores condiciones de trabajo, asegurarse de que las reuniones se hagan en el mejor entorno posible, etc. Algunos ejemplos de ampliación que tenemos en mente son:

- **Mejorar Sprint Review:** Aumenta la motivación recibida durante la próxima Sprint Review y disminuye las penalizaciones.
- **Persuadir :** Reduce la exigencia del PO al hacer la revisión del proyecto, haciendo que sea más fácil hacer que el esté satisfecho con el avance del equipo.
- **Mantenimiento del entorno:** Mejora la productividad del equipo durante los próximos X turnos.

  - **Team building:** Aumenta la motivación y moral de todo el equipo al final del turno, y garantiza que no habrán caras tristes el día siguiente, pero disminuye la energía de todo el equipo al final del turno y la productividad del turno en el que se hace esta acción baja.

- **Mejoras a la interfaz de usuario:** Algunas partes de la interfaz de usuario son imperfectas, especialmente en el Daily Scrum. Hay múltiples mejoras que podrían aplicarse, por ejemplo seleccionar el personaje o tarea al que se le quiere aplicar una acción tocando su sprite, representar mejor la cantidad de "vida"que le queda a una tarea...

# Parte VII

# Apéndices

# 15

# Apéndices

## 15.1 Mock-ups

En esta sección, distinguiremos dos tipos de Mock-ups: los mockups que se hicieron antes de comenzar a trabajar en el proyecto y los que se crearon durante el transcurso del trabajo de fin de grado.

### 15.1.1 Mock-ups antiguos

Los primeros mockups vienen originados porque, a la hora de explicar nuestra propuesta a los tutores, quisimos acompañarlo de mockups simples. Los elaboramos de manera sencilla mediante RPGMaker [10], herramienta que consideraríamos más adelante como una de las opciones de tecnologías a usar.

Algunos de estos mockups, a día de hoy, no se pueden considerar, ya que son ideas que quedaron descartadas al poco de comenzar el proyecto. Sin embargo, sus descartes sirvieron para determinar mejor la apariencia final que queríamos obtener.

La principal razón por la que muchos fueron descartados fue porque, a la hora de presentar la propuesta, teníamos la idea de que Sprint Planning, Sprint Review y Sprint Retrospective. transcurrieran en un mapa en el que se pudiera ver e interactuar con el equipo. Fue posteriormente, cuando iniciamos la implementación, cuando decidimos realizar estas ceremonias mediante interfaces.

El mockup más parecido a la idea final fue el de la interfaz de combate que, si bien no se muestra en la misma disposición que el resultado final, nos sirvió como guía inicial de los elementos que buscábamos mostrar durante la Daily Scrum. Las principales diferencias en cuanto a la información sobre los combatientes y los menús son las siglas utilizadas para definir la magia y la vida, que no se muestran las caras del calendario niko niko junto a los nombres y la ausencia del burndown.

Otro mockup que puede todavía ser utilizado, en parte, sería el que muestra la break room, idea que podría ser considerada para el trabajo futuro, que serviría para

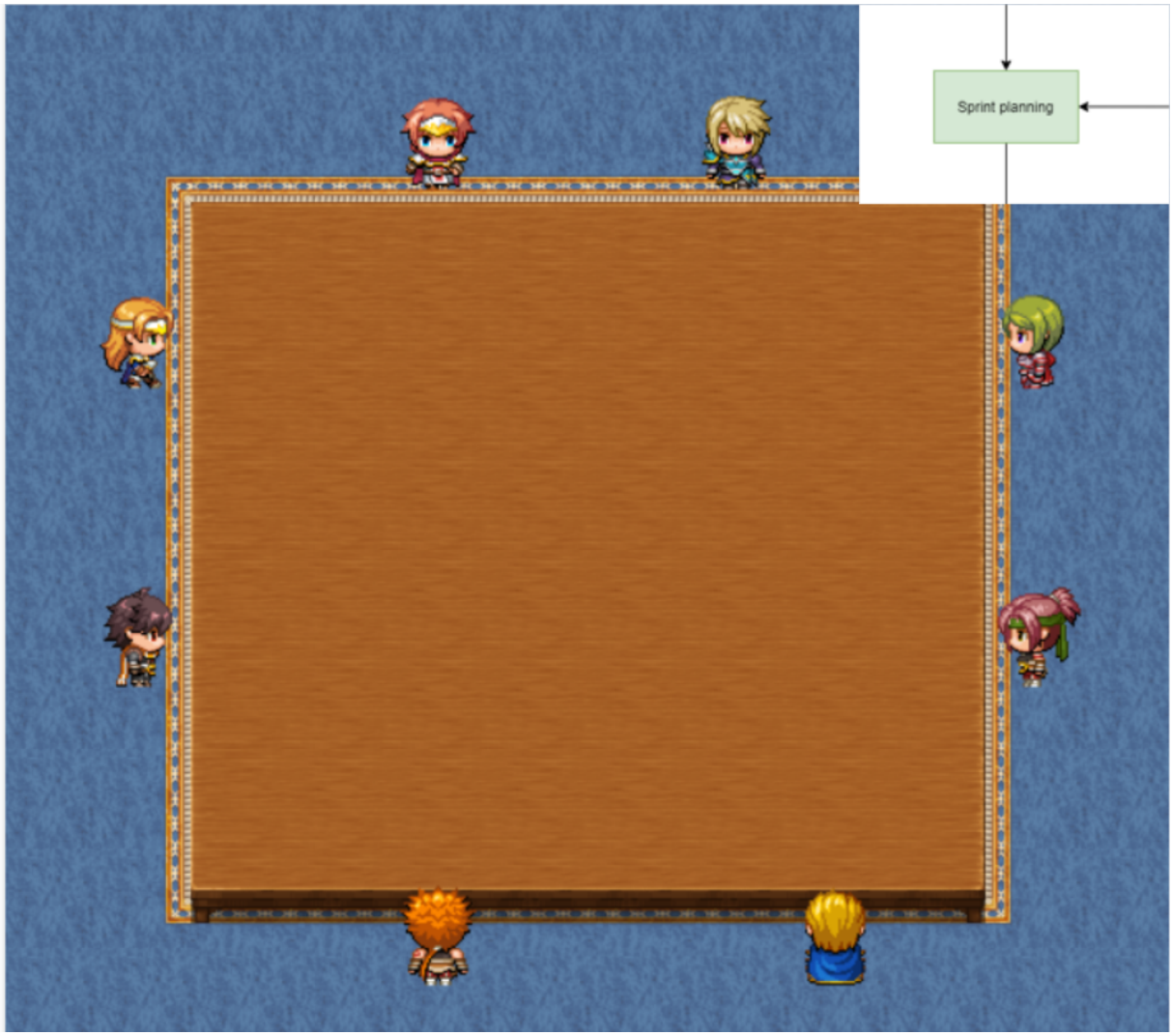



Figura 15.1: Mockup de Sprint Planning sobre el mapa

interaccionar con los personajes y crear consecuencias de estas interacciones durante la Daily Scrum.

### 15.1.2 Mock-ups desarrollados durante el proyecto

Durante el proyecto, como la interfaz de combate venía en parte estructurada por ORKFramework, el espacio de la interfaz disponible para añadir los detalles que no se incluían en esta herramienta estaba limitado. Como también teníamos una idea clara de los elementos a mostrar, no fue necesario hacer mockup.

Como la idea de realizar las ceremonias de Sprint Planning, Review y Retrospective surgió durante el transcurso del proyecto, se hicieron mockups de las versiones

iniciales de los menús, que cambiarían más adelante para adecuarnos mejor a las necesidades del juego.

## 15.2 Manual de usuario

Para la instalación del proyecto, basta con descargar el archivo "ScrumRPG.apk" e instalarlo. Recordatorio de que sólo es compatible con dispositivos Android, no con iOS, Linux, Windows o macOS.

En el propio proyecto se incluye el manual de usuario. Está en forma de un vídeo en la pantalla de inicio y luego en cada una de las escenas del juego podemos encontrar un icono "?" que nos permitirá obtener información de dicha escena en particular.

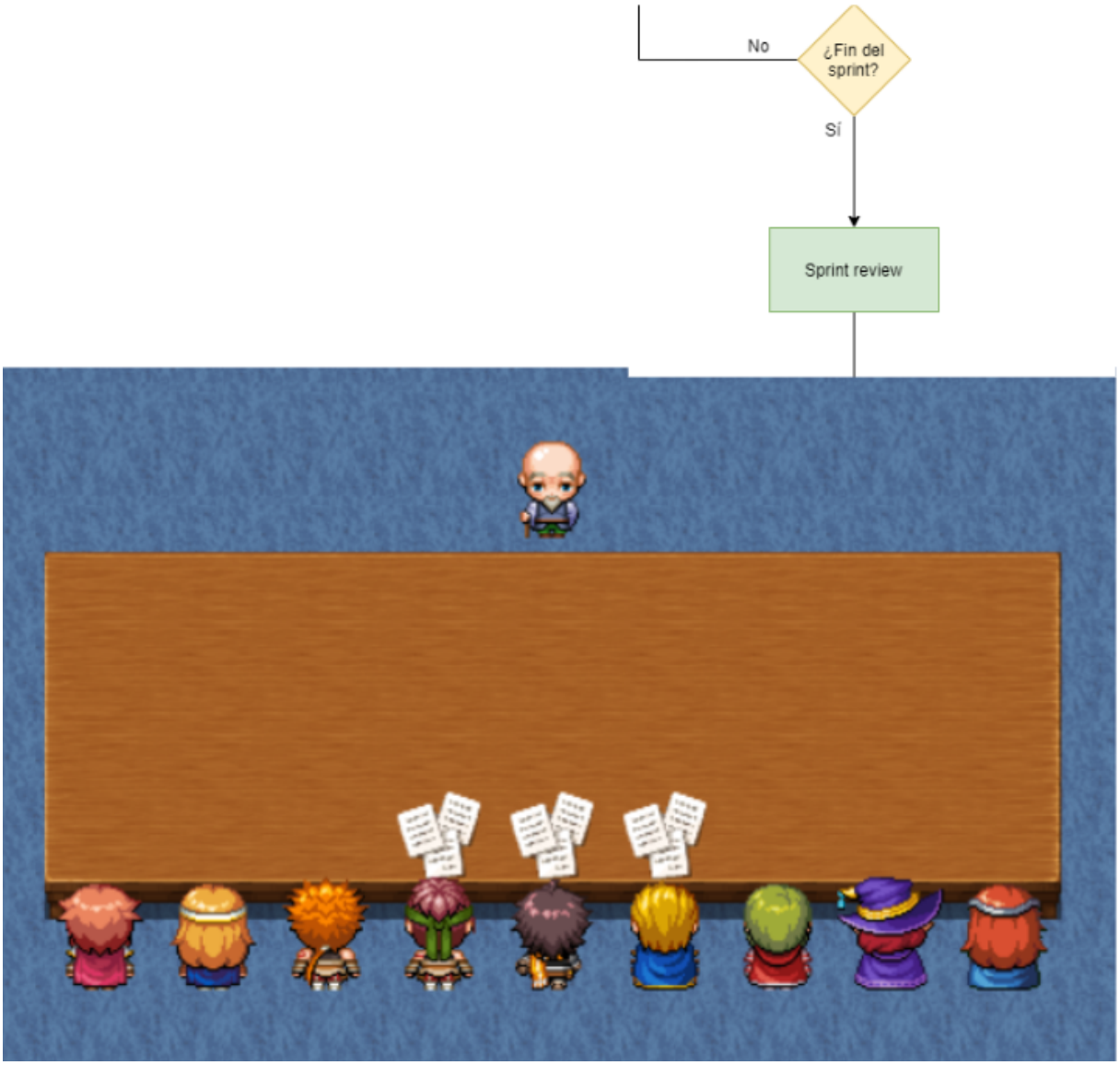



Figura 15.2: Mockup de Sprint Review sobre el mapa

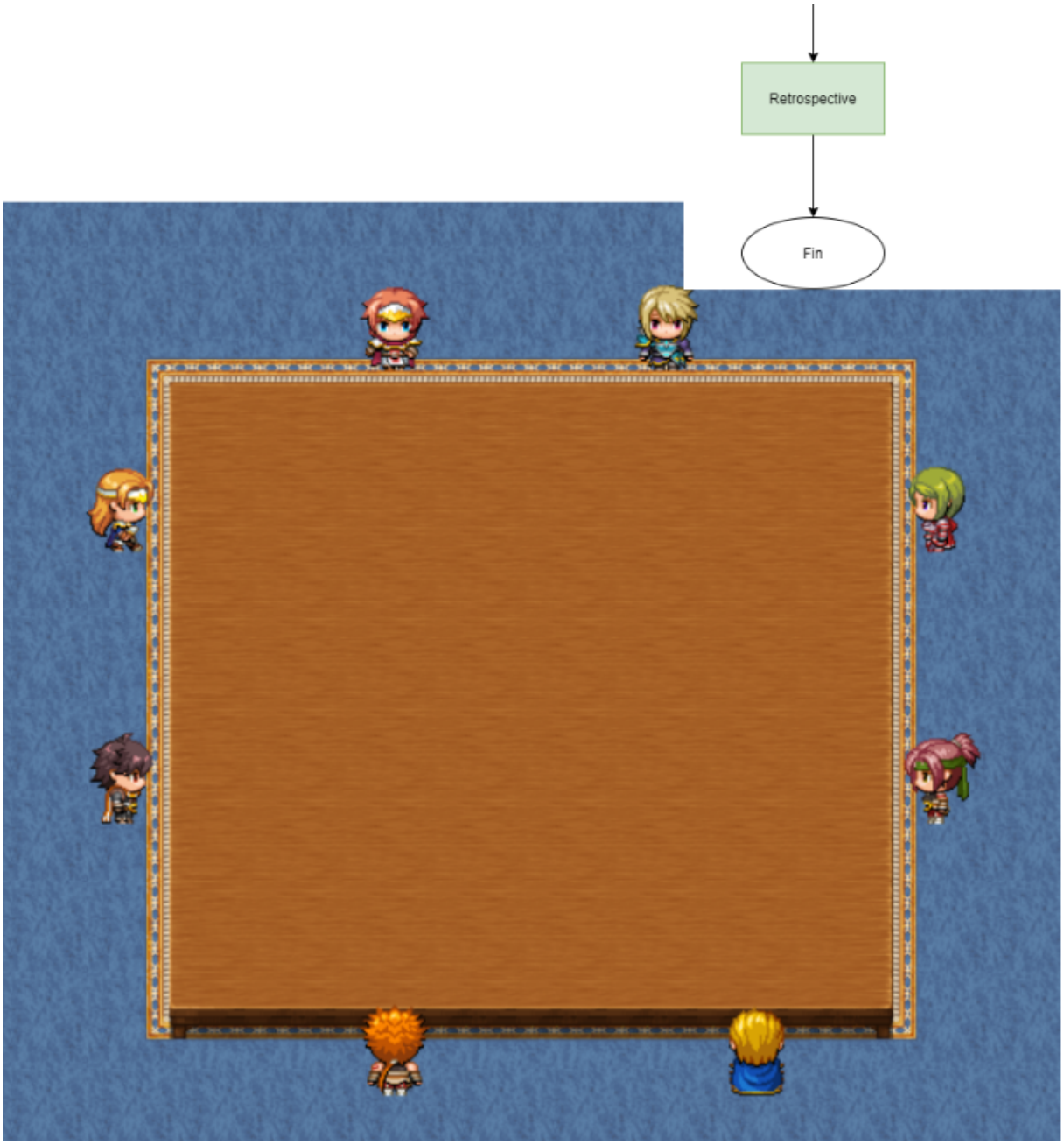



Figura 15.3: Mockup de Sprint Retrospective sobre el mapa

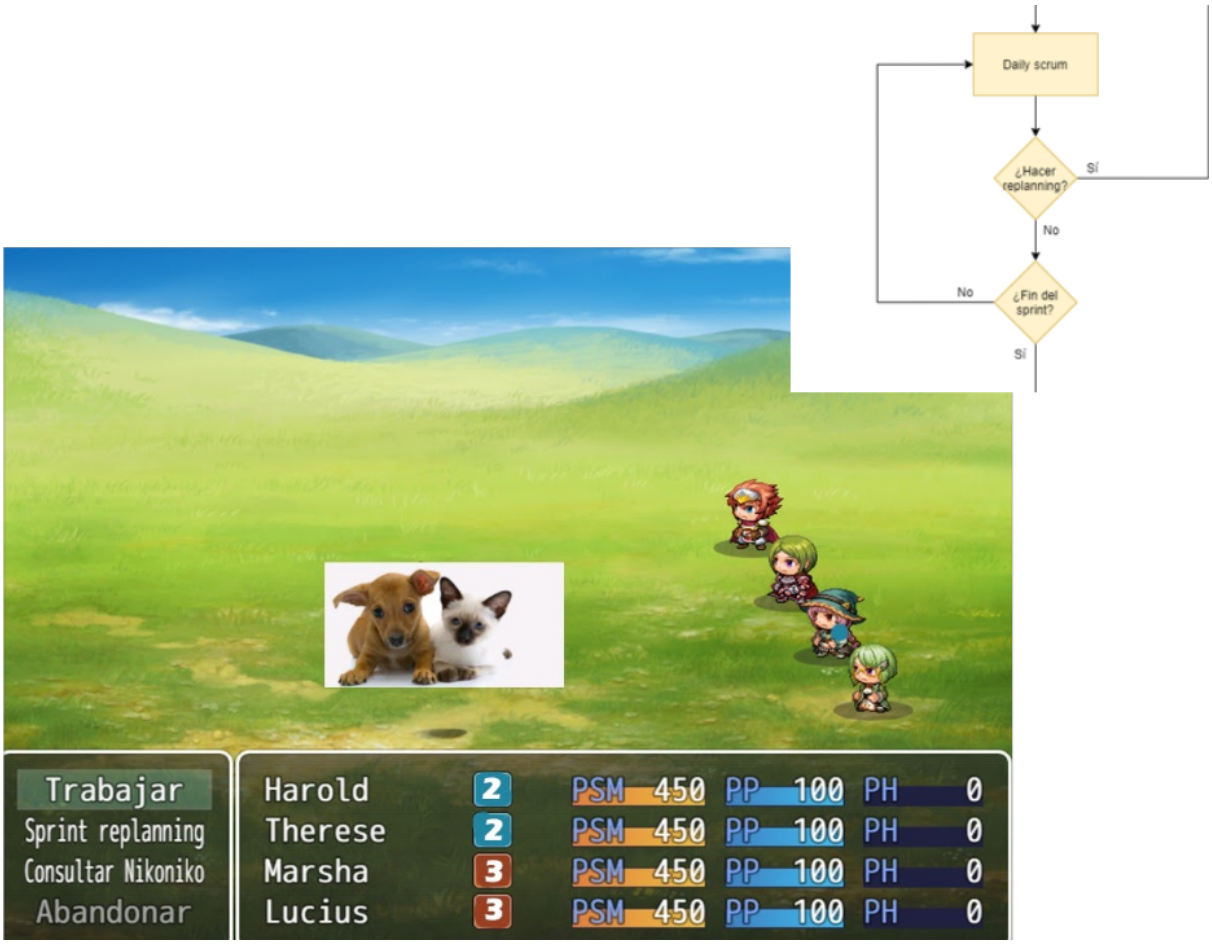



Figura 15.4: Mockup de idea inicial de Daily Scrum

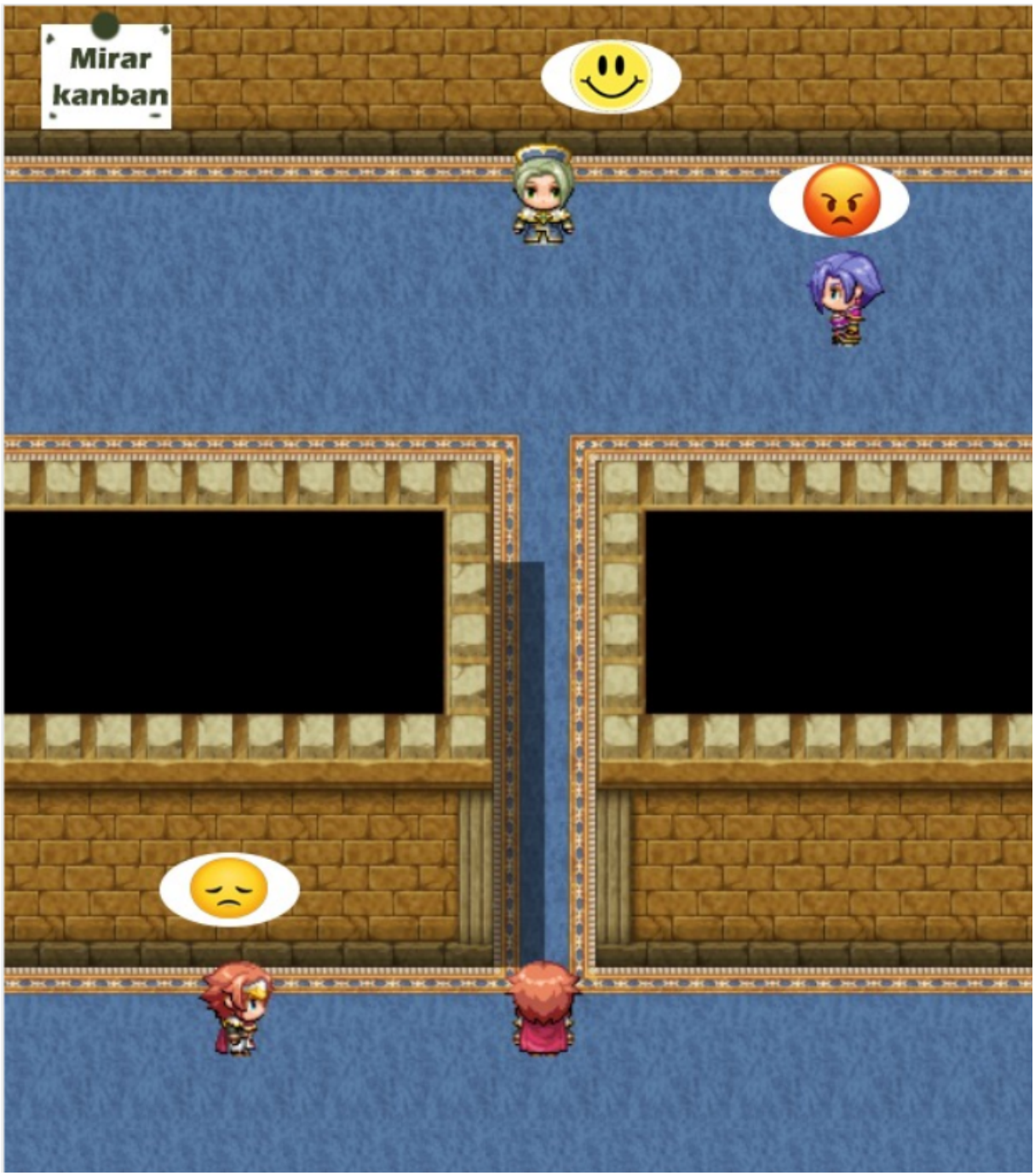



Figura 15.5: Mockup de idea inicial de Break Room

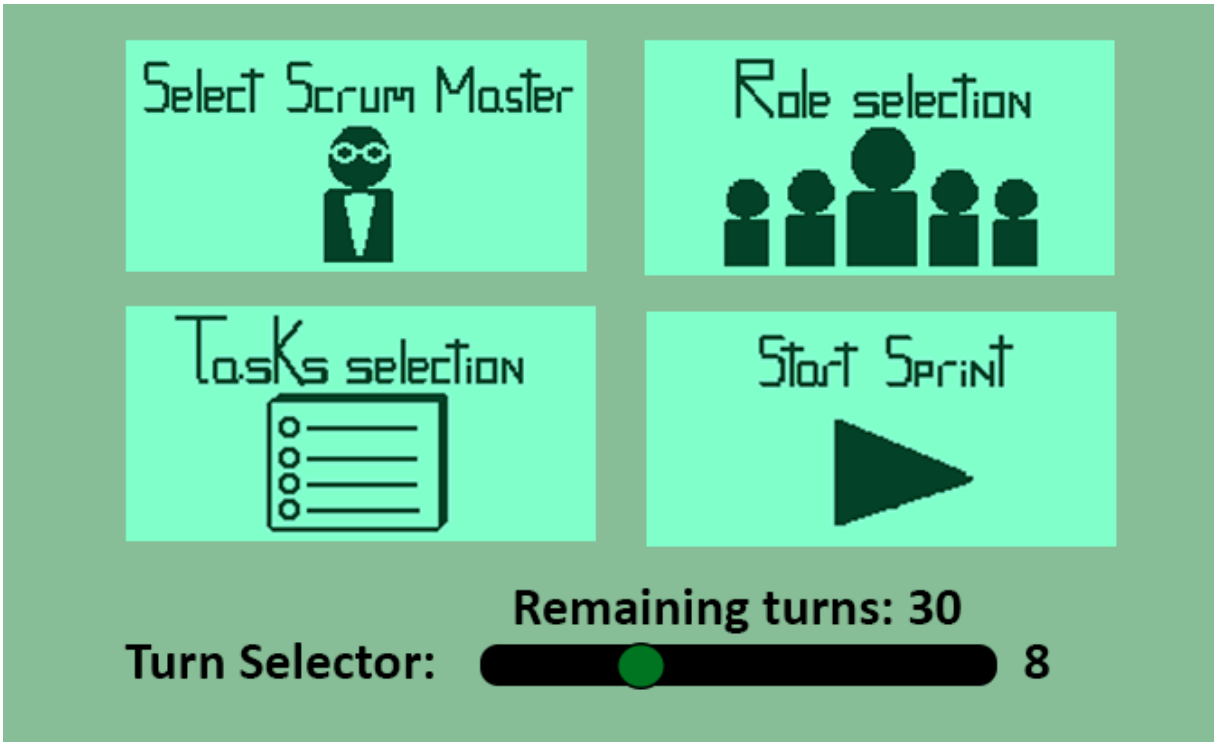



Figura 15.6: Mockup general del Sprint Planning

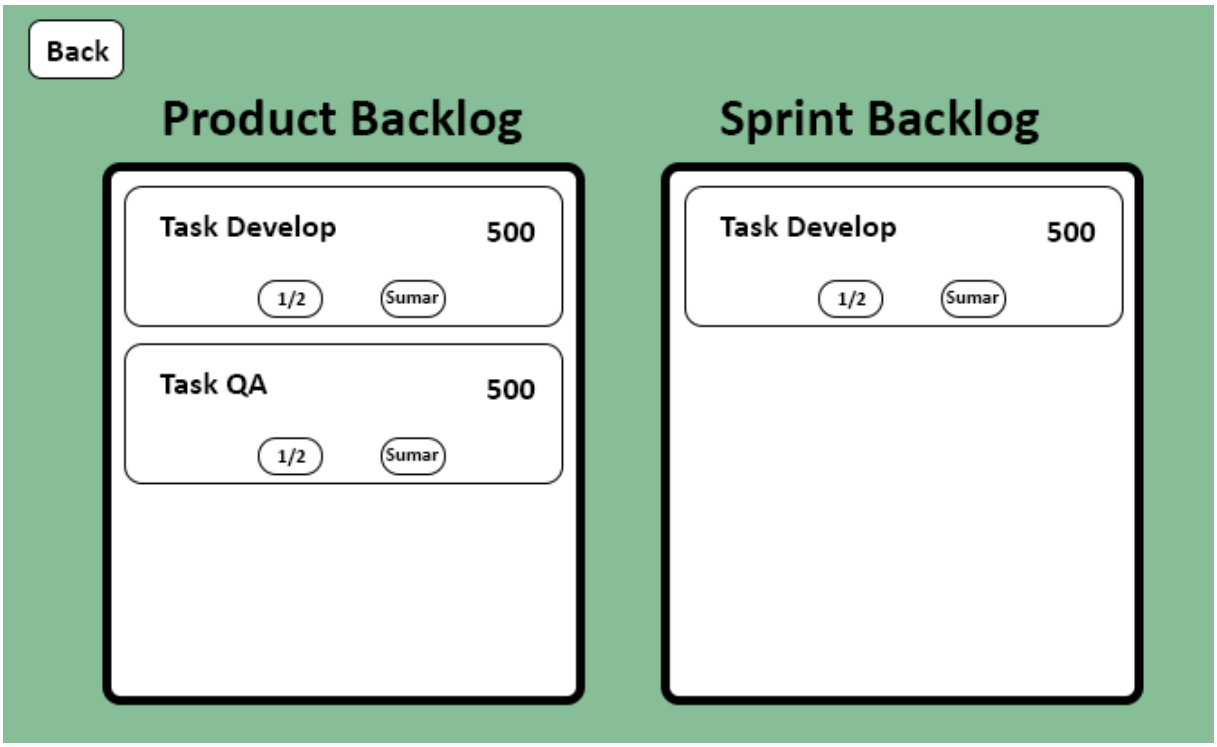



Figura 15.7: Mockup del menú de selección de tareas

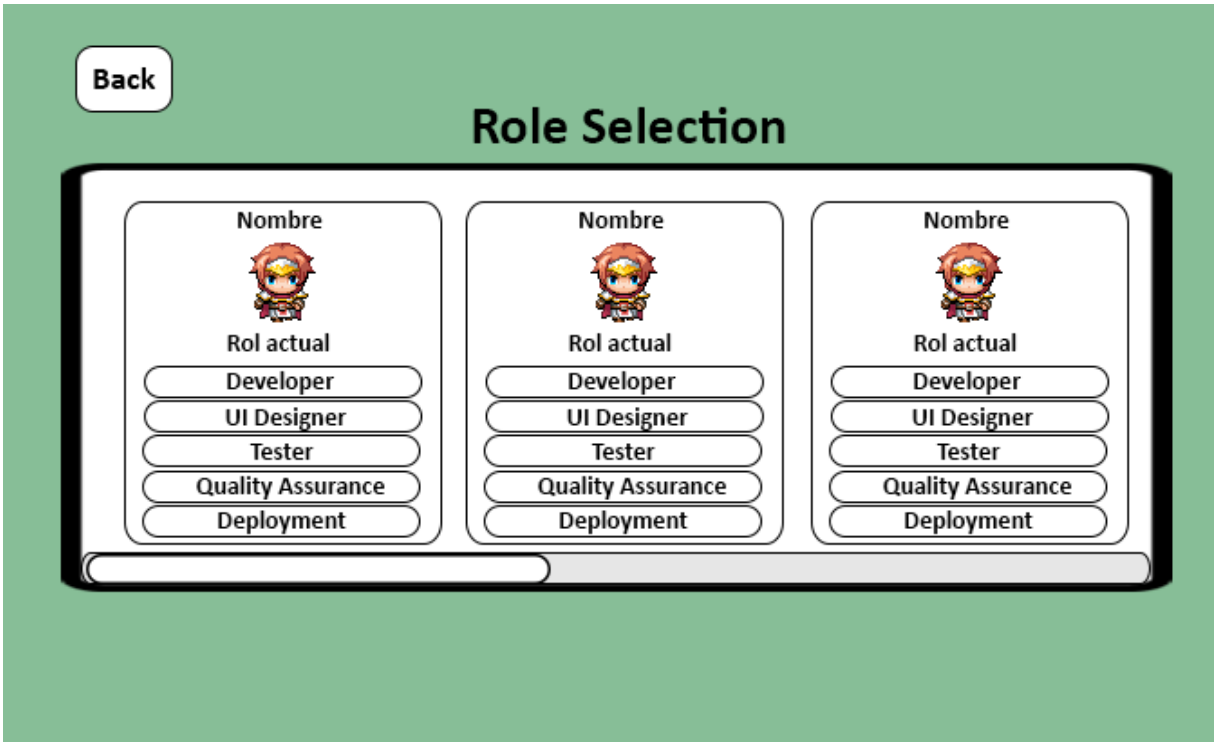



Figura 15.8: Mockup del menú de elección de roles

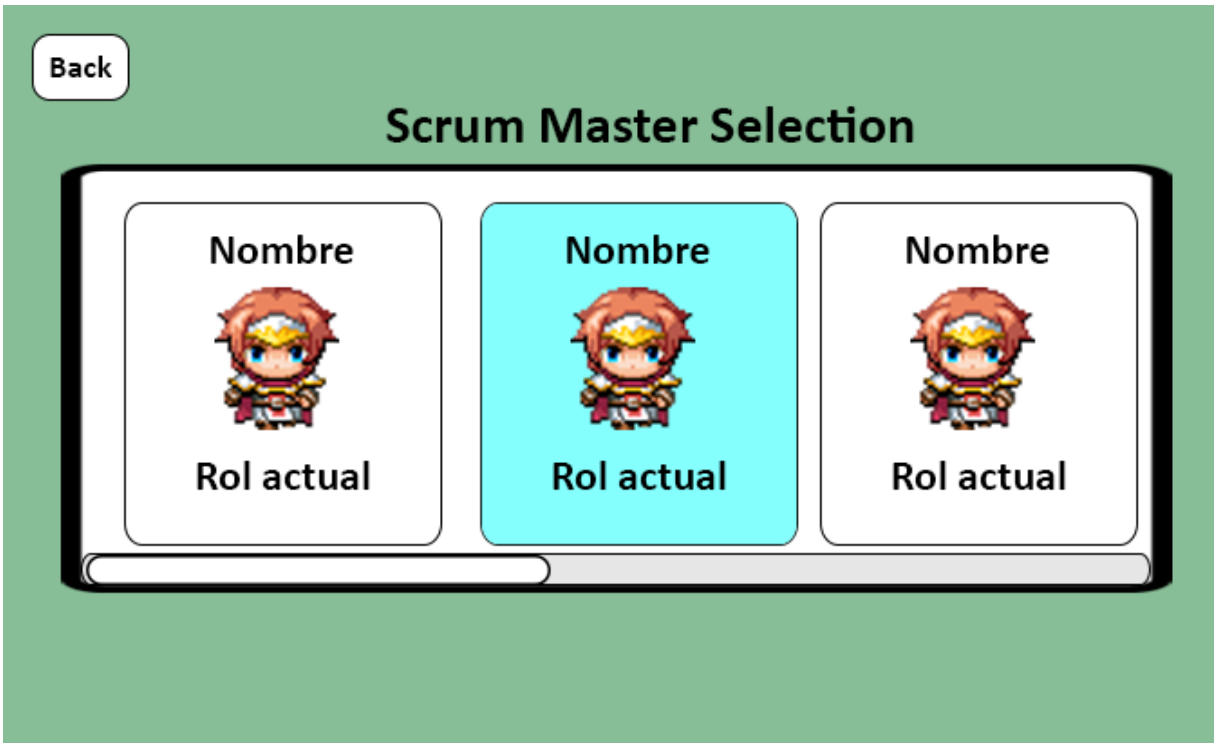



Figura 15.9: Mockup del menú de elección de Scrum Master

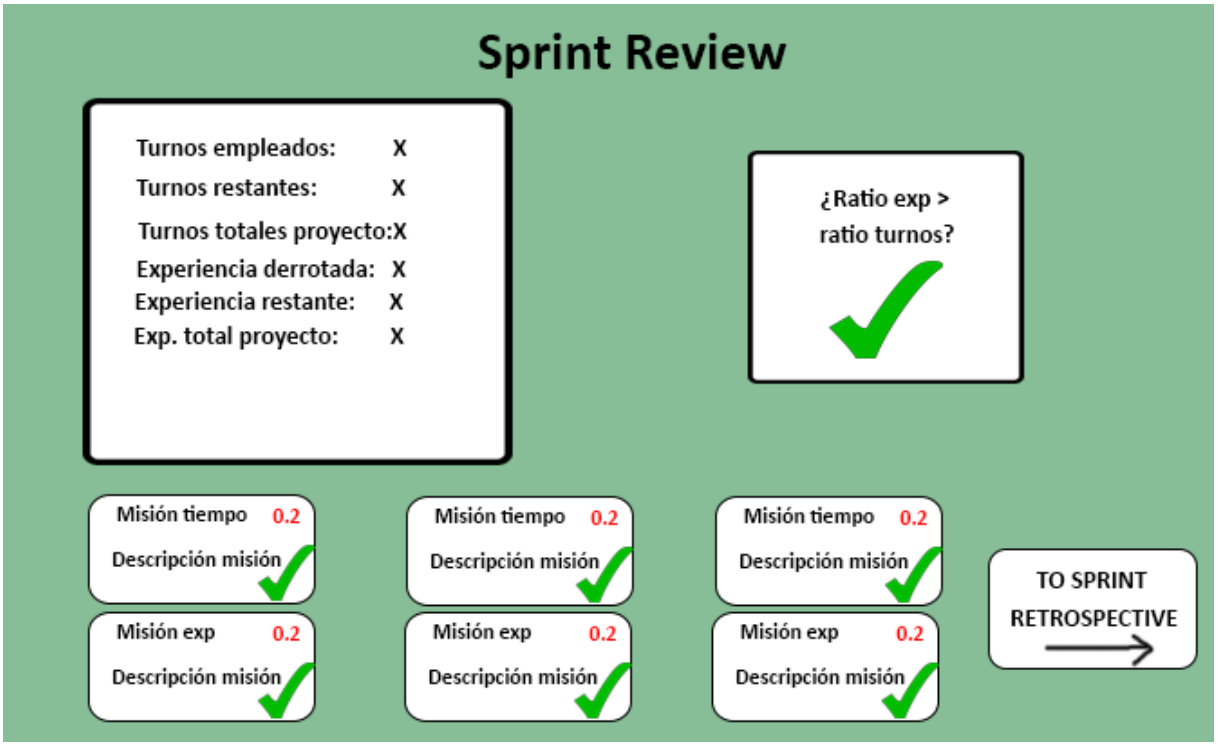



Figura 15.10: Mockup del menú de elección de Sprint Review

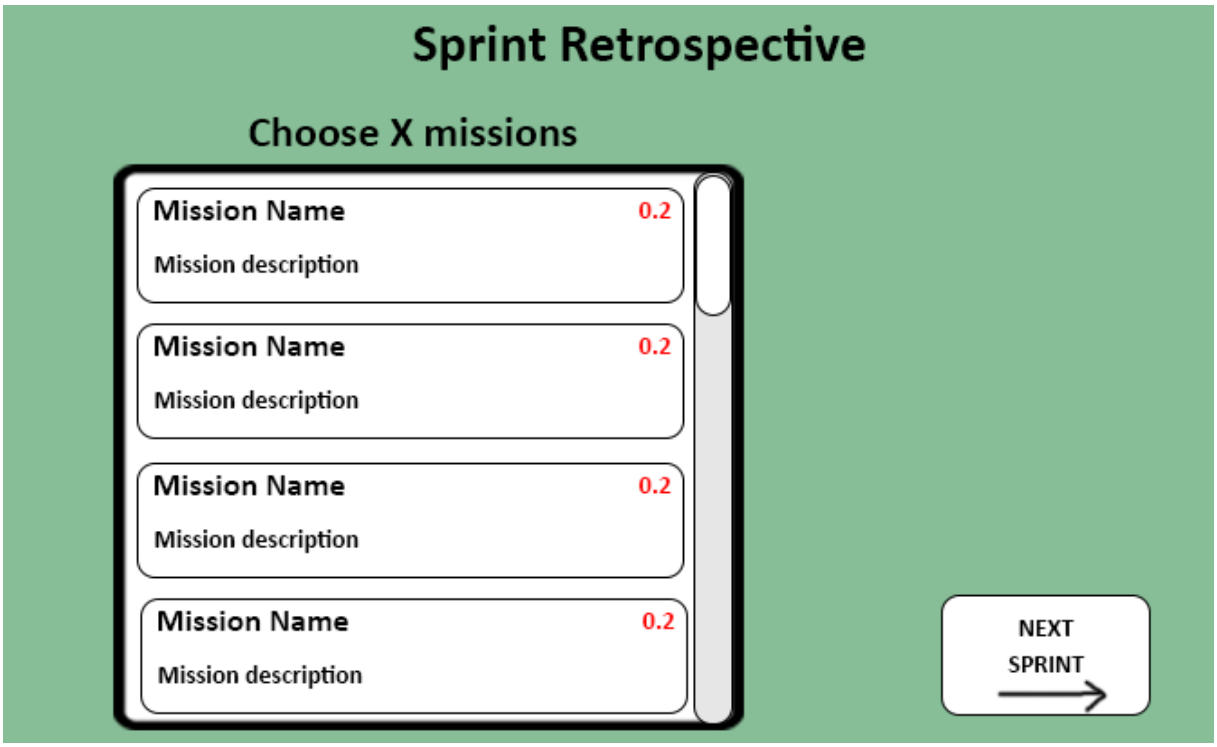



Figura 15.11: Mockup del menú Sprint Retrospective

# BIBLIOGRAFÍA


[1] 2d rpg kit. URL https://assetstore.unity.com/packages/templates/packs/2d-rpg-kit-163910. (page 72).

[2] Unity asset store. URL https://assetstore.unity.com/. (page 71).

[3] Blacksmith: Action 2d rpg engine / starter kit. URL https://assetstore.unity.com/packages/tools/game-toolkits/blacksmith-action-2d-rpg-engine-starter-kit-149631. (page 71).

[4] Certified scrummaster. URL https://www.scrumalliance.org/get-certified/scrum-master-track/certified-scrummaster. (page 14).

[5] Godot. URL https://godotengine.org/features. (page 69).

[6] Orkframework, . URL https://assetstore.unity.com/packages/tools/game-toolkits/rpg-editor-ork-framework-2-125912. (page 72).

[7] Página oficial de orkframework, . URL https://orkframework.com/. (page 72).

[8] Pavcreations. URL https://pavcreations.com/turn-based-battle-and-transition-from-a-game-world-unity/. (page 73).

[9] Rpg builder, . URL https://assetstore.unity.com/packages/tools/game-toolkits/rpg-builder-177657. (page 71).

[10] Rpg maker, . URL https://www.rpgmakerweb.com/products/rpg-maker-mz20/10. (pages 68 y 195).

[11] Scrum knowsy, . URL https://www.scrumknowsy.com/. (page 12).

[12] Scrum pocket traning, . URL https://mariankoenig.com/apps/scrum-pocket-training.html. (page 12).

[13] Turn-based rpg battle engine 2d. URL https://assetstore.unity.com/packages/tools/game-toolkits/turn-based-rpg-battle-engine-2d-135496. (page 72).

[14] Unity. URL https://docs.unity3d.com/Manual/index.html. (page 69).

[15] Agile manifesto. URL https://agilemanifesto.org/. (page 62).

[16] Challenges in scrum. URL https://communities.bentley.com/cfs-file/__key/communityserver-discussions-components-files/5917/Issues_2D00_and_2D00_Challenges_2D00_in_2D00_Scrum_2D00_Implementation_2800_1_2900_.pdf. (pages 5, 6, 60, 61 y 63).

[17] Controlling flow. URL http://whats-in-a-game.com/controlling-flow. (page 189).

[18] C#. URL https://learn.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/. (pages 70 y 74).

[19] 4 gamified solutions for agile scrum training. URL https://www.gamification.co/2014/07/08/4-gamified-solutions-for-agile-scrum-training/. (page 12).

[20] Is scrum difficult? URL https://www.scrum.org/forum/scrum-forum/31395/scrum-difficult. (pages 5 y 60).

[21] Lego4scrum. URL https://www.lego4scrum.com/. (page 12).

[22] Canvas, manual de usuario de unity. URL https://docs.unity3d.com/es/530/Manual/UICanvas.html. (page 118).

[23] Parkinsons law. URL http://www.unilanguage.ru/_ld/0/80_read_book.pdf. (pages 61 y 186).

[24] Metodología scrum, . URL https://www.scrum.org/. (page 20).

[25] Scrum guide, . URL https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100. (page 61).

[26] The stages of scrum mastery. URL https://www.scrum.org/resources/blog/stages-scrum-mastery. (page 60).

[27] Tactical role-playing game. URL https://en.wikipedia.org/wiki/Tactical_role-playing_game. (page 177).

[28] Gestión de equipos. URL https://hdvirtual.us.es/discovirt/index.php/s/dLreApZw7n4SEbD#pdfviewer. (page 61).

[29] The scrum game. URL http://alecoledelavie.com/accueil/DossierScrum/6_ScrumGame/Scrum%20Game/Scrum%20Game%20Rules.pdf. (page 12).

[30] Tribal leadership. URL https://www.joostdevree.nl/bouwkunde2/jpgm/management_4_tribal_leadership_the_5_stages.pdf. (page 92).

[31] Visual studio. URL https://visualstudio.microsoft.com/es/. (page 74).

[32] Ways to learn about scrum. URL https://www.scrum.org/resources/ways-learn-about-scrum. (page 12).

[33] What is a scrum master? URL https://www.scrum.org/resources/what-is-a-scrum-master. (page 189).